webMethods Mobile Development Help

Version 9.8

April 2015

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

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

Document ID: DES-MDEV-OLH-98-20150415

Table of Contents

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

Online Information

Software AG Documentation Website

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

Software AG Empower Product Support Website

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

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

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

Software AG TECHcommunity

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

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

1 Introduction to Mobile Development

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

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 30.

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 48.

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 30.

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.

3. To remove a node, right-click the node and select **Delete**.

   Alternatively, you can select the node and press the DELETE key.

Tip: To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see "Generating Sources for a Mobile Project" on page 30.

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

1. Ensure the mobile project or specific window, view, or dialog for which you want to work with properties is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 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.
3. Select the node for which you want to set properties.
4. In the **Properties** section of the Outline Editor, fill in the properties for the selected node.

   For more information about the properties, see "User Interface Objects" on page 80 and "Services Object Reference" on page 129.

   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 141.

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 30.

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 "Generating and Building a Mobile Project" on page 53.

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

Option	Description
	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 to a Mobile Project" on page 50. **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.

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

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

- Adds your system language as a language that your application supports. You can add additional languages and/or remove languages after the project is created. For more information, see "Managing Languages the Application Supports" on page 45.
- Adds several universal devices that your application supports. You can add additional devices and/or remove devices after the project is created. For more information, see "Adding Devices to the Mobile Project" on page 43 and "Removing Devices from the Mobile Project" on page 44.

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* AppControllerImpl.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* AppControllerImpl.java, you need to take further action. For more information, see "Renaming the Application" on page 152.

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.

Error	Solution
Bundle ID has already been taken	The bundle ID identifies an application on the target device and must be unique. Therefore, specify an available bundle ID for the target platform. 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 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.

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 56.
- Define the configuration of panes to use for the application's window. For more information, see "Defining Panes for the Application Window" on page 57.
- 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 61 and "Adding Content to a View" on page 63.
- Add listeners that wait for user-initiated events when a user interacts with controls you add to the view and take an action based on the user-initiated event. For more information, see "About User-Initiated Events and Listeners" on page 72 and "Adding Listeners for User-Initiated Events" on page 73.
- 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 76.

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 31.

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 35.

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.

Package name in the gen/src folder	Description
package_name	This package contains general model-based Java classes.
package_name .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 "Managing Languages the Application Supports" on page 45.
package_name .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 48.
package_name .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.
package_name .ui.dialog	This package contains Java classes that correspond to the dialogs that you designed in the Outline Editor.
package_name .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

Package name in the gen/src folder	Description
	Templates to Define Custom Objects for a Mobile Project" on page 76.
package_name .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
package_name .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 single time. You add your application logic to these Java classes. The Java classes in this package are: ■ *application_name* AppControllerImpl.java In the name of the Java class, *application_name* is the name you assigned to the application. Mobile Development generates one *application_name* AppControllerImpl.java class for your mobile project. 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. ■ *view_name* ControllerImpl.java In the name of the Java class, *view_name* is the name of a view you defined in the Outline Editor. Mobile Development generates one *view_name* ControllerImpl.java class for each view in your model. Add logic specific to a view to this Java class. You can add custom code here that extends the

Package name in the src folder	Description
	generated abstract view controller methods that Mobile Development generates in the Abstract*view_name* Controller.java files, which reside in the gen/src folder in the *package_name* .ui package. For more information about the types of logic to add these Java classes, see "About Adding Application Logic" on page 136
package_name .ui.dialog	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. For each dialog you define in the Outline Editor, Mobile Development generates a *dialog_name* .java class, where *dialog_name* is the name you assigned the dialog in the Outline Editor. 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.
package_name .ui.templates	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 76. For each template you define in the Outline Editor, Mobile Development generates a *template_name* .java class, where *template_name* is the name you assigned the template in the Outline Editor. 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 76.

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 files contains lines for the text strings that you use in a mobile project's view. For example, the files contains a line for the view's **Header Text** property. If you add a **Textfield** object to a view, the file contains a line for the **Textfield** object 's **Text** property.

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

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

```
MASTERVIEW_HEADER_TEXT=Master View
```

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

```
MASTERVIEW_HEADER_TEXT=
```

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

Adding the Mobile Application Logic

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

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

For more information, see "About Adding Application Logic" on page 136.

Defining Resources for the Mobile Project

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

Using the Default Resource Handler

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

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

To use the default resource handler

1. Ensure the settings for your mobile project are set to use the default resource handler.
   a. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 17.
   b. Select the top-level child node of the project, which is the root application node.
   c. Ensure the **Res Handler** property is set to `UniversalResHandler`.
2. Save the resources for your mobile application, for example, audio files and icons. For instructions, see "Storing Resource Files for the Mobile Project" on page 37 and "Storing Image Files for the UniversalResHandler" on page 37.

Storing Resource Files for the Mobile Project

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

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

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

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

If you want to use a different subfolder than the ones provided, for example, if you want to use a subfolder named audio to save sound files, add the custom subfolders to the resource folder and code a custom resource handler for your application. For more information about using custom resource handlers, see "Coding a Custom Resource Handler" on page 42.

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

resources subfolder	Store this type of asset in the subfolder
graphics	Image files that the mobile application uses. These are image files that are larger than an icon. The UniversalResHandler resource handler requires a specific folder structure under the resources/graphics folder. For more information, see "Storing Image Files for the UniversalResHandler" on page 37.
icons	Small image files that the mobile application uses as icons.
text	Text files that contain Strings that the mobile application uses.
www	HTML files that contain web content that the mobile application uses.

Storing Image Files for the UniversalResHandler

When using the UniversalResHandler resource handler, you need to store image files in a specific folder structure. The required folder structure is automatically created for

a project when you create the project using the Mobile Development New Mobile Development Project wizard, as described in "Creating a New Mobile Project" on page 26.

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

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

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

Additionally, each of the main image folders (Android, iOS, WinPhone, and general) contain subfolders themselves. This allows you to supply different image files for different devices within a platform. For example, if a mobile application uses an image file named myimage.png, you might need one version of the image file for an iOS non-retina device and a different version for an iOS retina device. The following table describes the subfolder structure for each main image folder:

Main image folder	Description of its subfolder structure

Android	The subfolders are based on screen density of Android device on which the mobile application runs. The following shows the subfolder structure:
	`Android`
	`  drawable-hdpi`
	`  drawable-ldpi`
	`  drawable-mdpi`
	`  drawable-xhdpi`
	`  drawable-xxhdpi`

iOS	The subfolders are based on the display property of the iOS device, either non-retina or retina. The following shows the subfolder structure:
	`iOS`
	`  NonRetina`
	`  Retina`
	`  RetinaHD`

Main image folder	Description of its subfolder structure

WinPhone	The subfolders are based on the screen resolution of the Windows Phone device and whether background color of the view in which the image is displayed is dark or light. The following shows the subfolder structure:
	`WinPhone`
	`  Dark`
	`  1080p`
	`  720p`
	`  WVGA`
	`  WXGA`
	`  Light`
	`  1080p`
	`  720p`
	`  WVGA`
	`  WXGA`

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

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

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

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

Extending the UniversalResHandler to Allow Storing Image Files in Custom Subfolders

If you want to use the basic functionality of the UniversalResHandler resource handler, but want to use additional subfolders for the image files, you can create a custom resource handler that extends the UniversalResHandler resource handler. For example, suppose in addition to the standard iOS platform subfolders, which are NonRetina and Retina, you also want to use RetinaIPhone4 and RetinaIPhone5. In this case, you can create a custom resource handler the extends the UniversalResHandler resource handler and includes the logic to handle the new subfolders.

