Oracle Enterprise Pack for Eclipse

Developing Mobile Applications with Oracle Mobile Application Framework (OEPE Edition)

Release 2.3.0

E68520-01

June 2016

Documentation for Oracle Enterprise Pack for Eclipse developers that describes how to use Oracle Enterprise Pack for Eclipse to create mobile applications that run natively on devices.

ORACLE

Oracle Enterprise Pack for Eclipse Developing Mobile Applications with Oracle Mobile Application Framework (OEPE Edition) Release 2.3.0

E68520-01

Copyright © 2014, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Catherine Pickersgill

Contributing Author: Walter Egan, Sujatha Joseph

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface xxi

Audience xxi
Documentation Accessibility xxi
Related Documents xxi
Conventions xxi

What's New in This Guide for MAF Release 2.3.0 xxiii

New and Changed Features for MAF Release 2.3.0 xxiii
Other Significant Changes in this Document for MAF Release 2.3.0 xxv

1 Introduction to Oracle Mobile Application Framework

1.1 Introduction to the Mobile Application Framework 1-1
1.2 About the MAF Runtime Architecture 1-2
1.3 About Developing Applications with MAF 1-6
1.3.1 About Connected and Disconnected Applications 1-8
1.4 MAF Sample Applications 1-9

2 Getting Started with MAF Application Development

2.1 Introduction to the MAF Development Environment 2-1
2.1.1 About the MAF Application Editor 2-2
2.1.2 About the MAF Feature Editor 2-3
2.2 Using the Oracle MAF Perspective 2-3
2.3 Creating a MAF Application 2-4

2.3.1 How to Create a MAF Application ... 2-4
2.3.2 What Happens When You Create an MAF Application ... 2-4
2.4 Defining Application Features for a MAF Application ... 2-5
2.4.1 How to Define an Application Feature ... 2-6
2.5 Adding Content to an Application Feature ... 2-6
2.6 Adding Application Features to a MAF Application ... 2-7
2.6.1 How to Add an Application Feature to a MAF Application ... 2-7
2.6.2 What You May Need to Know About Feature Reference IDs and Feature IDs ... 2-7
2.7 Creating MAF AMX Pages and MAF Task Flows ... 2-8
2.7.1 How to Create an MAF AMX Page ... 2-10
2.7.2 How to Create MAF Task Flows ... 2-11
2.7.3 What Happens When You Create MAF AMX Pages and Task Flows ... 2-12
2.8 Containerizing a MAF Application for Enterprise Distribution ... 2-13

3 Configuring the Content of a MAF Application

3.1 Introduction to Configuring MAF Application Display Information ... 3-1
3.2 Setting Display Properties for a MAF Application ... 3-1
3.3 Changing the Launch Screen for Your MAF Application on iOS ... 3-4
3.4 Setting Display Properties for an Application Feature ... 3-5

4 Configuring MAF Application Features

4.1 Introduction to MAF Application Features ... 4-1
4.2 Configuring Application Navigation ... 4-2
4.2.1 How to Set the Display Behavior for the Navigation Bar ... 4-2
4.2.2 How to Set the Display Behavior for the Springboard ... 4-3
4.2.3 How to Set the Slideout Behavior for the Springboard ... 4-4
4.2.4 How to Set the Display Order for Application Features ... 4-4
4.3 What Happens When You Configure the Navigation Options ... 4-5
4.4 What Happens When You Set the Animation for the Springboard ... 4-6
4.5 What You May Need to Know About Custom Springboard Application Features with HTML Content 4-7
4.6 What You May Need to Know About Custom Springboard Application Features with MAF AMX Content 4-8
4.7 What You May Need to Know About the Runtime Springboard Behavior ... 4-10
4.8 Navigating a MAF Application Using Android's Back Button ... 4-10
4.8.1 How to Configure Behavior of the Android System Back Button ... 4-12
4.9 Creating a Sliding Window in Your MAF Application ... 4-13
4.10 Using Custom URL Schemes in MAF Applications ... 4-15

5 Defining the Content Type of MAF Application Features

5.1 Introduction to Content Types for an Application Feature ... 5-1
5.2 Defining the Application Feature Content as Remote URL or Local HTML ... 5-2
5.3 Defining the Application Feature Content as a MAF AMX Page or Task Flow ... 5-3
5.4 What You May Need to Know About Selecting External Resources ... 5-5

6 Localizing MAF Applications

6.1 Introduction to MAF Application Localization 6-1
6.2 Setting Resource Bundle Options for a MAF Application 6-2
6.3 Defining Text Resources in a Base Resource Bundle 6-3
6.3.1 How to Define a Text Resource in a Base Resource Bundle 6-5
6.3.2 What Happens When You Define a Text Resource in a Base Resource Bundle 6-6
6.4 Creating Locale-Specific Resource Bundles 6-8
6.4.1 How to Create a Locale-Specific Resource Bundle 6-9
6.5 Editing Resources in Resource Bundles 6-10
6.6 Localizing Image Files in a MAF Application 6-10
6.7 MAF Support of Languages 6-11
6.8 Localizable MAF Properties 6-12

7 Skinning MAF Applications

7.1 Introduction to MAF Application Skins 7-1
7.1.1 About the maf-config.xml File 7-3
7.1.2 About the maf-skins.xml File 7-4
7.2 Adding a Custom Skin to an Application 7-6
7.3 Specifying a Skin for an Application to Use 7-6
7.4 Registering a Custom Skin 7-7
7.5 Versioning MAF Skins 7-8
7.6 What Happens When You Version Skins 7-8
7.7 Overriding the Default Skin Styles 7-9
7.8 What You May Need to Know About Skinning 7-11
7.9 Adding a New Style Sheet to a Skin 7-12
7.10 Enabling End Users Change an Application's Skin at Runtime 7-13
7.11 What Happens at Runtime: How End Users Change an Application's Skin 7-15

8 Reusing the MAF Application Content

8.1 Introduction to Feature Archive Files 8-1
8.2 Using FAR Content in a MAF Application 8-1
8.3 What Happens When You Add a FAR 8-2
8.4 What You May Need to Know About Enabling the Reuse of Feature Archive Resources 8-3

9 Using Plugins in MAF Applications

9.1 Introduction to Using Plugins in MAF Applications 9-1
9.2 Enabling a Core Plugin in Your MAF Application 9-2
9.2.1 How to Enable a Core Plugin in Your MAF Application 9-2
9.2.2 What Happens When You Enable a Core Plugin in Your MAF Application 9-3
9.3 Registering Additional Plugins in Your MAF Application 9-3
9.3.1 How to Register an Additional Plugin 9-3
9.3.2 What Happens When You Register an External Plugin for Your MAF Application 9-4
9.4 Deploying Plugins with Your MAF Application 9-5
9.5 Importing Plugins from a Feature Archive File 9-6

9.6 Using a Plugin in a MAF Application 9-6

10 Using Lifecycle Listeners in MAF Applications

10.1 Introduction to Lifecycle Listeners in MAF Applications 10-1
10.2 Registering a Lifecycle Listener for a MAF Application or an Application Feature 10-3
10.3 What Happens When You Register a Lifecycle Listener 10-4

11 Creating MAF AMX Pages

11.1 Introduction to the MAF AMX Application Feature 11-1
11.2 Creating Task Flows 11-1
11.2.1 How to Create a Task Flow 11-2
11.2.2 What You May Need to Know About Task Flow Activities and Control Flows 11-3
11.2.3 What You May Need to Know About the task-flow-definition.xml File 11-5
11.2.4 What You May Need to Know About the MAF Task Flow Diagrammer 11-5
11.2.5 How to Add and Use Task Flow Activities 11-5
11.2.5.1 Adding View Activities 11-6
11.2.5.2 Adding Router Activities 11-8
11.2.5.3 Adding Method Call Activities 11-9
11.2.5.4 Adding Task Flow Call Activities 11-11
11.2.5.4.1 Calling a Bounded Task Flow Using a Task Flow Call Activity 11-11
11.2.5.4.2 Specifying Input Parameters on a Task Flow Call Activity 11-13
11.2.5.5 Adding Task Flow Return Activities 11-13
11.2.5.6 Using Task Flow Activities with Page Definition Files 11-14
11.2.6 How to Define Control Flows 11-15
11.2.6.1 Defining a Control Flow Case 11-15
11.2.6.2 Adding a Wildcard Control Flow Rule 11-16
11.2.6.3 What You May Need to Know About Control Flow Rule Metadata 11-16
11.2.6.4 What You May Need to Know About Control Flow Rule Evaluation 11-17
11.2.7 What You May Need to Know About MAF Support for Back Navigation 11-18
11.2.8 How to Enable Page Navigation by Dragging 11-18
11.2.9 How to Specify Action Outcomes Using UI Components 11-18
11.2.10 How to Create and Reference Managed Beans 11-19
11.2.11 How to Specify the Page Transition Style 11-23
11.2.12 What You May Need to Know About Bounded and Unbounded Task Flows 11-24
11.2.12.1 Unbounded Task Flows 11-24
11.2.12.2 Bounded Task Flows 11-25
11.2.12.3 Using Parameters in Task Flows 11-27
11.2.12.3.1 Passing Parameters to a Bounded Task Flow 11-27
11.2.12.3.2 Configuring a Return Value from a Bounded Task Flow 11-30
11.3 Creating Views 11-32
11.3.1 How to Work with MAF AMX Pages 11-32
11.3.1.1 Interpreting the MAF AMX Page Structure 11-32
11.3.1.2 Creating MAF AMX Pages 11-32
11.3.1.3 What Happens When You Create an MAF AMX Page 11-34
11.3.1.4 Using UI Editors 11-35
11.3.1.5 Accessing the Page Definition File 11-35
11.3.1.6 Sharing the Page Contents 11-37

11.3.2 How to Add UI Components and Data Controls to an MAF AMX Page 11-40
11.3.2.1 Adding UI Components 11-40
11.3.2.2 Using the Preview 11-41
11.3.2.3 Adding Data Controls to the View 11-41
11.3.2.3.1 Dragging and Dropping Attributes 11-43
11.3.2.3.2 Dragging and Dropping Operations 11-46
11.3.2.3.3 Dragging and Dropping Collections 11-48
11.3.2.3.4 What You May Need to Know About Generated Bindings 11-52
11.3.2.3.5 What You May Need to Know About Generated Drag and Drop Artifacts 11-54
11.3.2.3.6 Using the MAF AMX Editor Bindings Tab 11-57
11.3.2.3.7 What You May Need to Know About Removal of Unused Bindings 11-57
11.3.2.4 What You May Need to Know About Element Identifiers and Their Audit .. 11-59
11.3.3 What You May Need to Know About the Server Communication 11-61

12 Creating the MAF AMX User Interface

12.1 Introduction to Creating the User Interface for MAF AMX Pages 12-1
12.2 Designing the Page Layout 12-2
12.2.1 How to Use a View Component 12-5
12.2.2 How to Use a Panel Page Component 12-5
12.2.3 How to Use a Panel Group Layout Component 12-6
12.2.3.1 Customizing the Scrolling Behavior 12-6
12.2.4 How to Use a Panel Form Layout Component 12-7
12.2.5 How to Use a Panel Stretch Layout Component 12-8
12.2.6 How to Use a Panel Label And Message Component 12-8
12.2.7 How to Use a Facet Component 12-9
12.2.8 How to Use a Popup Component 12-10
12.2.9 How to Use a Panel Splitter Component 12-12
12.2.10 How to Use a Spacer Component 12-13
12.2.11 How to Use a Table Layout Component 12-14
12.2.12 How to Use a Masonry Layout Component 12-15
12.2.13 How to Use an Accessory Layout Component 12-17
12.2.14 How to Use a Deck Component 12-19
12.2.15 How to Use a Flex Layout Component 12-20
12.2.16 How to Use the Fragment Component 12-21
12.3 Creating and Using UI Components 12-22
12.3.1 How to Use the Input Text Component 12-24
12.3.1.1 Customizing the Input Text Component 12-25
12.3.2 How to Use the Input Number Slider Component 12-27
12.3.3 How to Use the Input Date Component 12-28
12.3.4 How to Use the Output Text Component 12-29
12.3.5 How to Use Buttons 12-29
12.3.5.1 Displaying Default Style Buttons 12-31
12.3.5.2 Displaying Back Style Buttons 12-32
12.3.5.3 Displaying Highlight Style Buttons 12-32
12.3.5.4 Displaying Alert Style Buttons 12-33
12.3.5.5 Using Additional Button Styles 12-33

12.3.5.6 Using Buttons Within the Application 12-34
12.3.5.7 Enabling the Back Button Navigation 12-35
12.3.5.8 What You May Need to Know About the Order of Processing Operations and Attributes 12-35
12.3.6 How to Use Links 12-36
12.3.7 How to Display Images 12-37
12.3.8 How to Use the Checkbox Component 12-38
12.3.8.1 Support for Checkbox Components on iOS Platform 12-39
12.3.8.2 Support for Checkbox Components on Android Platform 12-39
12.3.9 How to Use the Select Many Checkbox Component 12-39
12.3.9.1 What You May Need to Know About the User Interaction with Select Many Checkbox Component 12-40
12.3.10 How to Use the Choice Component 12-40
12.3.10.1 What You May Need to Know About the User Interaction with Choice Component on iOS Platform 12-41
12.3.10.2 What You May Need to Know About the User Interaction with Choice Component on Android Platform 12-42
12.3.10.3 What You May Need to Know About Differences Between Select Items and Select Item Components 12-42
12.3.11 How to Use the Select Many Choice Component 12-42
12.3.12 How to Use the Boolean Switch Component 12-43
12.3.12.1 What You May Need to Know About Support for Boolean Switch Components on iOS Platform 12-44
12.3.13 How to Use the Select Button Component 12-44
12.3.14 How to Use the Radio Button Component 12-45
12.3.15 How to Use List View and List Item Components 12-46
12.3.15.1 Configuring Paging and Dynamic Scrolling 12-55
12.3.15.2 Rearranging List View Items 12-57
12.3.15.3 Configuring The List View Layout 12-58
12.3.15.4 What You May Need to Know About Using a Static List View 12-65
12.3.16 How to Use the Carousel Component 12-65
12.3.17 How to Use the Film Strip Component 12-67
12.3.17.1 What You May Need to Know About the Film Strip Layout 12-69
12.3.17.2 What You May Need to Know About the Film Strip Navigation 12-69
12.3.18 How to Use Verbatim Component 12-69
12.3.18.1 What You May Need to Know About Using JavaScript and AJAX with Verbatim Component 12-70
12.3.19 How to Use Output HTML Component 12-70
12.3.20 How to Enable Iteration 12-71
12.3.21 How to Refresh Contents of UI Components 12-71
12.3.22 How to Load a Resource Bundle 12-73
12.3.23 How to Use the Action Listener 12-74
12.3.23.1 What You May Need to Know About Differences Between the Action Listener Component and Attribute 12-74
12.3.24 How to Use the Set Property Listener 12-75
12.3.25 How to Use the Client Listener 12-76
12.3.26 How to Convert Date and Time Values 12-78
12.3.26.1 What You May Need to Know About Date and Time Patterns 12-79
12.3.27 How to Convert Numerical Values 12-80

12.3.28 How to Enable Drag Navigation 12-81
12.3.28.1 What You May Need to Know About the disabled Attribute 12-83
12.3.29 How to Use the Loading Indicator 12-84
12.4 Enabling Gestures 12-86
12.5 Providing Data Visualization 12-88
12.5.1 How to Create an Area Chart 12-90
12.5.2 How to Create a Bar Chart 12-92
12.5.3 How to Create a Range Chart 12-93
12.5.4 How to Create a Bubble Chart 12-94
12.5.5 How to Create a Combo Chart 12-96
12.5.6 How to Create a Line Chart 12-97
12.5.7 How to Create a Pie Chart 12-100
12.5.7.1 Configuring the Pie Chart as a Ring Chart 12-102
12.5.7.2 Styling the Pie Chart 12-102
12.5.8 How to Create a Scatter Chart 12-102
12.5.9 How to Create a Spark Chart 12-104
12.5.10 How to Create a Funnel Chart 12-105
12.5.11 How to Create a Stock Chart 12-106
12.5.12 How to Style Chart Components 12-110
12.5.13 How to Use Events with Chart Components 12-111
12.5.14 What You May Need to Know About Customization of Chart Tooltips 12-111
12.5.15 How to Enable Sorting of Charts with Categorical Axis 12-111
12.5.16 How to Define the Initial Zooming of Charts 12-112
12.5.17 How to Define Stacking of Specific Chart Series 12-112
12.5.18 How to Enable Split Dual-Y Axis in Charts 12-112
12.5.19 How to Create an LED Gauge 12-112
12.5.20 How to Create a Status Meter Gauge 12-113
12.5.21 How to Create a Dial Gauge 12-114
12.5.22 How to Create a Rating Gauge 12-115
12.5.22.1 Overwriting the shortDesc Attribute 12-116
12.5.22.2 Applying Custom Styling to the Rating Gauge Component 12-116
12.5.23 How to Define Child Elements for Chart and Gauge Components 12-117
12.5.23.1 Defining Chart Data Item 12-118
12.5.23.2 Defining and Configuring Legend 12-118
12.5.23.3 Defining X Axis, YAxis, and Y2Axis 12-119
12.5.23.4 Defining Pie Data Item 12-119
12.5.23.5 Defining Spark Data Item 12-119
12.5.23.6 Defining Funnel Data Item 12-119
12.5.23.7 Defining Stock Data Item 12-119
12.5.23.8 Defining Threshold 12-119
12.5.24 How to Create a Geographic Map Component 12-119
12.5.24.1 Configuring Geographic Map Components With the Map Provider Information 12-121
12.5.24.2 Displaying Route in Geographic Map Components 12-123
12.5.25 How to Create a Thematic Map Component 12-124
12.5.25.1 Defining Custom Markers 12-125
12.5.25.2 Defining Isolated Area Layers 12-125

12.5.25.3 Defining Isolated Areas 12-126
12.5.25.4 Enabling Initial Zooming 12-126
12.5.25.5 Defining a Custom Base Map 12-126
12.5.25.6 What You May Need to Know About the Marker Support for Event Listeners 12-128
12.5.25.7 Applying Custom Styling to the Thematic Map Component 12-128
12.5.26 How to Use Events with Map Components 12-130
12.5.27 How to Create a Treemap Component 12-131
12.5.27.1 Applying Custom Styling to the Treemap Component 12-132
12.5.28 How to Create a Sunburst Component 12-134
12.5.28.1 Applying Custom Styling to the Sunburst Component 12-135
12.5.29 How to Create a Timeline Component 12-136
12.5.29.1 Applying Custom Styling to the Timeline Component 12-138
12.5.30 How to Create an NBox Component 12-140
12.5.31 How to Define Child Elements for Map Components, Sunburst, Treemap, Timeline, and NBox 12-142
12.5.32 How to Create Databound Data Visualization Components 12-142
12.5.32.1 What You May Need to Know About Setting Series Style for Databound Chart Components 12-150
12.5.33 How to Create Data Visualization Components Based on Static Data 12-150
12.5.34 How to Enable Interactivity in Chart Components 12-151
12.5.35 How to Create Polar Charts 12-152
12.6 Styling UI Components 12-152
12.6.1 How to Use Component Attributes to Define Style 12-152
12.6.2 What You May Need to Know About Skinning 12-153
12.6.3 What You May Need to Know About Using CSS ID Selectors for Skinning 12-153
12.6.4 How to Style Data Visualization Components 12-154
12.7 Localizing UI Components 12-156
12.8 Understanding MAF Support for Accessibility 12-159
12.8.1 How to Configure UI and Data Visualization Components for Accessibility 12-160
12.8.2 What You May Need to Know About the Basic WAI-ARIA Terms 12-165
12.8.3 What You May Need to Know About the Oracle Global HTML Accessibility Guidelines 12-167
12.9 Validating Input 12-167
12.10 Using Event Listeners 12-170
12.10.1 What You May Need to Know About Constrained Type Attributes for Event Listeners 12-173

13 Using Bindings and Creating Data Controls in MAF AMX

13.1 Introduction to Bindings and Data Controls 13-1
13.2 About Object Scope Lifecycles 13-2
13.2.1 What You May Need to Know About Object Scopes and Task Flows 13-3
13.3 Creating EL Expressions 13-3
13.3.1 About Data Binding EL Expressions 13-4
13.3.2 How to Create an EL Expression 13-5
13.3.2.1 About the Method Expression Builder 13-7
13.3.2.2 About Non EL-Properties 13-8
13.3.3 What You May Need to Know About MAF Binding Properties 13-9

13.3.4 How to Reference Binding Containers ... 13-9
13.3.5 About the Categories in the Expression Builder ... 13-11
13.3.5.1 About the Bindings Category ... 13-11
13.3.5.2 About the Managed Beans Category ... 13-15
13.3.5.3 About the Mobile Application Framework Objects Category ... 13-17
13.3.6 About EL Events ... 13-18
13.3.7 How to Use EL Expressions Within Managed Beans ... 13-19
13.4 Creating and Using Managed Beans ... 13-19
13.4.1 How to Create a Managed Bean in OEPE ... 13-20
13.4.2 What Happens When You Use OEPE to Create a Managed Bean ... 13-22
13.5 Exposing Business Services with Data Controls ... 13-22
13.5.1 How to Create Data Controls ... 13-22
13.5.2 What Happens in Your Project When You Create a Data Control ... 13-23
13.5.2.1 The Data Control Manager ... 13-23
13.5.2.2 The Palette ... 13-24
13.5.3 Data Control Built-in Operations ... 13-24
13.6 Creating Databound UI Components from the Data Controls Palette ... 13-25
13.6.1 How to Use the Data Controls Palette ... 13-26
13.6.2 What Happens When You Use the Data Controls Palette ... 13-28
13.7 What Happens at Runtime: How the Binding Context Works ... 13-28
13.8 Working with Data Control Attributes ... 13-30
13.8.1 Setting UI Hints on Attributes ... 13-30
13.9 Creating and Using Bean Data Controls ... 13-31
13.9.1 What You May Need to Know About Serialization of Bean Class Variables ... 13-32
13.10 Using the DeviceFeatures Data Control ... 13-33
13.10.1 How to Use the getPicture Method to Enable Taking Pictures ... 13-35
13.10.2 How to Use the SendSMS Method to Enable Text Messaging ... 13-39
13.10.3 How to Use the sendEmail Method to Enable Email ... 13-41
13.10.4 How to Use the createContact Method to Enable Creating Contacts ... 13-45
13.10.5 How to Use the findContacts Method to Enable Finding Contacts ... 13-49
13.10.6 How to Use the updateContact Method to Enable Updating Contacts ... 13-52
13.10.7 How to Use the removeContact Method to Enable Removing Contacts ... 13-55
13.10.8 How to Use the startLocationMonitor Method to Enable Geolocation ... 13-56
13.10.9 How to Use the displayFile Method to Enable Displaying Files ... 13-59
13.10.10 How to Use the addLocalNotification and cancelLocalNotification Methods to Manage Local Notifications 13-61
13.10.11 What You May Need to Know About Device Properties ... 13-62
13.11 Validating Attributes ... 13-64
13.12 About Data Change Events ... 13-66

14 Configuring End Points Used in MAF Applications

14.1 Introduction to Configuring End Points ... 14-1
14.2 Defining the Configuration Service End Point ... 14-1
14.3 Creating the User Interface for the Configuration Service ... 14-3
14.4 About the URL Construction ... 14-4
14.5 Setting Up the Configuration Service on the Server ... 14-4

15 Using Web Services in a MAF Application

15.1 Introduction to Using Web Services in a MAF Application 15-1
15.2 Creating a Rest Service Adapter to Access Web Services 15-1
15.2.1 Accessing Input and Output Streams 15-4
15.2.2 Support for Non-Text Responses 15-6
15.3 Accessing Secure Web Services 15-7
15.3.1 How to Enable Access to Web Services 15-8
15.3.2 What Happens When You Enable Access to Web Services 15-9
15.3.3 What You May Need to Know About Accessing Web Services and Containerized MAF Applications 15-9
15.3.4 What You May Need to Know About Credential Injection 15-10
15.3.5 What You May Need to Know About Cookie Injection 15-11
15.4 Configuring the Browser Proxy Information 15-11

16 Working with REST Services

16.1 Introduction to Working with REST Services 16-1
16.1.1 REST Client Page 16-2
16.1.2 REST API Page 16-3
16.1.3 Data Types Page 16-4
16.1.4 Using Authentication 16-6
16.1.5 How to Open the REST Service Editor 16-6
16.1.6 How to Create a REST Service Description 16-6
16.2 Using the REST Client 16-6
16.2.1 Specifying REST Service Connections 16-7
16.2.1.1 How to Enter a Simple URI 16-7
16.2.1.2 How to Compose an Address to a REST Service 16-8
16.2.1.3 How to Include Queries in the Request 16-8
16.2.1.4 How to Include Fragments in the Request 16-8
16.2.1.5 How to Use Connection Names from the Application 16-8
16.2.1.6 How to Create Persistent Connections 16-9
16.2.1.7 Overriding OWSM Policies 16-10
16.2.1.8 How to Use Authentication 16-11
16.2.2 Sending Requests to the REST Service 16-12
16.2.3 What Happens When You Send a Request 16-12
16.2.4 Generating Response Types 16-13
16.3 Modifying the Request Content 16-13
16.3.1 How to Use REST Headers in the Request 16-14
16.3.2 How to Use Query Parameters to Configure the Response. 16-14
16.3.3 How to Send Input 16-14
16.3.4 How to Specify Output 16-15
16.4 Importing and Modeling the REST API 16-15
16.4.1 Importing REST Client Information 16-15
16.4.2 What Happens When You Import REST Client Information 16-17
16.4.3 Manually Modeling the REST API 16-18
16.4.3.1 How to Create New Requests 16-19
16.4.3.2 How to Create New Paths 16-19
16.4.4 Working with Path Variable Types 16-19

16.4.4.1 How to Edit a Path Variable Type 16-21
16.4.4.2 Path Variable Types in Artifact Generation 16-21
16.4.4.3 Path Variable Types and the REST Client 16-21
16.5 Importing and Modeling Data Types 16-22
16.5.1 How to Create Data Types 16-23
16.5.2 How to Import Data Types 16-24
16.6 Testing Modeled Requests Against the REST Service 16-24
16.7 Creating REST Service Artifacts 16-25
16.7.1 How to Generate Java Artifacts for REST Services 16-25
16.7.2 What Happens When You Generate Java Artifacts 16-26
16.7.3 About the Generated Artifacts for REST Services 16-26
16.7.3.1 Generated Data Type Artifacts 16-27
16.7.3.2 Generated REST API Artifacts 16-28
16.7.3.3 Generated Service Artifacts 16-29
16.7.4 How to Use the Generated Artifacts in Your MAF Application 16-30
16.8 Using the Connections View 16-31
16.8.1 How to Open the Connections View 16-31
16.8.2 How to Edit a Connection 16-31
16.8.3 How to Reuse a Connection 16-32
16.8.4 How to Open connections.xml 16-32

17 Working with Oracle Cloud Services

17.1 Introduction to Working with Oracle Cloud Services 17-1
17.2 Introduction to Working with MCS 17-1
17.2.1 Getting Started with MCS 17-2
17.2.1.1 Consuming MCS Services in MAF Applications 17-2
17.2.1.2 Developing Mobile Clients for MCS Mobile Services 17-3
17.2.1.3 Using Mobile Backends 17-3
17.2.1.4 Importing a REST API 17-3
17.2.1.4.1 Importing a REST API from Oracle Cloud View 17-4
17.2.1.4.2 Importing a REST API Using the REST Service Editor 17-5
17.3 Enabling Mobile Applications to Access Data Hosted on Oracle Cloud 17-6
17.3.1 How to Authenticate Against Oracle Cloud 17-6
17.3.2 How to Create a Web Service Data Control to Access Oracle Java Cloud 17-7
17.3.2.1 Configuring the Policy for Web Services 17-10
17.3.3 What Happens When You Deploy a Mobile Application that Accesses Oracle Java Cloud Service 17-10

18 Creating Custom MAF AMX UI Components

18.1 Introduction to Creating Custom UI Components 18-1
18.2 Using MAF APIs to Create Custom Components 18-1
18.2.1 How to Use Static APIs 18-2
18.2.2 How to Use AmxEvent Classes 18-6
18.2.3 How to Use the TypeHandler 18-7
18.2.4 How to Use the AmxNode 18-9
18.2.5 How to Use the AmxTag 18-15

18.2.6 How to Use the VisitContext ... 18-18
18.2.7 How to Use the AmxAttributeChange ... 18-19
18.2.8 How to Use the AmxDescendentChanges ... 18-20
18.2.9 How to Use the AmxCollectionChange ... 18-20
18.2.10 How to Use the AmxNodeChangeResult ... 18-20
18.2.11 How to Use the AmxNodeStates ... 18-21
18.2.12 How to Use the AmxNodeUpdateArguments ... 18-21
18.3 Creating Custom Components ... 18-22

19 Using the Local Database in MAF AMX

19.1 Introduction to the Local SQLite Database Usage ... 19-1
19.1.1 Differences Between SQLite and Other Relational Databases ... 19-1
19.1.1.1 Concurrency ... 19-2
19.1.1.2 SQL Support and Interpretation ... 19-2
19.1.1.3 Data Types ... 19-2
19.1.1.4 Foreign Keys ... 19-2
19.1.1.5 Database Transactions ... 19-2
19.1.1.6 Authentication ... 19-3
19.2 Using the Local SQLite Database ... 19-3
19.2.1 How to Connect to the Database ... 19-3
19.2.2 How to Use SQL Script to Initialize the Database ... 19-4
19.2.3 How to Initialize the Database on a Desktop ... 19-6
19.2.4 What You May Need to Know About Commit Handling ... 19-7
19.2.5 Limitations of the MAF's SQLite JDBC Driver ... 19-7
19.2.6 How to Use the VACUUM Command ... 19-7
19.2.7 How to Encrypt and Decrypt the Database ... 19-7
19.2.8 What You May Need to Know About the StockTracker Sample Application ... 19-9

20 Implementing Application Feature Content Using Remote URLs

20.1 Introduction to Remote URL Applications ... 20-1
20.2 Enabling Remote Applications Access Container Services ... 20-2
20.3 Whitelisting Remote URLs in Your MAF Application ... 20-3
20.3.1 How to Whitelist Remote URLs on the Android Platform ... 20-4
20.3.2 How to Whitelist Remote URLs on the iOS Platform ... 20-5
20.3.3 How to Whitelist Remote URLs on Universal Windows Platform ... 20-6
20.4 Enabling the Browser Navigation Bar on Remote URL Pages ... 20-7
20.4.1 How to Add the Navigation Bar to a Remote URL Application Feature ... 20-7
20.4.2 What Happens When You Enable the Browser Navigation Buttons for a Remote URL Application Feature 20-8

21 Enabling User Preferences

21.1 Creating User Preference Pages for a Mobile Application ... 21-1
21.1.1 How to Create Mobile Application-Level Preferences Pages ... 21-4
21.1.1.1 How to Create a New User Preference Page ... 21-5
21.1.1.2 What Happens When You Add a Preference Page ... 21-7
21.1.1.3 How to Create User Preference Lists ... 21-7

21.1.1.4 What Happens When You Create a Preference List ... 21-9
21.1.1.5 How to Create a Boolean Preference List ... 21-9
21.1.1.6 What Happens When You Add a Boolean Preference ... 21-10
21.1.1.7 How to Add a Text Preference ... 21-11
21.1.1.8 What Happens When You Define a Text Preference ... 21-12
21.1.2 What Happens When You Create an Application-Level Preference Page ... 21-12
21.2 Creating User Preference Pages for Application Features ... 21-12
21.3 Using EL Expressions to Retrieve Stored Values for User Preference Pages ... 21-13
21.3.1 What You May Need to Know About preferenceScope ... 21-14
21.3.2 Reading Preference Values in iOS Native Views ... 21-15
21.4 Platform-Dependent Display Differences ... 21-15

22 Setting Constraints on Application Features

22.1 Introduction to Constraints ... 22-1
22.1.1 Using Constraints to Show or Hide an Application Feature ... 22-1
22.1.2 Using Constraints to Deliver Specific Content Types ... 22-2
22.2 Defining Constraints for Application Features ... 22-3
22.2.1 How to Define the Constraints for an Application Feature ... 22-3
22.2.2 What Happens When You Define a Constraint ... 22-4
22.2.3 About the property Attribute ... 22-4
22.2.4 About User Constraints and Access Control ... 22-4
22.2.5 About Hardware-Related Constraints ... 22-6
22.2.6 Creating Dynamic Constraints on Application Features and Content ... 22-12
22.2.6.1 About Combining Static and EL-Defined Constraints ... 22-12
22.2.6.2 How to Define a Dynamic Constraint ... 22-12

23 Using AppXray for MAF Artifacts

23.1 Introduction to Using AppXray for MAF Artifacts ... 23-1
23.2 Using AppXray ... 23-2
23.2.1 How to Open AppXaminer ... 23-2
23.2.1.1 About AppXaminer ... 23-2
23.2.2 Using AppXaminer ... 23-3
23.3 Refactoring with AppXray ... 23-3

24 Enabling and Using Notifications

24.1 Introduction to Notifications ... 24-1
24.2 Enabling Push Notifications ... 24-3
24.2.1 What You May Need to Know About the Push Notification Payload ... 24-5
24.3 Managing Local Notifications ... 24-5
24.3.1 How to Manage Local Notifications Using Java ... 24-5
24.3.2 How to Manage Local Notifications Using JavaScript ... 24-6
24.3.3 How to Manage Local Notifications Using the DeviceFeatures Data Control ... 24-8
24.3.4 How to Handle Local Notifications ... 24-8
24.3.5 What You May Need to Know About Local Notification Options and the Application Behavior 24-9