To extend the UniversalResHandler to support additional image subfolders

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

   The custom logic should:

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

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

   - resources/graphics/iOS/RetinaIPhone4/
   - resources/graphics/iOS/RetinaIPhone5/
   - resources/graphics/general/w1000/

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

public class MyUniversalResHandler extends UniversalResHandler {

   @Override
   public void projectResourceScript() {

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

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

      if (selectedHandset.startsWith("IOS")) {
```

```
        // Logic for IOS devices only

             // Processes the "iOS/RetinaIPhone4" folder to
             // make all images in the "RetinaIPhone4" folder
             // available at the run time.
             addResourceFolder("iOS", "RetinaIPhone4");

             // Processes the "iOS/RetinaIPhone5" folder to
             // make all images in the "RetinaIPhone5" folder
             // available at the run time.
             addResourceFolder("iOS", "RetinaIPhone5");
       }

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

3. Update the mobile project's properties so that the mobile project uses your custom resource handler.

   a. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 17.

   b. Select the top-level child node of the project, which is the root application node.

   c. Set the **Res Handler** property to the name of your custom resource handler, *package_name .custom_resource_handler_name* .java, for example com.softwareag.mobile.myproject.MyUniversalResHandler.java.

4. In the *application_name* AppControllerImpl.java, override the methods the application uses to obtain image files.

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

If you add a subfolder to this resources/graphics folder...	Override this method
Android	getAndroidGraphicsFolder
iOS	getIOSGraphicsFolder
WinPhone	getWinPhoneGraphicFolder and getWinPhoneThemeFolder
general	getGeneralGraphicsFolder

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

   - resources/graphics/iOS/RetinaIPhone4/

- resources/graphics/iOS/RetinaIPhone5/

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

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

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

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

Coding a Custom Resource Handler

To use a custom resource handler for your mobile application

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

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

2. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 17.
3. Select the top-level child node of the project, which is the root application node.
4. Set the **Res Handler** property to identify the fully-qualified name of the custom resource handler you coded.
5. Save the resources for your mobile application, for example, audio files and icons. For instructions, see "Storing Resource Files for the Mobile Project" on page 37.

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 30.

Adding Devices to the Mobile Project

When you create a project, Mobile Development adds the following universal devices (targets) to your project:

- AND_generic_Android23xAPI.xml
- AND_generic_Android3xAPI.xml
- AND_generic_Android4xAPI.xml
- IOS_Apple_Universal.xml
- IOS_Apple_UniversalRetina.xml
- WN8_generic_Windows8TabletARM.xml
- WN8_generic_Windows8TabletX86.xml
- WN8_generic_WindowsPhone8xAPI.xml

If needed, you can add additional devices to the mobile project later using the Mobile DesignerAdd-Handset Ant target as described below.

Note: Targets for devices that are no longer supported (such as the BlackBerry targets) have been removed from Mobile Development. If your existing projects still include such targets, you have to remove them by yourself. If you need to remove targets for unsupported devices from your project, see "Removing Devices from the Mobile Project" on page 44.

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

1. Open the mobile project in the Outline Editor if it is not already open. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 17.
2. In the Project Explorer, expand the project, and drag the build.xml file to the Ant view.
3. In the Ant view, double-click the **Add-Handset** Ant target and fill in the required information.

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

Removing Devices from the Mobile Project

When you create a project, Mobile Development adds several universal devices to your project. You can add additional devices using the procedure described in "Adding Devices to the Mobile Project" on page 43.

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

To remove a device from a mobile project

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

   The targets folder contains one .xml file for each device the application supports.

2. Delete the .xml file that corresponds to the device you want to remove from the mobile project.

   Important: Do not remove the _defaults_.xml file.

Compiling Resources for a Device

Use the +Run-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.

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 30.

Managing Languages the Application Supports

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

Note: If you want to change the default language, see "Setting the Default Language for the Project" on page 46.

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:

For this property...	Specify...
Directionality	Direction to use for the language. Select one of the following: ■ **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 30.

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 35. For information about working with languages that are not the default, see "Specifying Values for Non-Default Language Text Resources" on page 47.

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 30.

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 35.

To specify values for non-default language text resources

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

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

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

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

6. Save the file.

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

Adding Services to a Mobile Project

Mobile Development supports different types of services as described in the topics below.

Adding RESTful Services to a Mobile Project

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 68.

To add RESTful services to a mobile project

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 130.
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 131
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.
8. When you add a **Method** child node, select the node, and in the **Properties** section of the Outline Editor set the properties for the **Method** node. For more information, see "Method Properties" on page 131.

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

9. Select the **Request** node, and in the **Properties** section of the Outline Editor set the properties for the **Request** node. For more information, see "Request Properties" on page 132.
10. If the RESTful service requires input parameters, perform the following steps for each input parameter:
    a. Right-click the **Request** node and select **New Child > Parameter**.
    b. Select the **Parameter** node, and in the **Properties** section of the Outline Editor set the properties for the **Parameter** node. For more information, see "Parameter Properties" on page 132.
11. Select the **Response** node, and in the **Properties** section of the Outline Editor set the properties for the **Response** node. For more information, see "Response Properties" on page 133.

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 30. 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 Task Client Services to a Mobile Project

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 Services to a Mobile Project" on page 48. For detailed information on the available services, see the *webMethods Task Engine API and Service Reference*.

To add Task Client services to a mobile project

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 Task Engine**.
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 to a Mobile Project

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 to a mobile project

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 129.

Adding Web Services to a Mobile Project

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 to a mobile project

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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**.

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.

Generating and Building a Mobile Project

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

Button	Description
	version number, select the targets, and select to retain the output build files. **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.

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.

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.

4 Building the User Interface for a Mobile Application

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 57.

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 61. For information about the types of views you can add, see "Objects to Use for Views" on page 86.

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 63.

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 75.