25 Caching Data in a MAF Application

25.1 Introduction to Data Caching in MAF Applications 25-1
25.2 Enable Data Caching in a MAF Application 25-2
25.3 Specifying Cached Resources and Cache Policies in the sync-config.xml File 25-3
25.4 Caching Policies Provided by MAF 25-4
25.5 Using Configuration Service End Points in the sync-config.xml File 25-6
25.6 Encrypting Cached Data in a MAF Application 25-6
25.7 Packaging the sync-config.xml File in a FAR 25-7

26 Displaying Error Messages in MAF Applications

26.1 Introduction to Error Handling in MAF Applications 26-1
26.2 Displaying Error Messages and Stopping Background Threads 26-2
26.2.1 How Applications Display Error Message for Background Thread Exceptions 26-3
26.3 Localizing Error Messages 26-4

27 Deploying MAF Applications

27.1 Introduction to Deployment of MAF Applications 27-1
27.2 Working with Deployment Configurations 27-2
27.2.1 About Deployment Configurations 27-2
27.2.2 Differences Between Run Configurations and Debug Configurations 27-3
27.2.3 How to Create a Deployment Configuration 27-4
27.2.4 What Happens When You Create a Deployment Configuration 27-4
27.3 Deploying an Android Application 27-5
27.3.1 How to Create an Android Deployment Configuration 27-6
27.3.1.1 Setting Advanced Options 27-9
27.3.1.2 Setting Deployment Options 27-11
27.3.1.3 Defining the Android Signing Options 27-11
27.3.1.4 What You May Need to Know About Credential Storage 27-14
27.3.1.5 How to Add a Custom Image to an Android Application 27-14
27.3.1.6 What Happens When OEPE Deploys Images for Android Applications 27-15
27.3.2 How to Deploy an Android Application to an Android Emulator 27-16
27.3.3 How to Deploy an Application Locally as an APK File 27-17
27.3.4 How to Deploy an Application to an Android-Powered Device 27-18
27.3.5 How to Publish an Android Application 27-19
27.3.6 What Happens in OEPE When You Create an .apk File 27-19
27.3.7 Selecting the Most Recently Used Deployment Configurations 27-20
27.3.8 Viewing the Device/Emulator Log in the Console 27-20
27.4 Deploying an iOS Application 27-21
27.4.1 How to Create an iOS Deployment Configuration 27-22
27.4.2 Setting the Device Signing Options 27-26
27.4.3 How to Restrict the Display to a Specific Device Orientation 27-27
27.4.4 What Happens When You Deselect Device Orientations 27-29
27.4.5 Using Images with iOS Applications 27-29
27.4.5.1 Overriding the Default Oracle Images in the Configuration 27-31
27.4.5.2 Replacing the Default Images 27-32
27.4.5.3 Add Custom Images 27-32

27.4.6 How to Deploy an iOS Application to an iOS Simulator 27-32
27.4.6.1 Using Quick Deploy 27-34
27.4.7 How to Deploy an Application to an iOS-Powered Device 27-35
27.4.8 What Happens When You Deploy an Application to an iOS Device 27-39
27.4.9 What You May Need to Know About Deploying an Application to an iOS-Powered Device 27-39
27.4.9.1 Creating iOS Development Certificates 27-40
27.4.9.2 Registering an Apple Device for Testing and Debugging 27-40
27.4.9.3 Registering an Application ID 27-40
27.4.10 How to Distribute an iOS Application to the App Store 27-41
27.5 Deploying a MAF Application to the Universal Windows Platform 27-43
27.5.1 How to Create a Deployment Configuration for Universal Windows Platform .. 27-43
27.5.2 Defining the Windows Platform Signing Options 27-45
27.5.3 How to Deploy a MAF Application to the Universal Windows Platform 27-46
27.5.4 What Happens When You Deploy a MAF Application to the Universal Windows Platform 27-46
27.5.5 How to Deploy an Application Locally as a Package 27-47
27.5.6 What Happens When You Deploy Locally as a Package 27-47
27.6 Deploying Feature Archive Files (FARs) 27-48
27.6.1 How to Create a Mobile Feature Archive File 27-49
27.6.2 How to Deploy the Feature Archive 27-50
27.6.3 What Happens When You Create a Feature Archive File 27-51
27.7 Creating a Mobile Application Archive File 27-51
27.7.1 How to Create a Mobile Application Archive File 27-52
27.8 Creating Unsigned Deployment Packages 27-55
27.8.1 How to Create an Unsigned Application 27-55
27.8.2 What Happens When You Import a MAF Application Archive File 27-55
27.9 Deploying with Oracle Mobile Security Suite 27-55
27.9.1 What Happens When You Containerize Your App with OMSS 27-57

28 Understanding Secure Mobile Development Practices

28.1 Weak Server-Side Controls 28-1
28.2 Insecure Data Storage on the Device 28-2
28.2.1 Encrypting the SQLite Database 28-2
28.2.2 Securing the Device's Local Data Stores 28-2
28.2.3 About Security and Application Logs 28-3
28.3 Insufficient Transport Layer Protection 28-3
28.4 Side-Channel Data Leakage 28-3
28.5 Poor Authorization and Authentication 28-4
28.6 Broken Cryptography 28-4
28.7 Client-Side Injection From Cross-Site Scripting 28-5
28.7.1 Protecting MAF Applications from Injection Attacks Using Device Access Permissions 28-5
28.7.2 About Injection Attack Risks from Custom HTML Components 28-6
28.7.3 About SQL Injections and XML Injections 28-6
28.8 Security Decisions From Untrusted Inputs 28-6
28.9 Improper Session Handling 28-7

28.10 Lack of Binary Protections Resulting in Sensitive Information Disclosure 28-8

29 Securing MAF Applications

29.1 Introduction to MAF Security 29-1
29.2 About the User Login Process 29-2
29.3 Overview of the Authentication Process for Mobile Applications 29-4
29.4 Overview of the Authentication Process for Containerized MAF Applications 29-5
29.5 Configuring MAF Connections 29-6
29.5.1 How to Create a MAF Login Connection 29-6
29.5.2 How to Create a Multi-Tenant Aware MAF Login Connection 29-7
29.5.3 How to Configure Basic Authentication 29-10
29.5.4 How to Configure OAuth Authentication 29-12
29.5.5 How to Configure Web SSO Authentication 29-14
29.5.6 How to Update Connection Attributes of a Named Connection at Runtime 29-17
29.5.7 How to Store Login Credentials 29-20
29.5.8 What Happens When You Create a Connection for a MAF Application 29-21
29.5.9 What Happens When You Create a Multi-Tenant Aware Connection 29-22
29.5.10 What You May Need to Know About the Login Connection Configuration 29-22
29.5.11 What You May Need to Know About Login Connections and Containerized MAF Applications 29-22
29.5.12 What You May Need to Know About Multiple Identities for Local and Hybrid Login Connections 29-22
29.5.13 What You May Need to Know About Migrating a MAF Application and Authentication Modes 29-23
29.5.14 What You May Need to Know About Custom Headers 29-23
29.5.15 What Happens at Runtime: When MAF Calls a REST Web Service 29-23
29.5.16 What You May Need to Know About Injecting Basic Authentication Headers ... 29-23
29.5.17 What You May Need to Know about Web Service Security 29-24
29.5.18 How to Configure Access Control 29-25
29.5.19 What You May Need to Know About the Access Control Service 29-26
29.5.20 How to Alter the Application Loading Sequence 29-28
29.5.21 How to Configure Login Credentials Programmatically Prior to Authentication 29-29
29.6 Configuring Security for Mobile Applications 29-31
29.6.1 How to Enable Application Features to Require Authentication 29-32
29.6.2 How to Designate the Login Page 29-33
29.6.3 How to Create a Custom Login HTML Page 29-35
29.6.4 What You May Need to Know About Login Pages 29-36
29.6.4.1 The Default Login Page 29-36
29.6.4.2 The Custom Login Page 29-37
29.6.5 What You May Need to Know About Login Page Elements 29-39
29.6.6 What Happens in OEPE When You Configure Security for Application Features 29-39
29.7 Allowing Access to Device Capabilities 29-39
29.8 Enabling Users to Log Out from Application Features 29-40
29.9 Using MAF Authentication APIs 29-41
29.10 Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL 29-42
29.11 Configuring a MAF Application to Enable Two-Way SSL for Authentication 29-43

30 Testing and Debugging MAF Applications

30.1 Introduction to Testing and Debugging MAF Applications 30-1
30.2 Testing MAF Applications 30-2
30.2.1 How to Perform Accessibility Testing on iOS-Powered Devices 30-2
30.3 Configuring OEPE and MAF Applications to Debug Code 30-2
30.3.1 What You May Need to Know About the Debugging Configuration 30-3
30.3.1.1 Creating and Configuring a Debug Configuration 30-3
30.3.2 How to Enable Debugging of Java Code and JavaScript 30-4
30.3.2.1 What You May Need to Know About Debugging of JavaScript Using an iOS-Powered Device Simulator on iOS 6 Platform 30-6
30.3.3 How to Debug the MAF AMX Content 30-7
30.4 Debugging MAF Applications Deployed on the Android Platform 30-7
30.4.1 How to Debug Java Code on the Android Platform 30-7
30.4.1.1 Troubleshooting adb 30-8
30.4.2 How to Debug UI Code on the Android Platform 30-8
30.5 Debugging MAF Applications Deployed on the iOS Platform 30-9
30.5.1 How to Debug Java Code on the iOS Platform 30-10
30.5.2 How to Debug UI Code on the iOS Platform 30-10
30.6 Using and Configuring Logging in MAF Applications 30-15
30.6.1 How to Configure Logging Using the Properties File 30-17
30.6.2 How to Use JavaScript Logging 30-19
30.6.3 How to Use Embedded Logging 30-20
30.6.4 How to Use Xcode for Debugging and Logging on iOS Platform 30-20
30.6.5 How to Access the Application Log 30-20
30.6.6 How to Disable Logging 30-21
30.7 Measuring MAF Application Performance 30-21
30.8 Sending Diagnostic Information to Oracle Mobile Cloud Service 30-28
30.9 Sending Analytics Information to Oracle Mobile Cloud Service 30-29
30.9.1 How to Configure the Transfer of Analytics to Oracle Mobile Cloud Service 30-30
30.9.2 How to Programmatically Send Analytics to Oracle Mobile Cloud Service 30-32
30.9.3 How to Send Context Events to Oracle Mobile Cloud Service 30-34
30.9.4 How to Send Analytics to Other Repositories 30-35
30.9.5 MAF Framework Events that Capture Analytics Information 30-37

A Troubleshooting MAF Applications

A.1 Problems with Input Components on iOS Simulators A-1
A.2 Code Signing Issues Prevent Deployment A-2

B Local HTML and Application Container APIs

B.1 Using MAF APIs to Create a Custom HTML Springboard Application Feature B-1
B.1.1 About Executing Code in Custom HTML Pages B-2
B.2 The MAF Container Utilities API B-2
B.2.1 Using the JavaScript Callbacks B-3
B.2.2 Using the Container Utilities API B-4
B.2.3 getApplicationInformation B-5
B.2.4 gotoDefaultFeature B-6

B.2.5 gotoFeature ... B-6
B.2.6 getFeatures ... B-7
B.2.7 getFeatureByName ... B-8
B.2.8 getFeatureById ... B-9
B.2.9 resetFeature ... B-9
B.2.10 resetApplication ... B-10
B.2.11 gotoSpringboard ... B-11
B.2.12 showSpringboard ... B-11
B.2.13 hideSpringboard ... B-12
B.2.14 showNavigationBar ... B-13
B.2.15 hideNavigationBar ... B-13
B.2.16 showPreferences ... B-14
B.2.17 invokeMethod ... B-14
B.2.18 invokeContainerMethod ... B-15
B.2.19 invokeContainerJavaScriptFunction ... B-16
B.2.20 sendEmail ... B-17
B.2.21 sendSMS ... B-17
B.2.22 Application Icon Badging ... B-17
B.3 Accessing Files Using the getDirectoryPathRoot Method ... B-18
B.3.1 Accessing Platform-Independent Download Locations ... B-18

C MAF Application and Project Files

C.1 Introduction to MAF Application and Project Files ... C-1
C.2 About the Assembly-Level Resources ... C-2
C.3 About the Application Project Resources ... C-4
C.4 About the View Project Resources ... C-5

D Converting Preferences for Deployment

D.1 Naming Patterns for Preferences ... D-1
D.2 Converting Preferences for Android ... D-2
D.2.1 maf_preferences.xml ... D-3
D.2.1.1 Preferences Element Mapping ... D-3
D.2.1.2 Preference Attribute Mapping ... D-3
D.2.1.3 Attribute Default Values ... D-4
D.2.1.4 Preferences Screen Root Element ... D-5
D.2.2 arrays.xml ... D-6
D.2.3 Strings.xml ... D-7
D.3 Converting Preferences for iOS ... D-7
D.4 Converting Preferences for Windows ... D-8

E MAF Sample Applications

E.1 Overview of the MAF Sample Applications ... E-1

Preface

Welcome to the *Developing Mobile Applications with Oracle Mobile Application Framework (OEPE Edition)*.

Audience

This document is intended for developers tasked with creating cross-platform mobile applications that run as natively on the device.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following document:

- *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*
- *Oracle Enterprise Pack for Eclipse User's Guide*
- *Java API Reference for Oracle Mobile Application Framework*
- *JSDoc Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide for MAF Release 2.3.0

The following topics introduce the new and changed features of Oracle Mobile Application Development Framework (Oracle MAF) and other significant changes, which are described in this guide.

New and Changed Features for MAF Release 2.3.0

Oracle MAF Release 2.3.0 includes the following new and changed development features, which are described inthis guide.

- Support for deployment of MAF applications to the Universal Windows Platform (UWP). For more information, see:
  - The What You Need to Develop an Application for the Universal Windows Platform section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.
  - The Setting Up Development Tools for the Universal Windows Platform section in Installing *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.
  - Deploy your application to your local machine or to an installation package that you use to install the application on a supported UWP device. For more information, see Section 27.5, "Deploying a MAF Application to the Universal Windows Platform."
- This release of MAF uses newer versions of Cordova (4.x).The Plugins page of the MAF Application Editor has been enhanced to show the version of Cordova that MAF uses for each platform it supports. For more information, see Chapter 9, "Using Plugins in MAF Applications."
- New version of the `mobileAlta` skin (`mobileAlta-1.5`) that is the default for MAF applications created using this release. For more information, see Chapter 7, "Skinning MAF Applications."
- `RestServiceAdapter`. A new factory class (`RestServiceAdapterFactory`) has been introduced to create REST web service adapters. Use the `RestServiceAdapterFactory.createRestServiceAdapter()` API to create a REST web service adapter. The `RestServiceAdapter` interface has a new package location (`oracle.maf.api.dc.ws.rest`). The functionality that this interface specifies remains unchanged. For more information about creating a REST web service adapter, see Section 15.2, "Creating a Rest Service Adapter to Access Web Services.".

- Data visualization components
  - A new chart (range) that can be rendered using the `areaChart` or `barChart` components. For more information, see Section 12.5.3, "How to Create a Range Chart".
  - For a `geographicMap` component to render an Oracle map when you deploy your application to a device using iOS 9, you must configure an entry in your application's `adf-config.xml` file so that the component uses `HTTPS`. For more information, see Section 12.5.24.1, "Configuring Geographic Map Components With the Map Provider Information".
  - Gauge components have been enhanced in a number of ways in this release. Examples include support for human shapes, vertical orientation for the `ratingGauge` component plus the introduction of a range of other attributes to customize these components. For more information, see the implementations of the gauge components in the `CompGallery` sample application, described in Appendix E, "MAF Sample Applications,", the *Tag Reference for Oracle Mobile Application Framework*, and Section 12.5.22, "How to Create a Rating Gauge".
- Properties in your application's `logging.properties` file can be configured to monitor the performance of your application. MAF also provides a number of APIs to assist with this task. For more information, see Section 30.7, "Measuring MAF Application Performance.".
- MAF applications can send diagnostic and analytics information to Oracle Mobile Cloud Service for analysis by this service.
  - Send diagnostic information (device ID, session ID, and request time) to Mobile Cloud Service by using an adapter (`McsRestServiceAdapter`) that MAF introduced for this purpose. For more information, see Section 30.8, "Sending Diagnostic Information to Oracle Mobile Cloud Service."
  - Send analytics information by configuring the logging.properties file or by using the APIs that MAF introduced for this purpose. For more information, see Section 30.9, "Sending Analytics Information to Oracle Mobile Cloud Service."
- MAF provides a number of new APIs to assist you with authentication-related tasks. For more information, see Section 29.9, "Using MAF Authentication APIs."
- End users can now automatically install digital certificates (client certificates) in a MAF application's keystore to establish two-way SSL sessions for authentication with servers that require client certificates. For information about how to configure your MAF application to enable the automatic install of a client certificate, see Section 29.11, "Configuring a MAF Application to Enable Two-Way SSL for Authentication."
- The `AdfmfContainerUtilities.resetApplication()` API now applies all customizations downloaded from the Configuration Service without restarting the application. For more information, see *Java API Reference for Oracle Mobile Application Framework* and Chapter 14, "Configuring End Points Used in MAF Applications."
- Whitelisting for remote URL application features that access container services

  Application developers who want to implement whitelists in their MAF application can use Cordova plugins, as described in Section 20.3, "Whitelisting Remote URLs in Your MAF Application."

  This release of MAF has removed the design-time support that OEPE provided in previous releases. For example, the Remote URL Whitelist input field has been

removed from the MAF Application editor. Use of the `AdfmfJavaUtilities.addWhiteList` APIs has been deprecated.

Other Significant Changes in this Document for MAF Release 2.3.0

This document has been updated in several ways for this release. Following are the sections that have been added or changed.

- Added information about how you debug the UI code (JavaScript, HTML, and CSS) of MAF applications on Android. For more information, see Section 30.4.2, "How to Debug UI Code on the Android Platform."
- Removed content describing how to use the Mobile-Social authentication server type. This authentication server type was deprecated in a previous release of MAF and is no longer supported.
- Removed content describing how to use SOAP web service data controls. This feature was deprecated in a previous release of MAF and is no longer supported. Customers are recommended to use REST web services with JSON objects. For more information, see Chapter 15, "Using Web Services in a MAF Application."
- Added information describing the languages that MAF supports. For more information, see Section 6.7, "MAF Support of Languages."

1

Introduction to Oracle Mobile Application Framework

This chapter introduces Oracle Mobile Application Framework (MAF), a solution that enables you to create mobile applications that run natively on both iOS and Android phones and tablets.

This chapter includes the following sections:

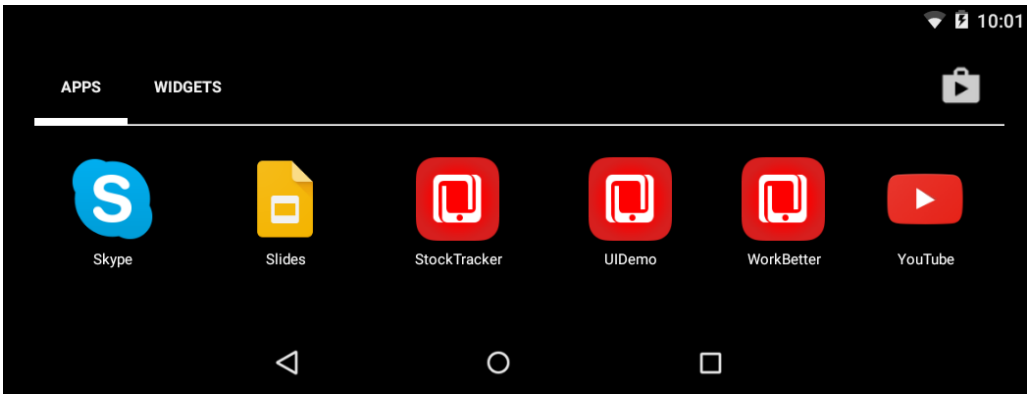
- Section 1.1, "Introduction to the Mobile Application Framework"
- Section 1.2, "About the MAF Runtime Architecture"
- Section 1.3, "About Developing Applications with MAF"
- Section 1.4, "MAF Sample Applications"

1.1 Introduction to the Mobile Application Framework

MAF is a hybrid mobile architecture, one that uses HTML5 and CSS to render the user interface, Java for the application business logic, and Apache Cordova to access device features such as GPS activities and e-mail. Because MAF uses these cross-platform technologies, you can build an application that runs on both Android and iOS devices without having to use any platform-specific tools. After deploying a MAF application to a device, the application behaves similarly to applications that are created using platform-specific tools, such as Objective C or Android SDK. Further, MAF enables you to build the same application for smartphones or for tablets, thereby allowing you to reuse the business logic in the same application and target various types of devices, screen sizes, and capabilities..

AMAF application installs on a user's device like any other application on the device.

Figure 1–1 MAF Applications Installed on a Device



MAF applications consist of one or more application features. An application feature is a reusable, self-contained module of application functionality. Each application feature performs a specific set of tasks, and application features can be grouped together to complement each other's functionality. For example, you can pair an application feature that provides customer contacts together with one for product inventory. Because each application feature has its own class loader and web view (essentially a native UI component that behaves as a browser), features are independent of one another; a single MAF application can be assembled from application features created by several different development teams. Application features can also be reused in other MAF applications. The MAF application itself can be reused as the base for another application, allowing ISVs (independent software vendors) to create applications that can be configured by specific customers.

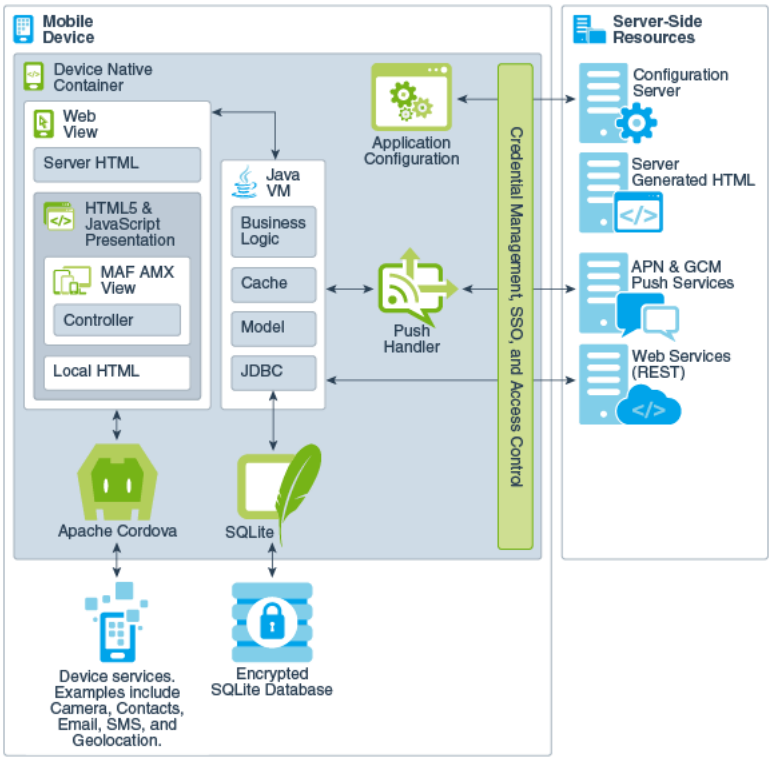
In addition to hybrid mobile applications that run locally on the device, you can implement application features as any of the following mobile application types, depending on the requirements of a mobile application and available resources:

- Mobile web applications—These applications are hosted on a server. Although the code can be portable between platforms, their access to device features and local storage can be limited, as these applications are governed by the device's browser.
- Native applications—These applications are authored using the platform tools (for example, Xcode for iOS) and are therefore limited in terms of serving the platform. Reuse of code is likewise limited.

1.2 About the MAF Runtime Architecture

As illustrated in Figure 1–2, MAF is a thin native container that is deployed to a device. MAF follows the model-view-controller (MVC) development approach, which separates the presentation from the model layer and the controller logic. The native container allows the MAF application to function as a native application on the platform (iOS, Android, or UWP) where you deploy MAF application.

Figure 1–2 The MAF Runtime Architecture



- **Web View**—Uses a mobile web engine of the device to display and process web-based content. In a MAF application, the web view delivers the user interface by rendering the application markup as HTML 5. You can create the user interface for a MAF application feature by implementing any of the following content types. Application features implemented from various content types can coexist within the same MAF application and can also interact with one another.
  - **MAF AMX Views**—Like an application authored in the language specific to the platform of the device, applications whose contents are implemented as MAF Application Mobile XML (AMX) views reside on the device and provide the most authentic device-native user experience. MAF provides a set of code editors that enable you to declaratively create a user interface from components that are tailored to the form factors of mobile devices. You can use these components to create the page layout, such as List View, as well as input components, such as Input Text. When you develop MAF AMX views, you can leverage data controls. These components enable you to declaratively create data-bound user interface components, access a web service, and the services of a mobile device (such as camera, GPS, or e-mail). At runtime, the JavaScript engine in the web view renders MAF AMX view definitions into HTML5 and JavaScript. For more information, see the following:
    * Chapter 11, "Creating MAF AMX Pages"
    * Chapter 12, "Creating the MAF AMX User Interface"
    * Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX"
    * Chapter 15, "Using Web Services in a MAF Application"
    * Chapter 14, "Configuring End Points Used in MAF Applications"
    * Chapter 19, "Using the Local Database in MAF AMX"

* Chapter 18, "Creating Custom MAF AMX UI Components"

Task Flow—The Controller governs the flow between pages in the MAF application, enabling you to break the application flow into smaller, reusable task flows and include non-visual components, such as method calls and decision points. For more information, see Section 11.2, "Creating Task Flows."

– **Server HTML**— With this content type, the user interface is delivered from server-generated web pages that can open within the web view of the application feature. Within the context of MAF, this content type is referred to as remote URL. The resources for these browser-based pages do not reside on the device. Instead, the user interface, page flow logic, and business logic are delivered from a remote server. When one of these remotely hosted web pages is allowed to open within the web view, it can use the Cordova JavaScript APIs to access any designated device-native feature or service, such as the camera or GPS capabilities. When implementing a feature using the remote URL content, you can leverage an existing browser-based application that has been optimized for mobile use, or use one that has been written specifically for a specific type of mobile device. For applications that can run within a browser on either desktops or tablets, you can implement the remote URL content using applications created through Oracle ADF Faces rich client-based components. For more information, see Chapter 20, "Implementing Application Feature Content Using Remote URLs."

Note: Because the content is served remotely, a feature that uses a remote URL is available only as long as the server connection remains active.

– **Local HTML**—HTML pages that run on the device as a part of the MAF application. Local HTML files can access device-native features services through the Cordova and JavaScript APIs.

- **Cordova**—The Apache Cordova JavaScript APIs that integrate the device's native features and services into a MAF application. Although you can access these APIs programmatically from Java code (or using JavaScript when implementing a MAF application as local HTML), you can add device integration declaratively when you create MAF AMX pages because MAF packages these APIs as data controls.
- **Java Virtual Machine**—Provides a Java runtime environment for a MAF application. This Java Virtual Machine (JVM) is implemented in device-native code, and is embedded (or compiled) into each instance of the MAF application as part of the native application binary. The JVM is based on the JavaME Connected Device Configuration (CDC) specification.
  - **Business Logic**—Business logic in MAF application may be written in Java. Managed Beans are Java classes that can be created to extend the capabilities of MAF, such as providing additional business logic for processing data returned from the server. Managed beans are executed by the embedded Java support, and conform to the JavaME CDC specifications. For more information, see Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX."
  - **Model**—Contains the binding layer that connects the business logic components with the user interface. In addition, the binding layer provides the execution logic to invoke REST web services. For more information, see Section 1.3.1, "About Connected and Disconnected

Applications."

  - **JDBC**— The JDBC API enables access to the data in the encrypted SQLite database through CRUD (Create, Read, Update and Delete) operations.

- **Application Configuration** refers to services that allow application configurations to be downloaded and refreshed, such as URL endpoints for a web service or a remote URL connection. Application configuration services download the configuration information from a WebDav-based server-side service. For more information, see Chapter 14, "Configuring End Points Used in MAF Applications."

- **Credential Management, Single Sign-on (SSO), and Access Control**—MAF handles user authentication and credential management using the mobile security SDK from Oracle Identity Management. MAF applications perform offline authentication, meaning that when users log in to the application while connected, MAF maintains the user name and password locally on the device, allowing users to continue access to the application even if the connection to the authentication server becomes unavailable. MAF encrypts the locally stored user information as well as the data stored in the local SQLite database. After authenticating against the login server, a user can access all of the application features secured by that connection. MAF also supports the concept of access control by restricting access to application features (or specific functions of application features) by applying user roles and privileges.

- **Push Handler**—Enables the MAF application to receive events from the iOS or Android notification servers. The Java layer handles the notification processing.

Resources that interact with the native container include:

- **Encrypted SQLite Database**—The embedded SQLite database is a lightweight, cross-platform relational database that protects locally stored data and is called using JDBC. Because this database is encrypted, it secures data if the device is lost or stolen. Only users who enter the correct user name and password can access the data in the local database. For more information, see Chapter 19, "Using the Local Database in MAF AMX."

- **Device Services**—The services and features that are native to the device and integrated into application features through the Cordova APIs.

The device native container enables access to the following server-side resources:

- **Configuration Server** —A WebDav-based server that hosts configuration files used by the application configuration services. The configuration server is delivered as a reference implementation. Any common WebDav services hosted on a J2EE server can be used for this purpose. For more information, see Chapter 14, "Configuring End Points Used in MAF Applications."

- **Server-Generated HTML**—Web content hosted on remote servers used for browser-based application features. For more information, see Chapter 20, "Implementing Application Feature Content Using Remote URLs."

- **APNs and GCM Push Services**—Apple Push Notification Service (APNs) and Google Cloud Messaging (GCM) are the notification providers that send notification events to MAF applications. Push notifications are not supported on MAF applications that you deploy to the Universal Windows Platform.

- **REST Services**—Remotely hosted REST-based web services, which can be accessed through the Java layer or through data controls. For more information,

see Chapter 15, "Using Web Services in a MAF Application" and Chapter 16, "Working with REST Services."

1.3 About Developing Applications with MAF

Although the components of a MAF application may be created by a single developer, an application may typically be built from resources provided by different development roles. An application *developer* builds the application data and the user interface logic either as an application or as a reusable program that can be used in an application feature. An application *assembler* gathers different application features into a single application and puts them in a user-friendly, navigable order. An application *deployer* ensures a controlled application deployment. For example, deployment of MAF applications may require certificates and uploads to public vendor sites such as the Apple App Store or GooglePlay.

Note: Depending on the application development team size and your organization, one person may fill many different roles.

Typically, you perform the following activities when building a MAF application:

- Gathering requirements
- Designing
- Developing
- Deploying
- Testing and debugging
- Securing
- Enabling access to the server-side data
- Redeploying
- Retesting and debugging
- Publishing

The steps you take to build a MAF application may be similar to the following:

1. **Gathering requirements**: Create a mobile use case (or user scenario) by gathering user data that describes who the users are, their essential tasks, and the location or context in which they perform them. Consider such factors as the type of information required to complete a task, the information that is available to the user, and how it is accessed or delivered.
2. **Designing**: After you construct a use case, create a wireframe that illustrates all of the steps (and associated user views) in the application's task flow. When creating a task flow, consider how, and when, different users may interact. Does viewing data (such as a push notification) suffice to complete a task? If not, how much data entry does the task require? To frame these tasks within a mobile context, compare completing tasks using a desktop application to a mobile application. A single desktop application may enable multiple functions that might be partitioned into several different mobile applications (or in the context of MAF, several different application features embedded in a MAF application). Because mobile applications are generally used in short bursts (about two minutes at a time), they must be easily navigable and accommodate the limited data entry of a mobile device.

During the design and development phases, keep in mind that mobile applications may require a set of mobile-specific server-side resources, because the applications may not be able to consume large amounts of data delivered through complex web services. In addition, a mobile application may require extensive client side logic to process data returned by services. It's usually best to shape the data coming into a mobile application on the server side to avoid forcing the client to process too much data.

3. **Developing**: Select the technology that is best suited for application. While the MAF web view supports remote content which may be authored using Apache Trinidad or ADF Faces Rich Client components, these applications do not support offline use. Applications authored in MAF AMX, which runs on the client, however, integrate with device services, enabling end users to not only view files and utilize GPS services, but also collaborate with one another by tapping a phone number to call or text. The MAF AMX component set includes data visualization tools (DVT) that enable you to add analytics that render appropriately on mobile screens. A MAF AMX application supports offline use by transferring data from remote source and storing it locally, enabling end users to view information when they are not connected.

   MAF provides a set of wizards and editors that build not only the basic application itself, but also the application features that are implemented from MAF AMX and local HTML content. Using these tools provides such artifacts as descriptor files for configuring the MAF application and incorporating its application features, a set of default images for application icons, springboards, navigation bar items that are appropriate to the form-factors of the supported platforms.

   For more information, see the following:

   - Chapter 2, "Getting Started with MAF Application Development"
   - Chapter 3, "Configuring the Content of a MAF Application"
   - Chapter 11, "Creating MAF AMX Pages"
   - Chapter 12, "Creating the MAF AMX User Interface"

4. **Deploying**: You deploy the MAF application not only in the context of publishing it to end users, but also for testing and debugging, because MAF applications cannot run until they have been deployed to a device or simulator. Depending on the phase of development, you designate the credential signing options (debug or release). For testing, you deploy the application to a mobile device or simulator. For production, you package it for distribution to application markets such as the Apple App Store or Google Play.