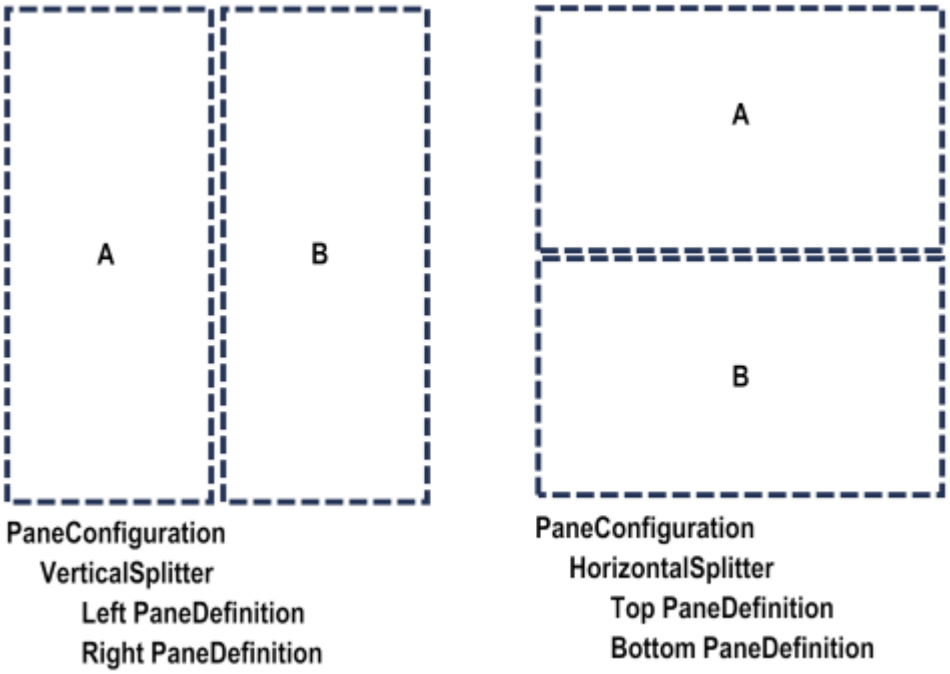
Defining Panes for the Application Window

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

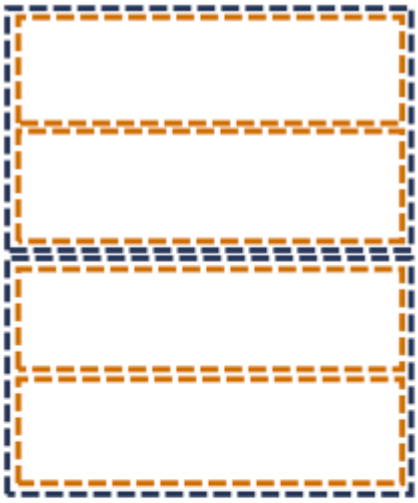
You define panes using the **PaneDefinition** object. You define a pane configuration using the **PaneConfiguration** object.

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

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

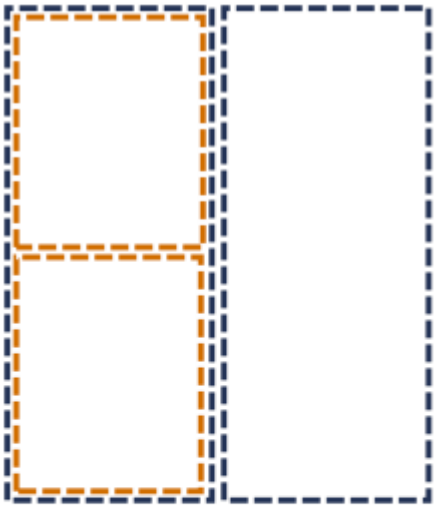


If you need a more complex arrangement of panes for an application's user interface, you can nest **VerticalSplitter** and **HorizontalSplitter** objects under parent splitter objects. For example, the following shows a layout with four panes arranged horizontally.



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
VeriticalSplitter
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 84.

> **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 85.

Adding Views to the Application's User Interface

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

To add a view to the user interface

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 86.
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 87
   - "NavView Properties" on page 88
   - "View Properties" on page 89
   - "WebView Properties" on page 90
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 121 and "Transition Properties" on page 125.

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 143.

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 30.

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 30.

   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 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:

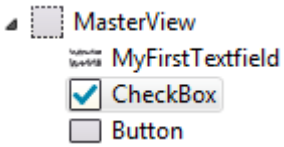
- "Objects to Use for the Layout of the User Interface" on page 91
- "Objects to Use for Tables" on page 95
- "Objects to Use for User Interface Controls" on page 99
- "Objects to Use for Content Providers" on page 115
- "Objects to Use for Event Listeners" on page 119
- "Objects to Use for Event Actions" on page 121

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, NavView, View, or WebView. For example, the only valid object that you can add to a NavView is a **NavButton** object. When using the Outline Editor to build a view's user interface, the Outline Editor only lists objects that are valid for each type of view.

The following sections provide general information about adding content to views. For information specifically about adding content to a ListView, see "Programmatically Populating a ListView" on page 65 and "Using a Content Provider to Populate a ListView" on page 68.

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:

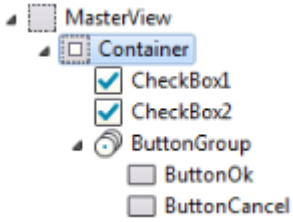


The following shows what the user interface might look like when the application executes:

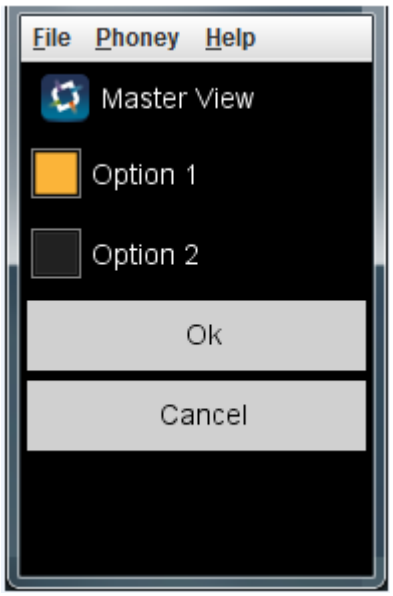
Nesting Objects in a View

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 68.

Objects to Add to the Project Model

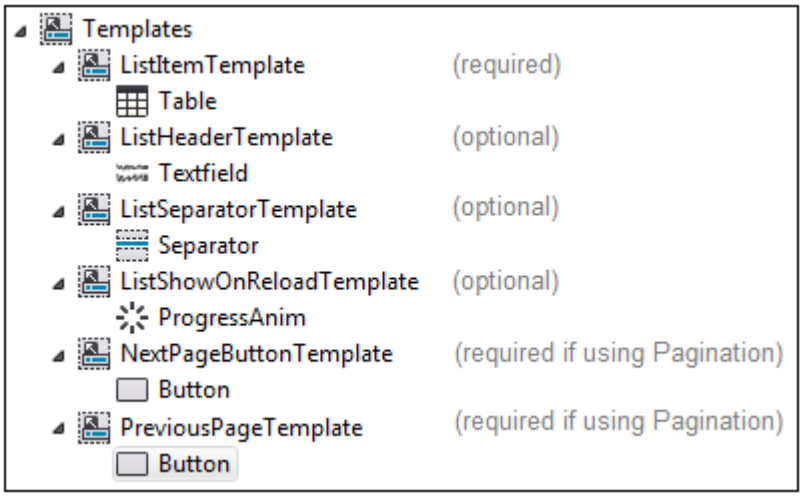
This section describes the objects you add to the model if you want to add logic to the view's controller to populate the **ListView** object.