   To deploy an application you first create a deployment profile that describes the target platform and its devices and simulators. Creating a deployment profile includes selecting the splash screen and launch icons used for the application in different orientations (landscape or portrait) and on different devices (phone or tablets). For more information, see Chapter 27, "Deploying MAF Applications."

5. **Testing and debugging**: During the testing and debugging stage, you optimize the application by deploying it in debug mode to various simulators and devices and then review the debugging output provided through OEPE and platform-specific tools. For more information, see Chapter 30, "Testing and Debugging MAF Applications."

6. **Securing**: Evaluate security risks throughout the application development process. While mobile applications have unique security concerns, they share the same

vulnerabilities as any application that accesses remotely served data. To ensure client-side security, MAF provides such features as:

- APIs that generate a strong password to secure access to the SQLite database and encrypt and decrypt its data.
- A set of web service policies that support SSL.
- A `cacerts` file of trusted Certificate Authorities to enforce deployment in SSL

MAF's security configuration includes selecting a login server, such as the Oracle Access Mobile and Social server, or any web page protected by the basic HTTP authentication mechanism, configuring the session management (session and idle timeouts) and also setting the endpoint to the access control service web service, which hosts the application's user roles. For more information, see Chapter 29, "Securing MAF Applications."

7. **Enabling access to the server-side data**: After ensuring that your application functions as expected at a basic level, you can implement the Java code or use data controls to access the server-side data. For more information, see Section 1.3.1, "About Connected and Disconnected Applications."
8. **Redeploying**: During subsequent rounds of deployment, ensure that after adding security to your application and enabling access to the server-side data, the application deployment runs as expected and the application is ready for the final testing and debugging.
9. **Retesting and debugging**: During the final round of testing and debugging, focus on the security and the server-side data access functionality, ensuring that their integration into the application did not result in errors and unexpected behavior.
10. **Publishing**: Deploying the application to the production environment typically involves publishing to an enterprise server, the Apple App Store, or Google Play. After you publish the MAF application, end users can download it to their mobile devices and access it by touching the designated icon. The application features bear the designated display icons and display as appropriate to the end user and the user's device.

1.3.1 About Connected and Disconnected Applications

A MAF application can run while connected to a network, but can also work in a disconnected mode, such as when there is no cellular signal. Examples include:

- A basic connected application that includes a user interface backed directly by a web service data control that, in turn, invokes a web service hosted on a server.
- A connected application that uses moderate (or complex) data services. For this type of application, Java classes (POJOs) exposed through data controls can dispatch data queries between the user interface and the service data source.
- A disconnected application that manipulates data stored in the SQLite database, enabling application users to work offline. The application may need to get data from a web service, but if connectivity is lost, the data is stored locally and synchronized when connectivity is restored.

1.4 MAF Sample Applications

MAF provides an extensive set of sample applications that implement a range of use cases. You can open these sample applications in OEPE to explore the source code and/or deploy to a device or emulator/simulator to view the runtime behavior.

Sample applications exist that demonstrate how you can implement a variety of functions in a MAF application, such as accessing device-native features, performing operations on a local database or implementing gestures, amongst other things.

A `HelloWorld` sample application demonstrates how to implement a single application feature with a local HTML file. We suggest that you use the `HelloWorld` application to verify that your development environment is set up correctly to compile and deploy an application.

Other sample applications that may be of interest to you when getting started include the:

- `CompGallery` (component gallery) that serves as an introduction to the MAF AMX UI components by demonstrating all of these components. Using this application on a device or emulator/simulator, you can change the attributes of components and see the effects of these changes in real time.
- `WorkBetter` sample application, illustrated in Figure 1–3, showcases the MAF AMX UI capabilities. It also demonstrates how you can programmatically access REST services.

After setting up your development environment (see *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*, you can open the MAF sample applications. In most cases, the name of the application's directory provides a good indicator as to its purpose. For example, the application in the `SkinningDemo` directory demonstrates how you can change the skin of the MAF application.

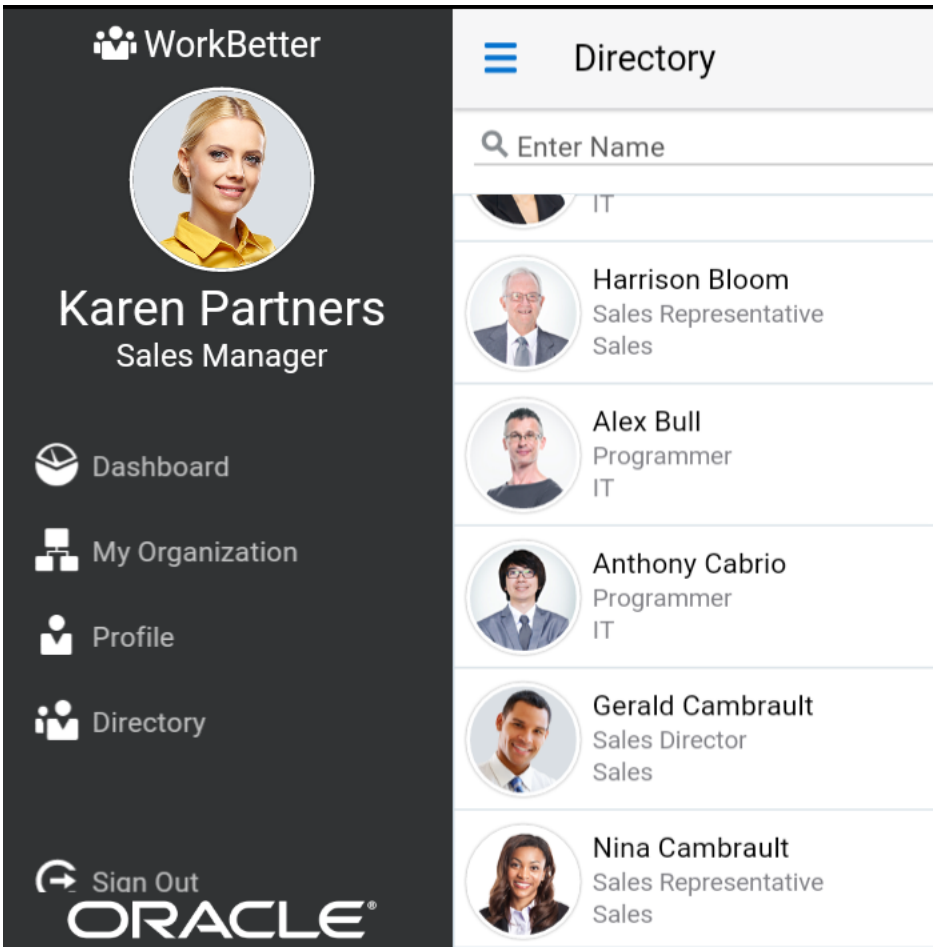
You can see the list of MAF Examples, and choose one or more to open as follows:

1. In OEPE, choose the Oracle MAF perspective.
2. Open the MAF Examples dialog by selecting **File > New > MAF Example**.
3. To see a description of a sample application, highlight the sample application.

   To open a sample application, select the checkbox and click **Finish**.

For more information, see Appendix E, "MAF Sample Applications."

Figure 1–3 The WorkBetter Sample Application



2

Getting Started with MAF Application Development

This chapter describes how to create a MAF application in OEPE and introduces the files and other artifacts that OEPE generates when you create the application.

This chapter includes the following sections:

- Section 2.1, "Introduction to the MAF Development Environment""
- Section 2.3, "Creating a MAF Application"
- Section 2.4, "Defining Application Features for a MAF Application"
- Section 2.5, "Adding Content to an Application Feature"
- Section 2.6, "Adding Application Features to a MAF Application"
- Section 2.8, "Containerizing a MAF Application for Enterprise Distribution"
- Section 2.8, "Containerizing a MAF Application for Enterprise Distribution"

2.1 Introduction to the MAF Development Environment

The Oracle Mobile Application Framework (MAF) extension in OEPE provides a number of editors and wizards to facilitate the development, testing, and deployment of MAF applications. Using these wizards, you can create a MAF application, define one or more application features, add content to an application feature, and deploy the MAF application to a test environment or device in a relatively short amount of time.

Figure 2–1 shows the WorkBetter sample application in OEPE's MAF Application Editor where a number of the items that you use to develop MAF applications are identified:

1. The MAF Application Editor (invoked from the assembly project), used to specify the MAF application's name, the default navigation menus (navigation bar or springboard) that the application renders, security, and device access options for the application.
2. The MAF Features Editor (invoked from the view project), where you define the application features that your MAF application contains.
3. By default, OEPE creates a MAF application with two data controls, one in the application project *`application`*`Application` and one in the view project *`application`*`View`. These data controls expose operations that you can drag to a MAF AMX page where OEPE displays context menus to complete configuration of the operation when you drop it on the page. For example, dragging the `hideNavigationBar()` operation to a page prompts OEPE to display a context

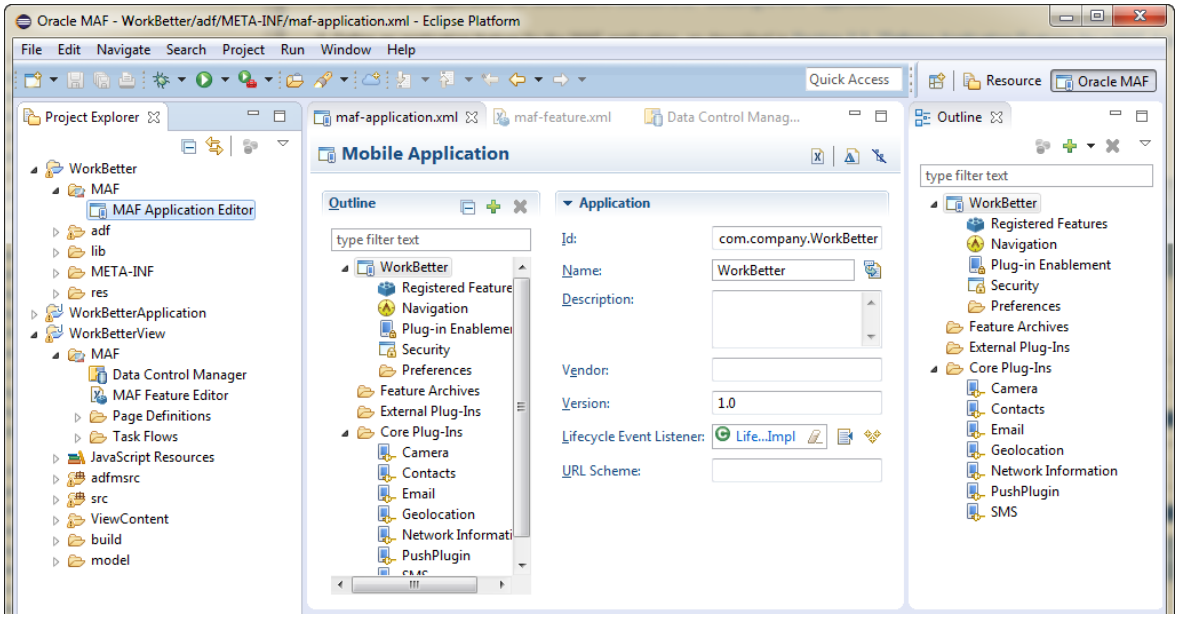
menu where you configure a control for end users to hide an application's navigation bar.

The WorkBetter sample application is one of a number of sample applications that MAF provides to demonstrate how to create mobile applications using MAF. For more information, see Section 1.4, "MAF Sample Applications."

OEPE proposes default options in the wizards so that you can create a MAF application with one application feature displaying one MAF AMX page as follows:

1. Create a MAF application, as described in Section 2.3, "Creating a MAF Application."
2. Define an application feature for the MAF application and add content, as described in Section 2.4, "Defining Application Features for a MAF Application."
3. Add content to the application feature, as described in Section 2.5, "Adding Content to an Application Feature."

Figure 2–1 MAF Application Editor



OEPE's MAF Feature Editor and MAF Application Editor are the primary tools you use to interact with a MAF application while you are creating it. These two editors interact with each other, and with the files that comprise the application you are developing, in a way that can affect your choices while editing.

As shown in Figure 2–1, the MAF Application Editor is located in the Project Explorer under the application and MAF node. It edits the `maf-application.xml` file.

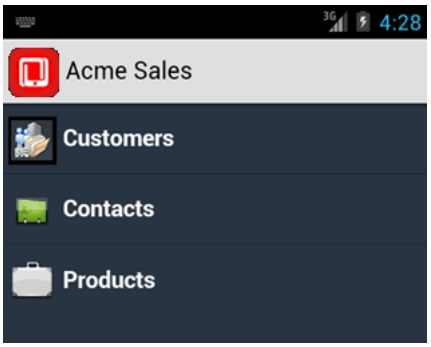
The MAF Feature Editor is located in Explorer under the view project and MAF node. It edits the `maf-feature.xml` file.

2.1.1 About the MAF Application Editor

The MAF Application Editor enables you to configure the `maf-application.xml` file to set the basic configuration of the mobile application by designating its display name, a unique application ID (to prevent naming collisions) and also by selecting the application features that will display on the application springboard (the equivalent of a home page on a smartphone). Further, this editor enables you to create the user

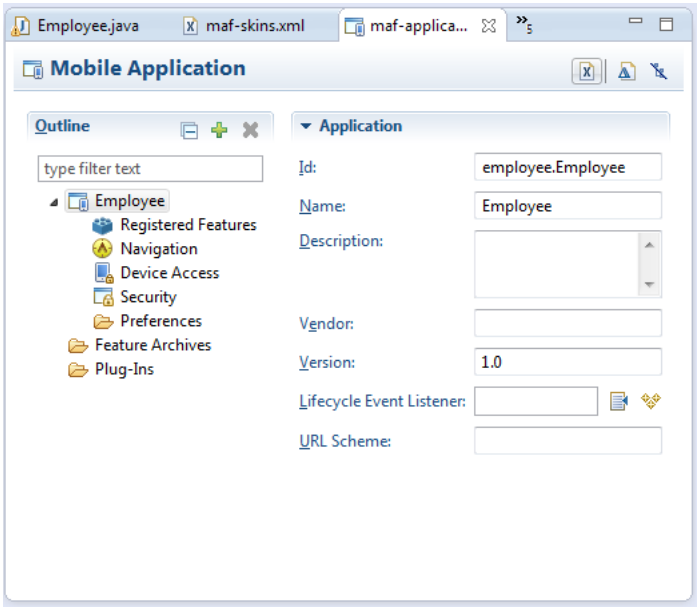
preferences pages for the mobile application. The editor is described in Section 2.3, "Creating a MAF Application,".

Figure 2–2 Application Features Displaying in the Mobile Application's Springboard



You can modify these elements declaratively using the MAF Application Editor, shown in Figure 2–3, or manually using the XML Editor or the Source editor. You can use these approaches interchangeably.

Figure 2–3 The MAF Application Editor for the maf-application.xml File



2.1.2 About the MAF Feature Editor

The MAF Feature Editor enables you to add features to your application. You create the features with this editor, and specify the characteristics (ID, name, description, vendor information, version, lifecycle event listener, security, and images for navigation and for the springboard) through the editor.

2.2 Using the Oracle MAF Perspective

In Eclipse, a perspective is a collection of views arranged in a specific layout that is suitable for a type of development. MAF has its own perspective which you should

use when you are developing MAF applications. It gives you access to menu and toolbar options that we describe elsewhere in this manual.

To change to the Oracle MAF perspective:

1. In the IDE, click **Window** > **Open Perspective** > **Other**.

   Alternatively, click .

2. In the Open Perspectives dialog, choose Oracle MAF. Click **OK**.

2.3 Creating a MAF Application

You create a MAF application using the creation wizards in OEPE.

2.3.1 How to Create a MAF Application

You create a MAF application in OEPE using the application creation wizard.

To create a MAF application:

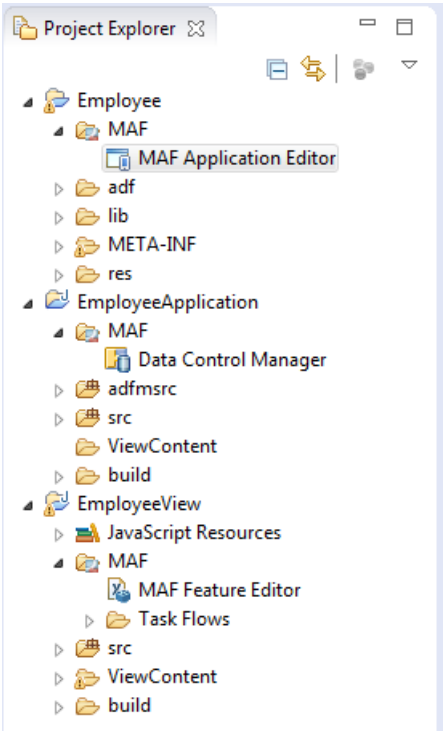
1. In the main menu, choose **File** > **New**, and then select **MAF Application**.
2. In the MAF Application wizard, enter application details like name and deployment targets.
3. Click **Finish**.