- 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	Required. You must create a template that defines how to display a single item from the data source. Typically, you create a template for a **Table** or **TableButton** object. 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.
List header	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**. If you want to provide a header for the **ListView**, specify the template in the **ListView** object's **List View Header** property.
List separator	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.

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:

- getNumber OfRows(). 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** and **RESTDataSource**. For more information about data sources, see "Objects to Use for Content Providers" on page 115.

Note: Instead of using a **ContentProvider**, you can programmatically populate the **ListView**. For more information, see "Programmatically Populating a ListView" on page 65.

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** or a **RESTDataSource**.
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.

Property	Description
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**.

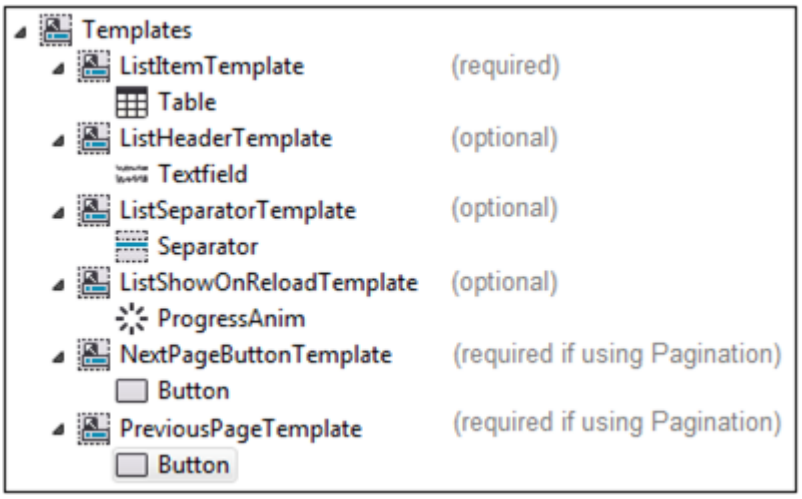
4. Add one or more **TemplateDataBinding** objects to the **ContentProvider** object. A **TemplateDataBinding** maps a control contained in the **RowTemplate** of the **ContentProvider** with a given set of data identified by the **Expression** property. Set the following properties:

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	Required. You must create a template that defines how to display a single item from the data source. Typically, you create a template for a **Table** or **TableButton** object. You reference this template in the **ContentProvider** object's **Template** property.
List header	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**. If you want to provide a header for the **ListView**, specify the template in the **ListView** object's **List View Header** property.
List separator	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. 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.

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	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.

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 48.

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`.

About User-Initiated Events and Listeners

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

You can add listeners to your model so that when a user-initiated event occurs for a control, your application can respond by taking an appropriate action. For more information about how to add listeners, see "Adding Listeners for User-Initiated Events" on page 73.

Types of Listeners

Mobile Development supports the following types of listeners:

- **GainFocusListener** that listens for when a user selects an object so that the user interface object gains focus.
- **LoseFocusListener** that listens for when a user interface object loses focus because the user stops selecting an object when the user selects another user interface control.
- **PostEditListener** that listens for when a user edits an object, for example an entry field, and generates an event after the object is edited.
- **PreEditListener** that listens for when a user edits an object, for example an entry field, and generates an event when the user first selects the object for editing.
- **TriggerListener** that listens for when a user uses an object, for example, presses a button.
- **RowSelectionListener** that listens to trigger events for each row in a **ListView** object, for example, when a user selects one row. This listener can be only added as a child of a **ContentProvider** object.
- **SwipeListener** that 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**, **NavView** 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 the following actions:

- **Back** action to transition to the previous view.
- **ChangePaneConfiguration** action to change the configuration of panes in the application's window.
- **Delegate** action to execute a method that you code.
- **OpenDialog** action to open an alert dialog that you have defined in your model.
- **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.

For more information about these objects, see "Objects to Use for Event Actions" on page 121.

Adding Listeners for User-Initiated Events

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

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

To add event listeners and associated actions

1. Ensure the view to which you want to add event listeners is displayed in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 following event listener objects:
   - **GainFocusListener**
   - **LoseFocusListener**
   - **PostEditListener**

- **PreEditListener**
- **TriggerListener**
- **RowSelectionListener**
- **SwipeListener**

For more information about these objects, see "About User-Initiated Events and Listeners" on page 72 and "Objects to Use for Event Listeners" on page 119.

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

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

   For more information about these objects, see "About User-Initiated Events and Listeners" on page 72 and "Objects to Use for Event Actions" on page 121.
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 121.

Note: Based on the event action you are using, you might also need to add application logic for the action. For more information, see "Logic to Respond to a Listener Event" on page 142.

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 30.

Defining Dialogs

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

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

To define a dialog for a mobile application

1. Ensure the mobile project is open in the Outline Editor. For instructions, see "Displaying a Mobile Project in the Outline Editor" on page 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 94.

   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 95.
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 121, "Adding Listeners for User-Initiated Events" on page 73, and "OpenDialog Properties" on page 124.
   - For information about how to add code to open a dialog, see "Logic to Display and Close a Dialog" on page 140.
   - For information about the code you can add to a dialog, see "Logic for a Dialog" on page 139.

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 30.

Using Templates to Define Custom Objects for a Mobile Project

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

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

For more information about creating a template, see "Creating a Template for a Custom Object" on page 76.

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 78.

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 127.
8. Save the mobile project and generate sources for the mobile project. For more information, see "Generating Sources for a Mobile Project" on page 30.

   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 65 and "Using a Content Provider to Populate a ListView" on page 68.

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	Template that you want to use. The list includes the templates that you have added to your project.

Tip: To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see "Generating Sources for a Mobile Project" on page 30.

5 User Interface Object Reference

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 80
Windows	"Objects to Use for Windows" on page 82
Panes	"Objects to Use for Panes" on page 83
Views	"Objects to Use for Views" on page 86
Layout	"Objects to Use for the Layout of the User Interface" on page 91
Dialogs	"Objects to Use for Dialogs" on page 93
Tables	"Objects to Use for Tables" on page 95
Controls	"Objects to Use for User Interface Controls" on page 99
Content Providers	"Objects to Use for Content Providers" on page 115
Event Listeners	"Objects to Use for Event Listeners" on page 119
Event Actions	"Objects to Use for Event Actions" on page 121
Templates	"Objects to Use for Templates" on page 125

Application Node Properties

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

Property	Description
Bundle Id	Package name for your mobile project. You initially define the package name for a mobile project when you create the project using the New Mobile Development Project wizard. You can use this property to change the package name. For more information, see "Changing the Package Name" on page 153.
Context Persistence Mode	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.
Default Language	Default language for the application. For more information, see "Setting the Default Language for the Project" on page 46.
Name	Name of the mobile application. This is an internal application name that Mobile Development uses. 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 152.
Orientation	Whether you want the application to display in portrait mode, landscape mode, or rotate as a user turns the device. 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 45.
Res Handler	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. If you want to use a custom resource handler, use this property to specify its fully-qualified name. For more information, see "Defining Resources for the Mobile Project" on page 36.

Property	Description
Use Camera	Whether you want to set the Mobile Designer property project.handset.uses.camera to `true`. See *Using webMethods Mobile Designer* for more information on this property.
Use Location	Whether you want to set the Mobile Designer property project.handset.uses.Location to `true`. See *Using webMethods Mobile Designer* 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 project.handset.uses.FCPIM to `true` in order to list all contacts on your device. See *Using webMethods Mobile Designer* for more information on this property.
Use Push Notifications	Whether you want to set the Mobile Designer property project.handset.push.notifications to `true`. See *Using webMethods Mobile Designer* for more information on this property.
Use Sensors	Whether you want to set the Mobile Designer property project.handset.uses.Sensors to `true`. See *Using webMethods Mobile Designer* for more information on this property.
Use WMA	WMA stands for Wireless Messaging API. Whether you want to set the Mobile Designer property project.handset.uses.WMA to `true`. See *Using webMethods Mobile Designer* 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 83.

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.

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 57.

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 84.
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 application's **Window** object when using this pane configuration. For information about setting properties for the **PaneConfiguration** object, see "PaneConfiguration Properties" on page 85.

Object	Description
PaneDefinition	Specifies the following for a single pane in a pane configuration: ■ 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. For information about setting properties for the **PaneDefinition** object, see "PaneDefinition Properties" on page 85.
VerticalSplitter	Indicates that you want to display two panes vertically, side by side. For information about setting properties for the **VerticalSplitter** object, see "VerticalSplitter Properties" on page 85.

HorizontalSplitter Properties

Property	Description

Height	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.
	■ To set the absolute size to use for the top pane, type the value. For example:
	  ■ 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:
	  ■ To use 320 pixels for the bottom pane, specify: `, 320`
	  ■ To use 38 percent for the bottom pane, specify: `, 38%`
	If you do not specify a value, the split creates two equal sections.
	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

Property	Description
	required for the NavView. The top pane uses the remaining space.

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	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.
	■ To set the absolute size to use for the left pane, type the value. For example:
	  ■ 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:
	  ■ To use 320 pixels for the right pane, specify: `, 320`

Property	Description
	■ To use 38 percent for the right pane, specify: `, 38%` If you do not specify a value, the split creates two equal sections.

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	Defines a view that displays a list of data obtained from a specified data source. Add a **ContentProvider** child object to the **ListView** object to define the content you want to list in the view. For information about setting properties for a **ListView**, see "ListView Properties" on page 87.
NavView	Defines a view that you want to use for navigation in your application. 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. For information about setting properties for a **NavView** object, see "NavView Properties" on page 88.
View	Defines a general purpose view for your application. For information about setting properties for a **View** object, see "View Properties" on page 89.
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 90.

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. 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.

Property	Description
Inner YSpacing	Vertical distance between each element in the view.
List View Header	Template that defines an object to display as the header for the list of data displayed in the ListView. Specify a template that you previously defined. The template should customize an object like a **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 ListView. Specify a template that you previous defined. The template should customize an object like a **Separator** object.
Show On Reload	Template that defines an object to display while the application is obtaining data to refresh the list of items in the view. Specify a template that you previous defined. The template should customize an object like a **ProgressAnim**, **Image** or **Textfield** object.
VScrollable	Whether you want to allow vertical scrolling in the view.

NavView Properties

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

View Properties

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

Property	Description
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.
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.

Property	Description
HScrollable	Whether you want to allow horizontal scrolling in the view.
Inner Height	Usable height of the view in which you can insert child objects.
Inner Width	Usable width of the view in which you can insert child objects.
Inner X	Distance from the view's left edge to where child elements are drawn.
Inner Y	Distance from the view's top edge to where child elements are drawn.
Name	Name you assign the view. This name does not appear in the application's user interface.
Popup Dismiss Text	For the iOS platform, the text to use on a label that closes an open keyboard or drop-down list.
Url	URL to the web page to load into the view.
VScrollable	Whether you want to allow vertical scrolling in the view.

Objects to Use for the Layout of the User Interface

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

Object	Description
Group	Creates a container that holds a group of user interface objects. To specify user interface objects to include in the group, add the objects as children of the **Group** object. Use the **Group** object's **Visible** property to indicate whether you want the group of user interface objects visible or hidden.

Object	Description
	For information about setting properties for the **Group** object, see "Group Properties" on page 92.
RadioButtonGroup	Creates a container that holds a group of radio buttons. To specify the radio buttons to include in the group, add **RadioButton** objects as children of the **RadioButtonGroup** object. The **RadioButtonGroup** object does not have any properties.
Separator	Displays a horizontal line that you can use to separate blocks of content. For information about setting properties for the **Separator** object, see "Separator Properties" on page 92.
Spacer	Displays blank space that you can use to create extra padding between user interface objects. For information about setting properties for the **Spacer** object, see "Spacer Properties" on page 93.

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.

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

Spacer Properties

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.
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	Displays a small pop-up that you can use 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. You must add at least one **AlertDialogButton** child object for the **AlertDialog** object. For information about setting properties for the **AlertDialog** object, see "AlertDialog Properties" on page 94.
AlertDialogButton	Displays a button to include in an **AlertDialog**. For information about setting properties for the **AlertDialogButton** object, see "AlertDialogButton Properties" on page 95.

AlertDialog Properties

Property	Description
Class Name	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. The generated class extends the NativeUI class for the element, which is com.softwareag.mobile.runtime.nui.nUIAlertDialog. You can specify the same value for **Class Name** for **AlertDialog** objects. Mobile Development generates only one class. For more information, see "Logic for a Dialog" on page 139.
Header Text	Text you want displayed in the header area of the dialog. Leave this property blank if you do not want text in the header.
Text	Text to display in the dialog. You must specify this property.

AlertDialogButton Properties

Property	Description
Id	Identifier you assign to the button. You must specify this property.
Text	Text you want displayed on the button.

Objects to Use for Tables

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

Object	Description
DynamicTablecell	Specifies a method that executes at run time to populate a table cell. A **DynamicTablecell** object is the child of a **TableRow** object. For information about setting properties for the **DynamicTablecell** object, see "DynamicTablecell Properties" on page 96.
DynamicTablerow	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. For information about setting properties for the **DynamicTablerow** object, see "DynamicTablerow Properties" on page 96.
Table	Displays a table. To specify the rows in the table, add one or more **TableRow** objects or a single **DynamicTablerow** object as children of the **Table** object. For information about setting properties for the **Table** object, see "Table Properties" on page 97.
TableButton	Displays a table that contains other objects and that acts as a button.

Object	Description
	For information about setting properties for the **TableButton** object, see "TableButton Properties" on page 98.
TableCell	Adds a cell to a table row. A **TableCell** object is the child of a **TableRow** object. To specify user interface objects that you want to display in the table cell, add the objects as children of the **TableCell** object. For information about setting properties for the **TableCell** object, see "Tablecell Properties" on page 98.
TableRow	Adds a single row to a table. A **TableRow** object is the child of a **Table** object. A child of a **Table** object. Use to add a single row to the table. 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. For information about setting properties for the **TableRow** object, see "Tablerow Properties" on page 99.