2.3.2 What Happens When You Create an MAF Application

OEPE creates a MAF application with three projects and two data controls (for application features and device features), as shown in Figure 2–4. It also creates files that you use to configure your MAF application and files that your MAF application needs when you deploy it to the Android and/or iOS platform(s).

For more information about the files and artifacts that OEPE generates when you create a MAF application, see Appendix C, "MAF Application and Project Files.".

Figure 2–4 Generated MAF Application Artifacts



Note: If you have the Error Log open when you create a MAF application you will see messages similar to:

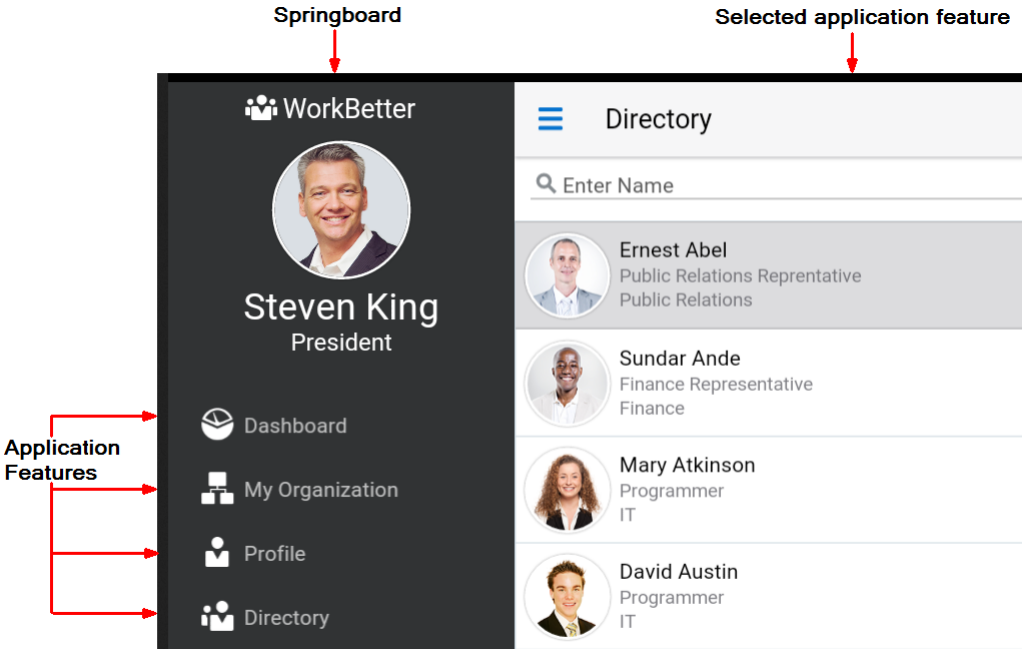
```
org.eclipse.core.runtime.CoreException: Illegal install location
D:\p4\depots\OEPE\tools-eclipse\annex\maf-2.1\install for vmInstall
oracle.eclipse.tools.maf.JVMCDCv201 contributed by
oracle.eclipse.tools.maf.dt.v201: Associated MAF runtime is not
installed.
```

There will be one entry for each release of MAF with OEPE. You can safely ignore them.

2.4 Defining Application Features for a MAF Application

A MAF application must have at least one application feature. The WorkBetter sample application, for example, includes four application features (Dashboard, People, Organizations, and Springboard). Figure 2–5 shows three of these application features displaying in that application's custom springboard.

Figure 2–5 Application Features in the WorkBetter Application's Springboard



2.4.1 How to Define an Application Feature

You define an application feature for a MAF application using the MAF Application Editor.

To define an application feature for a MAF application:

1. In the Project Explorer, expand the view project, *applicationView* and **MAF** and double-click **MAF Feature Editor**.
2. In the Mobile Features page, click the **Add** icon.
3. In the New Object dialog, select Feature and

   Complete entries in the New Object dialog as follows:

   - **Type of new object**: `Feature`.
   - **Feature ID**: Enter a unique ID for the application feature or accept the value that OEPE generates.
4. Click **OK**.

2.5 Adding Content to an Application Feature

One of the tasks to do after you define an application feature is to add content to the application feature. In the Outline section of the editor, expand the new node for the feature and click the **Add** icon. Choose the type of new feature and complete the information required:

- **MAF AMX Page**: Choose this content type if you want the application feature to render MAF AMX pages.
- **MAF Task Flow**: Choose this content type if you want the application feature to render a collection of activities that make up a task flow. Examples of activities that you can include in a task flow are views (to display MAF AMX pages),

method calls (to invoke managed bean methods), and task flow calls (to call other task flows).

- **Local HTML**: Choose if you want the application feature to render HTML page.
- **Remote URL**: Choose if you want the application feature to render content from a remote URL.

The general steps to add a content type to an application feature are the same for all content types. That is, you choose the type of content to add to the application feature in the Content tab of the Features page of the maf-features.xml file's overview editor. For the specific steps for each content type, see Chapter 5, "Defining the Content Type of MAF Application Features."

2.6 Adding Application Features to a MAF Application

Application features are automatically added to the MAF application when you create them using the MAF Feature Editor.

2.6.1 How to Add an Application Feature to a MAF Application

To work with the features in the application:

1. In the Project Explorer, expand the assembly project, *application* and **MAF** and double-click **MAF Application Editor**.
2. In the outline, select **Registered Features**.
3. In the Registered Features pane you can see the features associated with the application.

Select a feature to define how it works in the application. For example, you can specify that the feature is available on the navigation bar, or on the springboard.

2.6.2 What You May Need to Know About Feature Reference IDs and Feature IDs

OEPE writes an entry in the `maf-application.xml` file to reference the application feature that you add to the MAF application.

In the `maf-application.xml` file, the `refId` attribute of an `<adfmf:featureReference>` element identifies the corresponding application feature in the `maf-feature.xml` file. For this reason, the value of the `refId` attribute for a `<adfmf:featureReference>` element in the `maf-application.xml` file must match the value of the id attribute defined for the `<adfmf:feature>` element in the `maf-feature.xml file`.

Use a naming convention consistently to make sure that application feature IDs are unique. Application feature IDs must be unique across a MAF application.

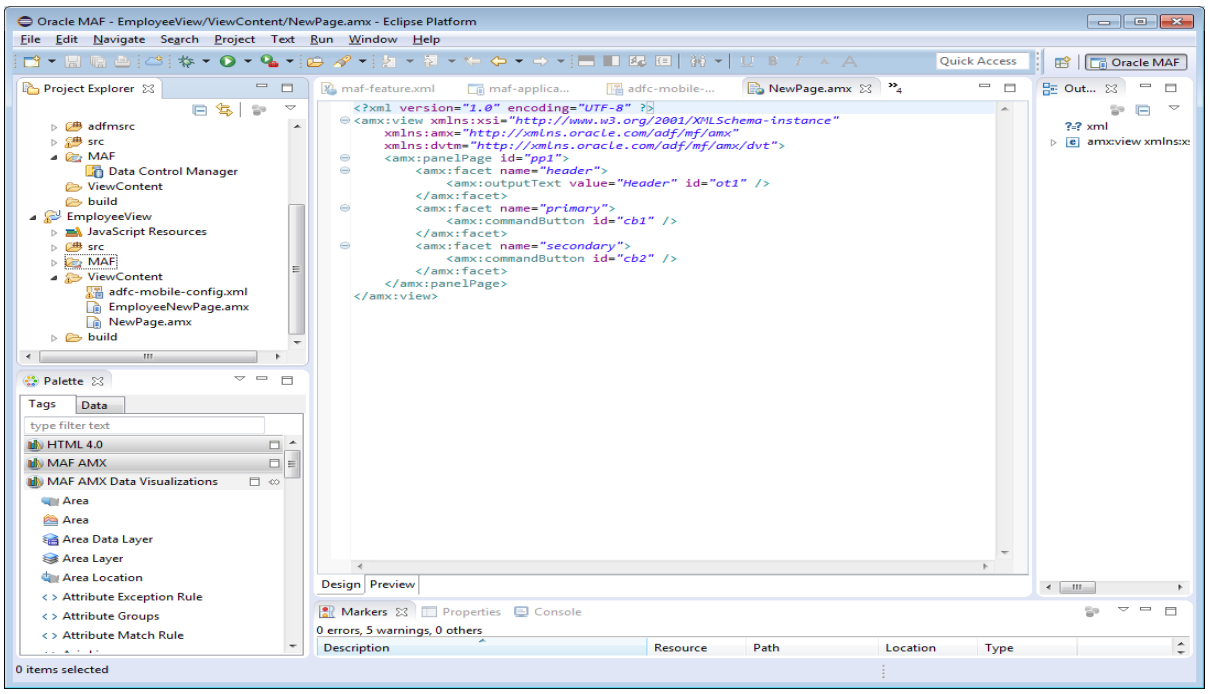
The example below shows the entries for the People application feature in the WorkBetter sample application's `maf-application.xml` and `maf-feature.xml` files.

```
<!-- Feature Reference ID in maf-application.xml File -->
<adfmf:featureReference id="fr2" refId="People"/>
...
<!-- Feature ID in maf-feature.xml File -->
<adfmf:feature id="People" name="People" icon="images/people.png"
image="images/people.png">
...
```

2.7 Creating MAF AMX Pages and MAF Task Flows

As described in Chapter 11, "Creating MAF AMX Pages" the MAF AMX components enable you to build pages that run identically to those authored in a platform-specific language. MAF AMX pages enable you to declaratively create the user interface using a rich set of components. Figure 2–6 illustrates the declarative development of an MAF AMX page, which involves selecting options in the Palette and adding them to the MAF AMX page.

Figure 2–6 Creating an MAF AMX Page



These pages may be created by the application assembler, who creates the MAF application and embeds application features within it, or they can be constructed by another developer and then incorporated into the MAF application either as an application feature or as a resource to an MAF application.

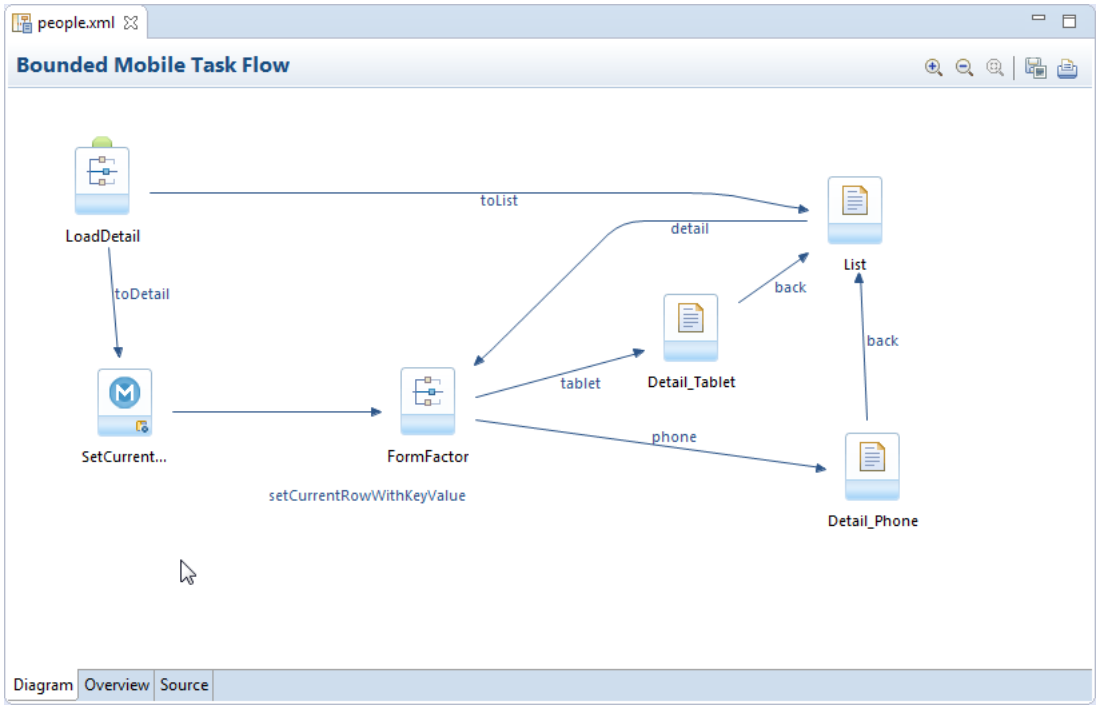
The project in which you create the MAF AMX page determines if the page is used to deliver the user interface content for a single application feature, or be used as a resource to the entire MAF application. For example, a page created within the application controller project, as shown in Figure 2–7, would be used as an application-wide resource.

> **Tip:** To make pages easier to maintain, you can break it down in to reusable segments known as page fragments. An MAF AMX page may be comprised one or more page fragments.

MAF enables you to arrange MAF AMX view pages and other activities into an appropriate sequence through the MAF task flow. As described in Figure 11.2 an MAF task flow is visual representation of the flow of the application. It can be comprised of MAF AMX-authored user interface pages (illustrated by such view activities, such as the WorkBetter sample application's default *List* page and the *Detail* page in Figure 2–8) and nonvisual activities that can call methods on managed beans. The non-visual elements of a task flow can be used to evaluate an EL expression or call another task flow. As illustrated by Figure 2–8, MAF enables you to declaratively

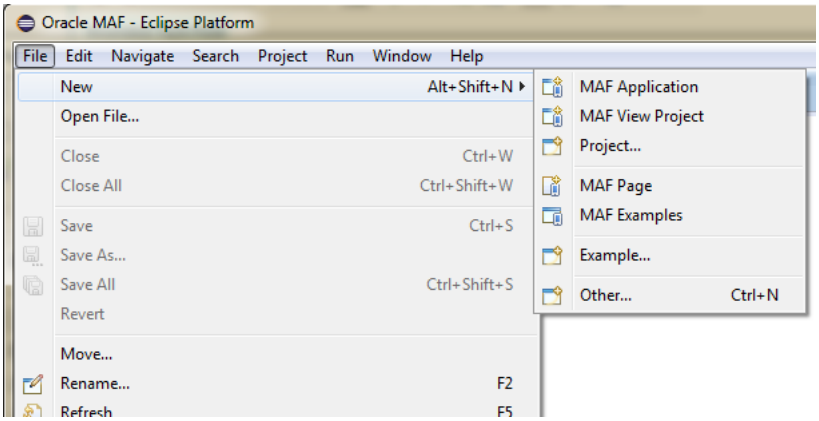
create the task flow by dragging task flow components onto a diagrammer. MAF provides two types of task flows: a bounded task flow, which has a single point of entry, such as the *List* page in the WorkBetter sample application, and an unbounded task flow, which may have multiple points of entry into the application flow. The WorkBetter sample application sample application is This sample application is available from **File** > **New** > **MAF Examples**.

Figure 2–7 MAF Task Flow



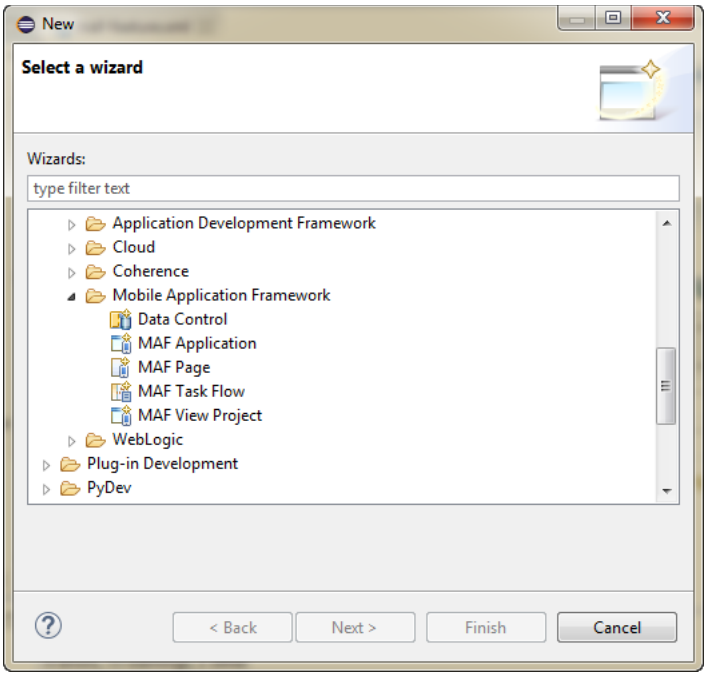
MAF provides a number of dialogs and wizards to add MAF pages, reusable portions of MAF AMX pages called MAF page fragments, and application features. Figure 2–8 shows the menu options available from the **File** menu.