DynamicTablecell Properties

Property	Description
Method Name	Name of a method you code to populate the table cell. For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see "Logic for a Method Name Property" on page 140.

DynamicTablerow Properties

Property	Description
Method Name	Name of a method you code to populate the table. For information about the Java sources that Mobile Development generates for the method and how to

Property	Description
	provide logic for the method, see "Logic for a Method Name Property" on page 140.

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.

Property	Description
Spacing Height	Distance between the table rows.
Spacing Width	Distance between the table columns.
Visible	Whether the table is visible or hidden.
Width	Width of the table.

TableButton Properties

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.
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 103.
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
	For information about setting properties for the **Button Group** object, see "ButtonGroup Properties" on page 104.
Checkbox	Displays a check box. For information about setting properties for the **Checkbox** object, see "CheckBox Properties" on page 104.
Container	Creates a container that holds other user interface objects. To specify user interface objects that you want to display in the container, add the objects as children of the **Container** object. 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. For information about setting properties for the **Container** object, see "Container Properties" on page 105.
DateEntry	Displays a date or time selector control. For information about setting properties for the **DateEntry** object, see "DateEntry Properties" on page 106.
DropDownListEntry	Displays a drop-down list that contains selection items. 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. For information about setting properties for the **DropDownListEntry** object, see "DropDownListEntry Properties" on page 106.
DynamicDisplayObject	Name of a method you code to display a user interface object. For information about setting

Object	Description
	properties for the **DynamicDisplayObject** object, see "DynamicDisplayObject Properties" on page 107.
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 107. For information about setting properties for the **DynamicDisplayObjectArray** object, see "DynamicDisplayObjectArray Properties" on page 107.
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 108.
Entry	Displays a text entry box. You can set the **Entry** object's **Input Type** property to: ■ 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 108.
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 109.

Object	Description
NavButton	Displays a button that an application uses for navigation. For information about setting properties for the **NavButton** object, see "NavButton Properties" on page 110.
Pagination	Adds objects that a user selects to display the next or previous page of list items in a ListView. For information about setting properties for the **Pagination** object, see "Pagination Properties" on page 111.
ProgressAnim	Displays an animated status indicator that indicates background activity is in progress. For information about setting properties for the **ProgressAnim** object, see "ProgressAnim Properties" on page 111.
RadioButton	Displays a single radio button that uses two states, selected or cleared. For information about setting properties for the **RadioButton** object, see "RadioButton Properties" on page 112.
SearchEntry	Displays a search entry field. For information about setting properties for the **SearchEntry** object, see "SearchEntry Properties" on page 113.
StringDropDownListEntryItem	Adds a single entry to a drop-down list. A **StringDropDownListEntryItem** object is the child of a **DropDownListEntry** object. For information about setting properties for the **StringDropDownListEntryItem** object, see "StringDropdownlistEntry Properties" on page 113.
Textfield	Displays plain text in a label or for a block of text.

Object	Description
	For information about setting properties for the **Textfield** object, see "Textfield Properties" on page 114.
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 115.

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.
Text	Text to display on the button.
Visible	Whether the button is visible or hidden.
Width	Width of the button.

ButtonGroup Properties

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

CheckBox Properties

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

Container Properties

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.
Visible	Whether the container is visible or hidden.
VScrollable	Whether you want to allow vertical scrolling in the container.
Width	Width of the container.

DateEntry Properties

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

DropDownListEntry Properties

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

Property	Description
Position Y	Distance from the drop-down list's top edge to its parent object's inner Y position.
Visible	Whether the drop-down list is visible or hidden.
Width	Width of the drop-down list.

DynamicDisplayObject Properties

Property	Description
Create On Condition	Whether to create the user interface object at run time.
Method Name	Name of a method you code to display a user interface object For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see "Logic for a Method Name Property" on page 140.
Name	Name you assign the user interface object in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Visible	Whether the user interface object is visible or hidden.

DynamicDisplayObjectArray Properties

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 140.

Property	Description
Name	Name you assign the array of user interface objects in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Visible	Whether the array of user interface objects is visible or hidden.

DynamicDropdownlistEntryItems Properties

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 140.

Entry Properties

Property	Description
Context Key	Name of a context key you want to reference using syntax such as `${CONTEXT_key}`. 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.

Property	Description
Input Type	Type of input a user can supply in the text entry box. For example, you might specify `text`, `textPassword`, or `number`.
Lines	Number of lines to display in the text entry box. This is the number of lines into which a user can type information.
Name	Name you assign the text entry box in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the text entry box's left edge to its parent object's inner X position.
Position Y	Distance from the text entry box's top edge to its parent object's inner Y position.
Text	Text to display in the text entry box.
Visible	Whether the text entry box is visible or hidden.
Width	Width of the text entry box.

Image Properties

Property	Description
Create On Condition	Whether to display the image at run time.
HAlign	Horizontal alignment of the image.
Image	Image to display. **Note:** The image must be one of your application resources. For more information about resources, see "Defining Resources for the Mobile Project" on page 36.
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.

Property	Description
Position X	Distance from the image's left edge to its parent object's inner X position.
Position Y	Distance from the image's top edge to its parent object's inner Y position.
Scale to Parent Height	Whether you want the image to be scaled vertically to match the height of its parent object. Select the **Select to Parent Height** check box if you want the image scaled to the parent object's height.
Scale to Parent Width	Whether you want the image to be scaled horizontally to match the width of its parent object. Select the **Select to Parent Width** check box if you want the image scaled to the parent object's width.
Visible	Whether the image is visible or hidden.

NavButton Properties

Property	Description
Create On Condition	Whether to create the button at run time.
Icon	Icon image to display on the button. **Note:** The icon must be one of your application resources. For more information about resources, see "Defining Resources for the Mobile Project" on page 36.
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: ■ BACK This type is placed on the left of the view header. ■ DEFAULT

Property	Description
	This type is placed on the right of the view header.
Visible	Whether the button is visible or hidden.

Pagination Properties

Property	Description
Max Number Per Page	Maximum number of list items to display on a single page of a ListView.
Next Page Template	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. Specify a template that you previously defined. The template should customize an object like a **Button** object.
Previous Page Template	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. Specify a template that you previously defined. The template should customize an object like a **Button** object.

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.

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.
Width	Width of the radio button.

SearchEntry Properties

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

StringDropdownlistEntry Properties

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

Textfield Properties

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

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

Objects to Use for Content Providers

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

Object	Description
TemplateDataBinding	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** object. For information about setting properties for the **TemplateDataBinding** object, see "TemplateDataBinding Properties" on page 117.
DataBinding	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** or **Image** object. For information about setting properties for the **DataBinding** object, see "DataBinding Properties" on page 117.
DynamicDataSource	Holds a list of elements to be bound to the user interface elements. You need to implement IListDatasource. The **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. **Note:** Using a **DynamicDatasource** object as a child of an **ContentProvider** object is deprecated. For information about setting properties for the **DynamicDataSource** object, see "DynamicDataSource Properties" on page 118.
ContentProvider	Specifies the content you want listed in a **ListView** object. For information about setting properties for the **ContentProvider** object, see "ContentProvider Properties" on page 118.
RESTDataSource	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. **Note:** Using a **RESTDatasource** object as a child of a **ContentProvider** object is deprecated.

Object	Description
	For information about setting properties for the **RESTDataSource** object, see "RESTDataSource Properties" on page 119. See also "Objects to Use for RESTful Services" on page 130.
DataTransformer	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. For information about setting properties for the **DataTransformer** object, see "DataTransformer Properties" on page 119.

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** or **RESTDataSource** object to use to obtain the data to bind to the control.
Expression	Expression that identifies the data to bind to the control specified by the **Control** property.

DataBinding Properties

Property	Description
Data Source	Name of the **DynamicDataSource** or **RESTDataSource** 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	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 140. The method you code must return an instance of IListViewDatasource, which is in the gen/api-src folder in the com.softwareag.mobile.runtime.toolkit.delegates package.
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.
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	Name of a **Template** object that you previously defined. 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. For more information about templates, see "Using Templates to Define Custom Objects for a Mobile Project" on page 76 and "Using a Content Provider to Populate a ListView" on page 68.

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 com.softwareag.mobile.runtime.toolkit.rest.JPath.
Filter	A JSON expression value to filter the underlying JSON data. For more information about JSON expressions, see com.softwareag.mobile.runtime.toolkit.rest.JPath.
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	Method to invoke to provide the data for the **ListView** object. 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.

DataTransformer Properties

Property	Description
Class Name	The class name specifies the name of the corresponding Java class which will be generated to src/*bundleId* /data/ transformer/*classname* .java. Implementations need to implement IDataTransformer.

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 "About User-Initiated Events and Listeners" on page 72.

Object	Description
GainFocusListener	Generates a user-initiated event when a user interface object gains focus. 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. The **GainFocusListener** object does not have any properties.
LoseFocusListener	Generates a user-initiated event when a user interface object loses focus. 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. The **LoseFocusListener** object does not have any properties.
PostEditListener	Generates a user-initiated event after a user edits an object. **Note:** In the case of keyboard entry, a new event is generated for each character typed or deleted. 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. The **PostEditListener** object does not have any properties.
PreEditListener	Generates a user-initiated event when a user is about to edit an object. 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. The **PreEditListener** object does not have any properties.
TriggerListener	Generates a user-initiated event after a user uses an object. 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.

Object	Description
	Note: If you add a **TriggerListener** as the child of an **Image** object, the image acts as a button. The **TriggerListener** object does not have any properties.
SwipeListener	Generates a user-initiated event when a user swipes to the left or right. A **SwipeListener** is the child of a **View** object. For information about setting properties for the **SwipeListener** object, see "SwipeListener Properties" on page 121.
RowSelectionListener	Generates a user-initiated event when a user selects a row. A **RowSelectionListener** is the child of a **ContentProvider** object. The **RowSelectionListener** object does not have any properties.

SwipeListener Properties

Property	Description
Direction	Either a **LeftToRight** or **RightToLeft** swipe event.

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 "About User-Initiated Events and Listeners" on page 72.

Object	Description
Back	The application displays the previous view when the associated user-initiated event occurs.

Object	Description
	For information about setting properties for the **Back** object, see "Back Properties" on page 123. **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 137.
ChangePaneConfiguration	The application switches to a pane configuration you specify when the associated user-initiated event occurs. For information about setting properties for the **ChangePaneConfiguration** object, see "ChangePaneConfiguration Properties" on page 123.
Delegate	The application executes a method you specify when the associated user-initiated event occurs. For information about setting properties for the **Delegate** object, see "Delegate Properties" on page 124.
OpenDialog	The application displays the **AlertDialog** you specify when the associated user-initiated event occurs. For information about setting properties for the **OpenDialog** object, see "OpenDialog Properties" on page 124.
ReloadContentProvider	The application reloads the data source. The **ReloadContentProvider** object does not have any properties.
ToggleVisibility	The application switches between making a user interface object that you specify either visible or hidden. ■ If the object is currently visible, the application hides it. ■ If the object is currently hidden, the application makes it visible. An example of using the **ToggleVisibility** action might be to show a button if a user enters text in a text entry field. For information about setting properties for the **ToggleVisibility** object, see "ToggleVisibility Properties" on page 124.

Object	Description
Transition	The application transitions to the new view you specify when the associated user-initiated event occurs. For information about setting properties for the **Transition** object, see "Transition Properties" on page 125.

Back Properties

Property	Description
Back To Root	Back behavior that you want the application to take when the user-initiated event occurs. ■ 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. When you select the check box, the TransitionStackController opens the first pushed view. For more information about the TransitionStackController, see "About the TransitionStackController" on page 137.

ChangePaneConfiguration Properties

Property	Description
Method Name	Name of a method that performs the change to the new pane configuration. Specifying the **Method Name** property is optional. 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 140.
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

Property	Description
	a **PaneConfiguration** object you previously defined in the model.

Delegate Properties

Property	Description
Method Name	Name of a method you code and that the application executes when the user-initiated event occurs. 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 140.

OpenDialog Properties

Property	Description
Dialog	Dialog to display when the user-initiated event occurs. Specify the name of a **AlertDialog** object you previously defined in the model.

ToggleVisibility Properties

Property	Description
Control	The user interface object that you want to make visible or hide in response to the user-initiated event. **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	Name of a method that performs the transition to the new view. Specifying the **Method Name** property is optional. 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 140.
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**, **NavView**, **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 76.

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 126.
Template	Defines a custom user interface object. For information about setting properties for the **Template** object, see "Template Properties" on page 127.
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 127.

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

Property	Description
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 76.

TemplateReference Properties

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

6 Services Object Reference

Objects to Use for 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 130.
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 131.
Method	Identifies the method to execute. For information about setting properties for the **Method** object, see "Method Properties" on page 131.
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 132.
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 132.
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 133.

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, https://{company}.apps.com/networking, where {company} is a dynamic URI element.

Resource Properties

Property	Description
Path	Path to the service you want to use. You can include dynamic URI elements, for example, record/{objectName}, where {objectName} is a dynamic URI element. You can nest **Resource** objects under a parent **Resource** object to provide subpath values. 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.

Method Properties

Property	Description
Class Name	Name of a Java class that Mobile Development generates in the gen/src folder in the *package_name* .services.rest package, where *package_name* is the package name that you specified for your mobile project. 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. The Java class includes an execute method that the mobile application invokes to execute the REST service.
Name	How the request is sent over the network. Select one of the following: ■ **DELETE** to delete the resource identified by the request URI on the server.