Figure 2–8 Options for Creating Resources for Application Features



Additional options are available from **File** > **New** > **Other** (see Figure 2–9). In the New dialog, expand **Oracle** then expand **Mobile Application Framework**

Figure 2–9 Wizards for Creating Resources for Application Features



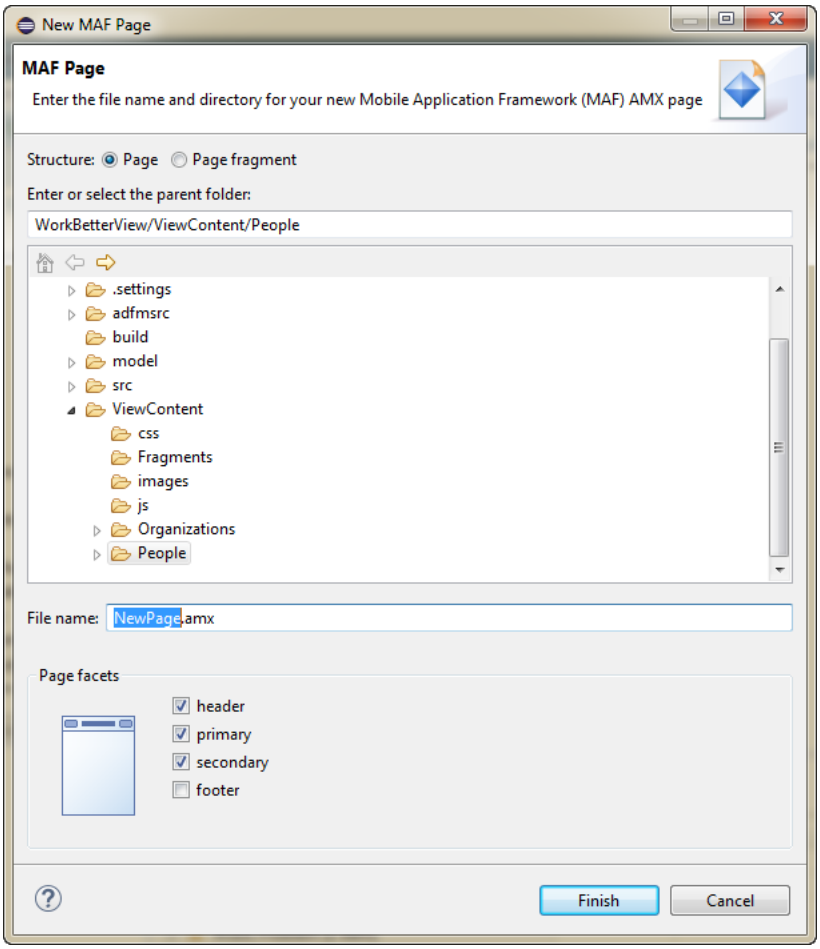
2.7.1 How to Create an MAF AMX Page

You can use the MAF Page dialog to create AMX pages used for the user interface for an application feature, or as an application-level resource (such as a login page) that can be shared by the application features that comprise the MAF application. For more information on application feature content, see Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML."

To create an MAF AMX page as Content for an Application Feature:

1. Choose **File** > **New** and then **MAF Page**.
2. Complete the New MAF Page dialog, shown in Figure 2–10, by choosing the parent folder, for example `ViewContent` and entering a name in the **File Name** field.

Figure 2–10 Creating an MAF AMX Page in an Application Controller Project



3. Select (or deselect) the Page Facets that are used to create a primary and secondary header and footer. Click **OK**.

   For more information, see Section 12.2.2, "How to Use a Panel Page Component."

4. Click **Finish**. For more information using the AMX components, see Section 11.3.1.2, "Creating MAF AMX Pages." See also Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML."

To create an MAF AMX page as a Resource to an MAF application:

1. Choose **File** > **New** and then **MAF Page**.
2. Complete the Create MAF AMX Page dialog, shown in Figure 2–10, by entering a name in the **File Name** field. In the **Directory** field, enter the file location, which must be within the `public_html` folder of the application controller project. Click **OK**.
3. Build the MAF AMX page. For more information, see Section 11.3.1.2, "Creating MAF AMX Pages."

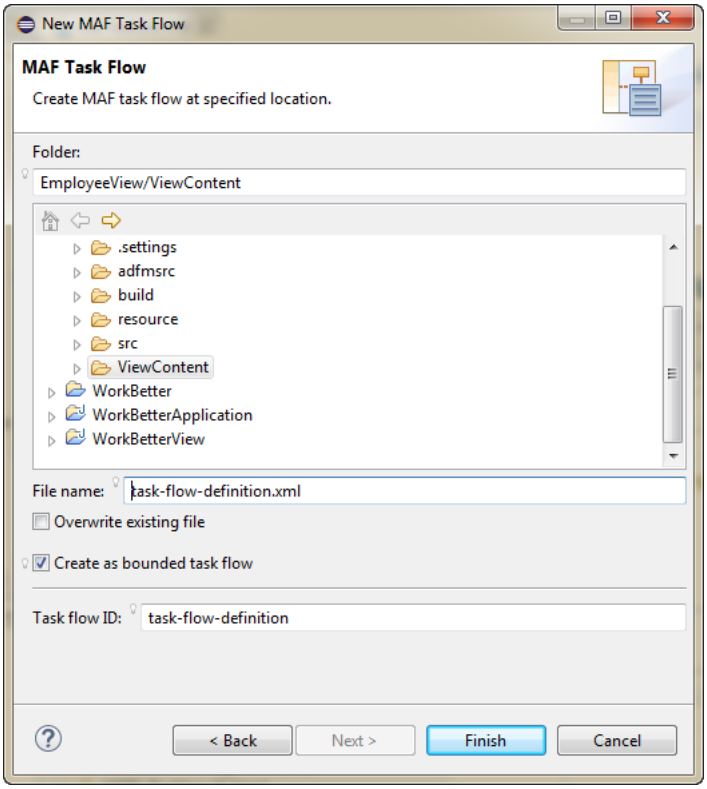
2.7.2 How to Create MAF Task Flows

You can deliver the content for an application feature as an MAF task flow.

To create an MAF Task Flow as content for an application feature:

1. Choose **File** > **New** > **Other**.
2. In the New dialog, expand **Oracle** then expand **Mobile Application Framework**. Select **MAF Task Flow** and click **Next**.
3. Complete the New MAF Task Flow dialog, shown in Figure 2–11, by entering a name in **File Name**. Click **OK**.

Figure 2–11 Creating an MAF Task Flow in a View Controller Project



4. Click **Finish** to build the task flow. See also Section 11–2, "Task Flow File."

2.7.3 What Happens When You Create MAF AMX Pages and Task Flows

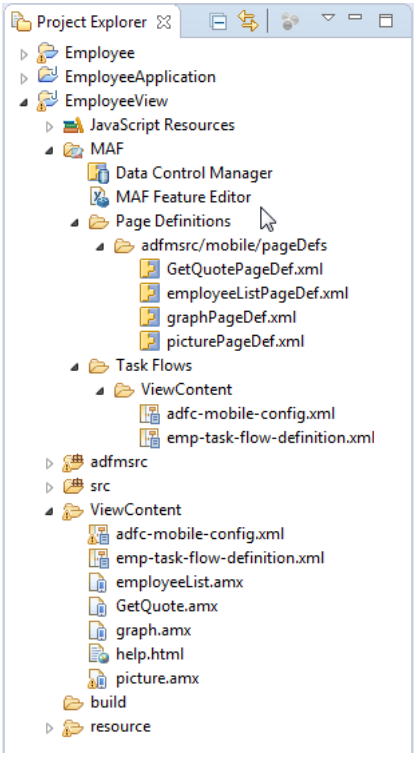
OEPE places the MAF AMX pages and task flows in the **ViewContent** node of the view project, as shown by `employeeList.amx` and `emp-task-flow-definition.xml` in Figure 2–12. These artifacts are referenced in the `maf-feature.xml` file.

The task flows are also listed under the view project MAF node.

Other resources, such as customized application splash screen (or launch) images and navigation bar images, are also housed in the **ViewContent** node.

To manage the unbound task flows, OEPE generates the `adfc-mobile-config.xml` file. Using this file, you can declaratively create or update a task flow by adding the various task flow components, such as a view (a user interface page), the control rules that define the transitions between various activities, and the managed beans to manage the rendering logic of the task flow.

Figure 2–12 MAF AMX Pages and Task Flows within the View Project



OEPE places the MAF AMX page and task flow as application resources to the mobile application in the **ViewContent** node of the application controller project. As illustrated in Figure 2–12, the file for the MAF AMX page is called `application_resource.amx` and the task flow file is called `ApplicationController-task-flow.xml` (the default name).

2.8 Containerizing a MAF Application for Enterprise Distribution

At the time of deployment, you can choose to wrap the MAF application with the OMSS (Oracle Mobile Security Suite) to take advantage of its enterprise mobile application management capabilities. OMSS allows secure access to corporate apps and data from mobile devices while preserving a rich user experience. Its Mobile Security Container creates the enterprise Workspace on any iOS or Android device, corporate-owned or personal. Employees get seamless access to corporate data and apps with enterprise-grade security and single sign-on authentication.

With the Mobile Security App Containerization Tool, you can add a standardized security layer to native mobile apps. The containerization process is simple and injects the following security services into your app.

- Secure data transport: An encrypted AppTunnel through Mobile Security Access Server to application back-end resources behind an enterprise firewall.
- Authentication: Managed by the Secure Workspace app and provides single sign-on across apps in the secure workspace.
- Secure data storage: Encrypted storage of app data, including files, database, app cache and user preferences.

- Data leakage controls: The ability to restrict file sharing and copy paste to only other trusted apps. This enables you to control the sharing of data, including e-mail, messaging, printing and saving.
- Dynamic policy engine: More than 50 detailed app controls, including authentication frequency, geo and time fencing as well as remote lock and wipe.

The OMSS single sign-on authentication and user identity propagation to MAF app services is supported only for applications configured to use the Web SSO authentication server type for login connections. Applications using HTTP Basic or OAuth authentication will be required to log in to the MAF app after the Container authentication is successful. For details about these authentication types and the role they play in OMSS, see Section 29.5.11, "What You May Need to Know About Login Connections and Containerized MAF Applications."

The containerization process is simple and does not change the way you develop the MAF application. In fact, you should not change application code specifically with containerization in mind. You develop the MAF application the same way whether or not you intend to deploy with OMSS containerization enabled.

When you deploy the application with OMSS containerization enabled, OEPE runs the Mobile Security App Containerization Tool provided by OMSS to containerize the MAF application.

After deployment, the MAF app developer works with the OMSS system administrator to get the app added to OMSS Mobile App Catalog and to configure appropriate policies. For details, see the "Managing Mobile Apps" chapter in *Administering Oracle Mobile Security Suite*.

When the user launches the containerized MAF application, the Secure Workspace app redirects to the Mobile Security Container, which performs SSO authentication before handing the session back to the MAF application. The containerized MAF application does not require VPN to connect to internal websites or services. Instead a secure AppTunnel is established between the application and Mobile Security Access Server (MSAS), which provides secure transport for accessing internal sites and services that have been registered for access by mobile device users.

For more information about how OMSS containerization affects MAF applications, see these sections:

- For the OEPE procedure to containerize the MAF application for OMSS, see Section 27.9, "Deploying with Oracle Mobile Security Suite."
- For details about accessing secure web services behind the corporate firewall by the containerized MAF application, see Section 15.3.3, "What You May Need to Know About Accessing Web Services and Containerized MAF Applications."
- For details about the authentication process of containerized MAF applications, see Section 29.4, "Overview of the Authentication Process for Containerized MAF Applications."

In the OMSS documentation library, refer to the following list of resources for details about OMSS administration tasks related to mobile devices and workspace containers.

- For background information about OMSS, see the "Understanding Oracle Mobile Security Suite" chapter in *Administering Oracle Mobile Security Suite*.
- For details about how system administrators use the OMSS Mobile Security Manager console to enroll the MAF application user's mobile device and workspace, see the "Enrolling Devices and Workspaces" chapter in *Administering Oracle Mobile Security Suite*.

- For details about how system administrators use the OMSS Mobile App Catalog to manage the MAF application provisioned to devices and workspaces, see the "Managing Mobile Apps" chapter in *Administering Oracle Mobile Security Suite*.
- For details about how system administrators use the OMSS Mobile Security Manager console to manage access to a corporate file shared by MAF application capabilities, see the "Managing Mobile Security Policies" chapter in *Administering Oracle Mobile Security Suite*.

In the OMSS documentation library, refer to the following list of resources for details about MSAS administration tasks related to securing resources and authentication.

- For background information about MSAS, see the "Getting Started with Mobile Security Access Server" chapter in *Administering Oracle Mobile Security Access Server*.
- For details about how system administrators create a proxy application to define forward proxy URLs for protected resources accessed by MAF applications, see the "Managing Mobile Security Access Server Applications" chapter in *Administering Oracle Mobile Security Access Server*.
- For details about how system administrators attach predefined security policies to forward proxy URLs and secure access by the MAF application to protected web services, see the "Securing Mobile Security Access Server Resources" chapter in *Administering Oracle Mobile Security Access Server*.
- For details about how system administrators configure a MSAS authentication endpoint to handle authentication on the MAF application user's mobile device by the Secure Workspace app, see the "Configuring a Mobile Security Access Server Instance" chapter in *Administering Oracle Mobile Security Access Server*.

3

Configuring the Content of a MAF Application

This chapter describes using the MAF Application Editor and MAF Features Editor to define the display behavior of the mobile application's springboard and navigation bar and how to designate content by embedding application features.

This chapter includes the following sections:

- Section 3.1, "Introduction to Configuring MAF Application Display Information"
- Section 3.2, "Setting Display Properties for a MAF Application"
- Section 3.3, "Changing the Launch Screen for Your MAF Application on iOS"
- Section 3.4, "Setting Display Properties for an Application Feature"

3.1 Introduction to Configuring MAF Application Display Information

You can configure the display information that appears to the end users of your MAF application by setting values in the MAF Application Editor, which provides a declarative interface for editing the `maf-application.xml` file. Examples of the type of information you enter for the application include the display name, a description of your application, and the application's version number. You can enter similar information for individual application features that you include in your MAF application or distribute for use in other MAF applications. Additionally, you can specify icons that an application feature displays when it renders in a MAF application's navigation bar or springboard.

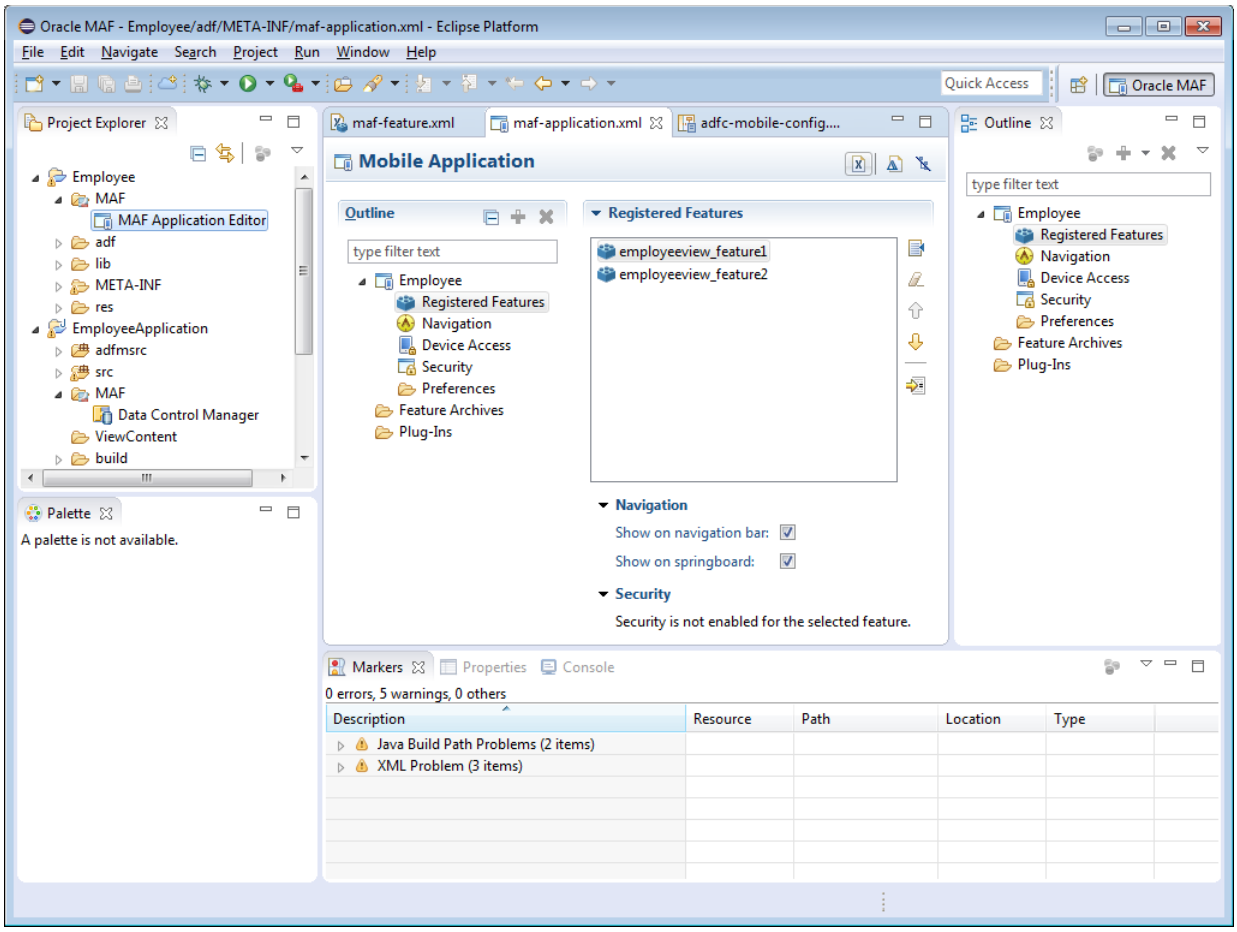
3.2 Setting Display Properties for a MAF Application

Figure 3–1 shows the MAF Application Editor where you set the display name and application ID of your MAF application.