Property	Description
	■ **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	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: ■ **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	Default value for the parameter. If you specify a value, Mobile Development does not create a method parameter for the **Parameter** object. If you specify a value for **Default Value**, the **Repeating** property must be cleared.
Name	Name of the parameter. The value you specify must be unique among all parameters you specify for a specific **Request** object.

Property	Description
	Note: Do not use `postData` as a value. The value `postData` is reserved.
Repeating	Whether the parameter contains multiple values. Select the **Repeating** check box if the parameter contains multiple values. If you select the **Repeating** check box, you cannot specify a value for the **Default Value** property.
Style	Whether the parameter is to be added to the header or query string. ■ **HEADER** if the parameter is to be added to the header. ■ **QUERY** if the parameter is to be added to the URL.

Response Properties

Property	Description
Accept	Format of the response from the REST service. Specify one of the following: ■ **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	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 in the **Update Interval** property. A **SyncComponent** object can be used as a REST method for a REST data source.

Object	Description
	For information about setting properties for the **SyncComponent** object, see "SyncComponent Properties" on page 134.

SyncComponent Properties

Property	Description
App Name	Name of the mobile application. For more information, see *Developing Data Synchronization Solutions with webMethods Mobile Support*.
App Version	Version of the mobile application. For more information, see *Developing Data Synchronization Solutions with webMethods Mobile Support*.
Hostname	Host name of the Enterprise Gateway Server.
Msc Alias	Alias used to identify the mobile sync component. For more information, see *Developing Data Synchronization Solutions with webMethods Mobile Support*.
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.

7 Creating Application Logic

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 138.
Dialog	Business logic for a dialog. For more information, see "Logic to Display and Close a Dialog" on page 140.
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 140.

Type of Logic	Description
Property value	Logic to programmatically set a property. 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. For information, see "Logic to Programmatically Set a Property Value at Run Time" on page 141.
Event handling	Logic to respond to a user-initiated event. For more information, see "Logic to Respond to a Listener Event" on page 142.
View transitions	Logic to transition to another view. For more information, see "Logic to Transition to Another View" on page 143
Templates	Logic for templates to define custom user interface objects. For more information, see "Creating a Template for a Custom Object" on page 76.

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 31.

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 145.

Logic for a Dialog

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

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

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

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

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

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

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

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

Logic to Display and Close a Dialog

You can open a dialog in the following ways:

- Open a dialog in response to a listener event.

  If you want to open a dialog in response to a listener event, add an **OpenDialog** object to the model and select the name of the dialog you want to open for the **OpenDialog** object's **Dialog** property. For more information, see "About User-Initiated Events and Listeners" on page 72, "Adding Listeners for User-Initiated Events" on page 73, and "OpenDialog Properties" on page 124.

- 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 Abstract*view_name* Controller.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 Abstract*view_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 140.

- **Fully-qualified name**

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

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

Logic to Respond to a Listener Event

When you add an event listener object to the model, for example, a **GainFocusListener** object, you are required to add a child event action object that indicates how the application responds when the user-initiated event occurs. For more information, see "About User-Initiated Events and Listeners" on page 72 and "Objects to Use for Event Actions" on page 121.

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

Event Action Object	Description
Back	Mobile Development generates the logic to perform this action.
ChangePaneConfiguration	Mobile Development generates logic for this action. However, you can customize the logic. If you specify a value for the **ChangePaneConfiguration** object's **Method Name** property, Mobile Development generates the method you specify and places the logic to change pane configurations in the method. If you want to customize the logic, the method into the user space and make your customizations. For more information about how to provide code for the method, see "Logic for a Method Name Property" on page 140.
Delegate	Mobile Development generates a method you specify in the user space. However, this method does not perform any useful task. You must add the logic you want performed when the user-initiated event occurs. For more information about how to provide code for the method, see "Logic for a Method Name Property" on page 140.
OpenDialog	Mobile Development generates the logic to perform this action.
ToggleVisibility	Mobile Development generates the logic to perform this action.
Transition	Mobile Development generates logic for this action. However, you can customize the logic.

Event Action Object	Description
	If you specify a value for the **Transition** object's **Method Name** property, Mobile Development generates the method you specify and places the logic to transition to the new view in the method. If you want to customize the logic, copy the method into the user space and make your customizations. For more information about how to provide code for the method, see "Logic for a Method Name Property" on page 140.

Logic to Transition to Another View

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

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

  When you use the **Transition** event action object, you do not need to add any code if you simply want to transition to another view. However, if you want to perform actions before or after the transition, you can add custom code. For more information, see "Logic to Respond to a Listener Event" on page 142.

- **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 137.

  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
MySecondViewControllerImpl());
```

The following shows a sample method you can you 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
onCreateWindow()	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.
onOrientationChange()	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.

Method	Description
	By default, this method updates the dimensions of the panes in the window. If you need to take further action, you can uncomment the onOrientationChange() method and add your custom logic.
onPushNotification(String message)	This method is called after a push notification has been received by the device. The passed string represents the message that has been sent with the push notification.
updatePaneDimensions()	This method is called by onOrientationChange 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.

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

Method	Description
	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 137.
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 create using Mobile Development use the provided TransitionStackController, 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. **Note:** If your application does not use the TransitionStackController, views do not automatically have a Back button. You manually add a Back button to the view using the addBackButton() method. For more information about the TransitionStackController, see "About the TransitionStackController" on page 137.
getBackButtonTitle()	This method is executed when adding the Back button to the view. By default, applications you create using Mobile Development use the provided TransitionStackController, 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".
nUIEventCallback()	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.

Method	Description

	You have to call the super method so that the default event implementations are not disabled:
	`public void nUIEventCallback() {`
	`  super.nUIEventEventCall();`
	`}`

initBindings()	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.

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
init()	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.
onTemplateCreated()	This method is executed by init() after all contents have been created. You can add custom business logic to this method.
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.

Method	Description
initBindings()	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 to a Mobile Project" on page 50 and "Services Object Reference" on page 129.

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 157.

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.

8 Managing a Project

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 152.

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 152.

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 30.

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.

9 Code Snippets

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(AbstractApplicationController.getInstance()).open();
  }
  public void onOperationFinished(final IOperation operation) {
    if (operation.getResultCode() != HttpConnection.HTTP_OK) {
      onOperationFailed(operation);
    }
  }
}
public class GetIDResultDelegate extends ErrorDelegate {
  public void onOperationFinished(final IOperation operation) {
    super.onOperationFinished(operation);
    if (operation.getResultCode() == HttpConnection.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() == HttpConnection.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 161 and " Downloading an Image from a Remote Host" on page 162.

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 AbtractRestOperation 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 158.

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 159.

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 158. 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 159). 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 = HttpConnection.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 `HTTPConnection.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 HTTPConnection, 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.HttpConnection;
import javax.microedition.lcdui.Image;
import com.softwareag.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 {
    HttpConnection hpc = null;
    DataInputStream dis = null;
    final ByteArrayOutputStream output = new ByteArrayOutputStream();
    try {
      hpc = (HttpConnection) 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	*project_name* /native/android
iOS	*project_name* /native/ios
Windows	*project_name* /native/winphone

   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.