Before you begin:

Open the MAF Application Editor by double-clicking **MAF Application Editor** (located in the MAF node of the assembly project in the Project Explorer panel, as shown in Figure 3–1).

Figure 3–1 Selecting MAF Application Editor in the Project Explorer



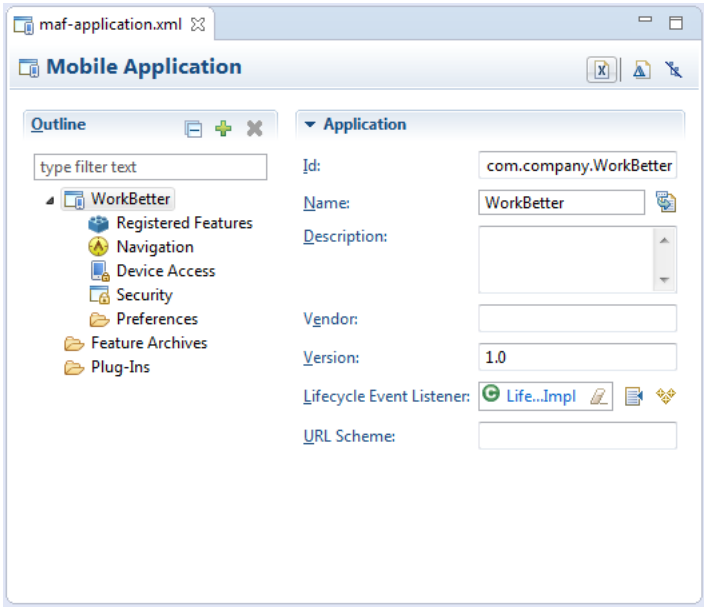
To set the basic information for a mobile application:

1. Choose the **Application** page by selecting the application's name in the Outline.

Note: By default, the editor opens the Application page.

Figure 3–2 shows the portion of the application page where you define the basic information.

Figure 3–2 Setting the Basic Information for the Mobile Application



2. Enter a display name for the application in the **Name** field. This attribute can be localized. For more information, see Section 6.1, "Introduction to MAF Application Localization."

> **Note:** MAF uses the value entered in this field as the name for the iOS archive (`.ipa` or `.app`) file that it creates when you deploy the application to an iOS-powered device or simulator. For more information, see Section 27.4.1, "How to Create an iOS Deployment Configuration."

3. Accept the default, or enter a unique ID in the **Id** field.

   To avoid naming collisions, Android and iOS use reverse package names, such as *com.company.application*. OEPE prefixes *com.company* as a reverse package to the application name, but you can overwrite this value with another as long as it is unique and adheres to the ID guidelines for both iOS- and Android-powered devices. For iOS application, see the "Creating and Configuring App IDs" section in *iOS Team Administration Guide* (available from the iOS Developer Library at `http://developer.apple.com/library/ios`). For Android, refer to the document entitled "The AndroidManifest.xml File," which is available from the Android Developers website (`http://developer.android.com/guide/topics/manifest/manifest-intro.html`). You can overwrite this ID in the deployment profiles described in Section 27.3.1, "How to Create an Android Deployment Configuration" and Section 27.4.1, "How to Create an iOS Deployment Configuration."

> **Note:** To ensure that an application deploys successfully to an Android-powered device or emulator, the ID must begin with a letter, not with a number or a period. For example, an ID comprised of a wholly numeric value, such as 925090 (*com.company.925090*) will prevent the application from deploying. An ID that begins with letters, such as hello925090 (*com.company.hello925090*) will enable the deployment to succeed.

4. Enter a short, but detailed summary of the application that describes the application's purpose in the **Description** field.
5. Enter the version in the **Version** field.
6. Enter the name of the vendor who originated this application in the **Vendor** field.
7. In the **Lifecycle Event Listener** field, enter a class with code that executes in response to lifecycle events in your MAF application. A newly-created MAF application specifies `application.LifeCycleListenerImpl` by default.

   For more information, see Chapter 10, "Using Lifecycle Listeners in MAF Applications."

   The default application listener class is `application.LifeCycleListenerImpl`. MAF does not register this class by default, because it starts the JVM and therefore may not be preferable for each mobile application. You must instead register this class manually using the Select Class dialog, shown in Figure 3–2. After you close this dialog, OEPE updates the `<adfmf:application>` element with a `listener-class` attribute, as illustrated in the example below of adding the listener-class Attribute.

   > **Note:** The application lifecycle listener must remain within the application controller project (its default location).

```
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                   name="OracleMobileApplication"
                   id="com.company.OracleMobileApplication"
                   appControllerFolder="ApplicationController"
                   listener-class="application.LifeCycleListenerImpl"
                   version="1.1"
                   vendor="Oracle">
   <adfmf:description>This is a mobile application</adfmf:description>
   <adfmf:featureReference id="feature1"/>
</adfmf:application>
```

   For more information, see Section 10.1, "Introduction to Lifecycle Listeners in MAF Applications."

3.3 Changing the Launch Screen for Your MAF Application on iOS

MAF provides a HTML page to display the launch screen that appears to end users when your MAF application starts up on an iOS device.

This HTML page is designed to render responsively on the iOS device where the MAF application runs. That is, the page uses the available screen and displays the copyright information and logo in a size appropriate to the device.

You can create a custom HTML page where you define an alternative launch screen. You do this from the Launch Screen section of the Navigation tab of the MAF Application Editor. The HTML page you create is saved in the *application project*/ViewContent directory of your MAF application. The following XML entries appear in the maf-application.xml file's source if you create a HTML page to use as a launch screen:

```
...
  <adfmf:configuration>
    <adfmf:launchScreen url="custom-launch-screen.html"/>
  </adfmf:configuration>
...
```

The URL attribute defines the path, relative to the *application project*/ViewContent directory, that the application uses to find the HTML page you create as the launch screen.

View the HTML page that MAF renders as the default launch screen for iOS devices for ideas on how to create a custom HTML page to render as the launch screen. The default launch screen (maf-launch-screen.html) can be found in the following sub-directory of the deployment profile that you use to first deploy the MAF application:

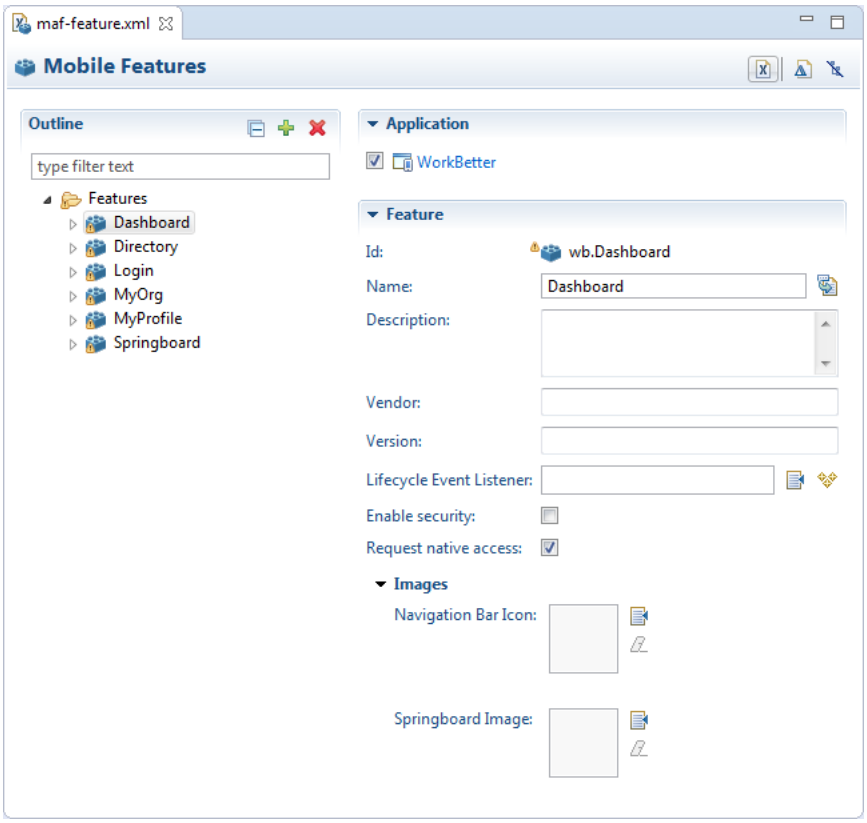
```
.../FARs/OracleStandardADFmfUiComponents/public_html/
```

3.4 Setting Display Properties for an Application Feature

Each MAF application must have at least one application feature. Application features can be developed independently of each other (and also from the MAF application itself). The MAF Features Editor enables you to define the child elements of <adfmf:features> in the maf-feature.xml file to differentiate the application features by assigning each application feature a name, an ID, and setting how their content can be implemented. Using the overview editor for application features, you can also control the runtime display of the application feature within MAF application and designate when an application feature requires user authentication.

Figure 3–3 shows the MAF Features Editor for the WorkBetter sample application. Use this tab to specify information such as the name of the application feature and the icons that display in the springboard and navigation bar.

Figure 3–3 Application Features in MAF Features Editor



4

Configuring MAF Application Features

This chapter introduces application features, describes how you add one to your MAF application, and discusses how you configure an application feature to render as a sliding window.

This chapter includes the following sections:

- Section 4.1, "Introduction to MAF Application Features"
- Section 4.2, "Configuring Application Navigation"
- Section 4.3, "What Happens When You Configure the Navigation Options"
- Section 4.4, "What Happens When You Set the Animation for the Springboard"
- Section 4.5, "What You May Need to Know About Custom Springboard Application Features with HTML Content"
- Section 4.6, "What You May Need to Know About Custom Springboard Application Features with MAF AMX Content"
- Section 4.7, "What You May Need to Know About the Runtime Springboard Behavior"
- Section 4.8, "Navigating a MAF Application Using Android's Back Button"
- Section 4.9, "Creating a Sliding Window in Your MAF Application"
- Section 4.10, "Using Custom URL Schemes in MAF Applications"

4.1 Introduction to MAF Application Features

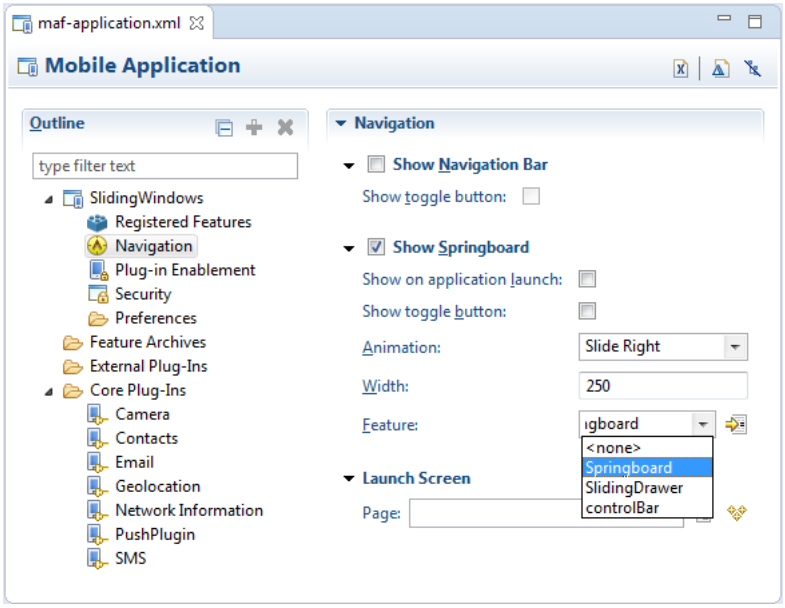
You can configure the MAF application to control the display behavior of the springboard and the navigation bar in the following ways:

- Hide or show the springboard and navigation bar to enable the optimal usage of the mobile device's interface. These options override the default display behavior for the navigation bar, which is shown by default unless otherwise specified by the application feature.
- Enable the springboard to slide from the right. By default, the springboard does not occupy the entire display, but instead slides from the left, pushing the active content (which includes the navigation bar's Home button and application features) to the right.

4.2 Configuring Application Navigation

The Navigation options of the Applications page, shown in Section 4–1, "The Navigation Options of the Application Page", enable you to hide or show the navigation bar, select the type of springboard used by the application, and define how the springboard reacts when users page through applications.

Figure 4–1 The Navigation Options of the Application Page



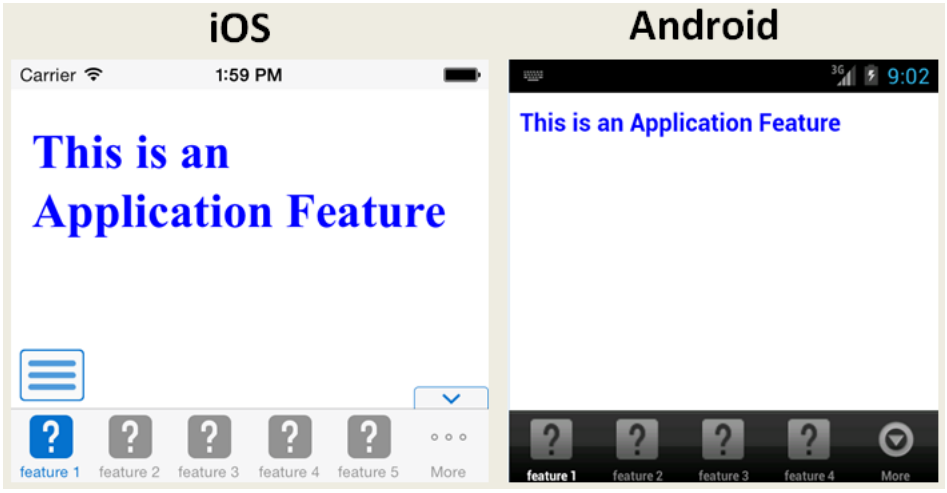
4.2.1 How to Set the Display Behavior for the Navigation Bar

The default behavior for a MAF application is to show the navigation bar on application launch. You can change this default behavior in the MAF Application editor.

To set the display behavior for the navigation bar:

1. Select **Show Navigation Bar** to enable the MAF application to display its navigation bar (instead of the springboard), by default, as shown in Figure 4–2

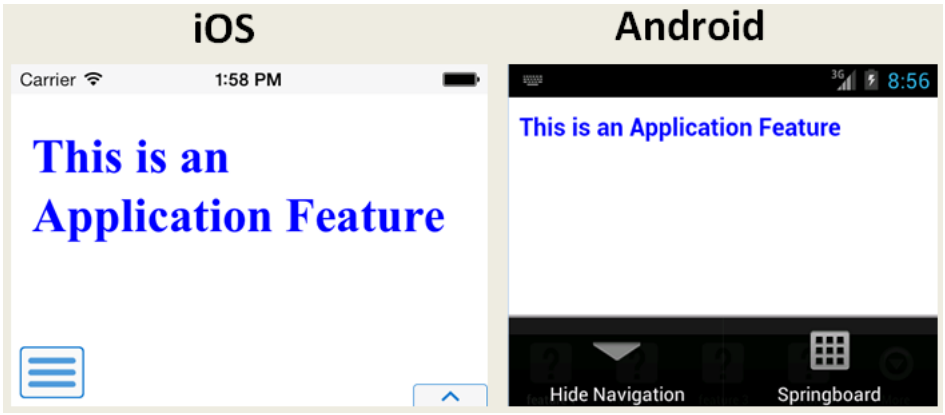
Figure 4–2 The Navigation Bar, Shown By Default



If you clear this option, then you hide the navigation bar when the application starts, presenting the user with the springboard as the only means of navigation. Because the navigation bar serves the same purpose as the springboard, hiding it can, in some cases, remove redundant functionality.

2. Select **Show Toggle Button** to hide the navigation bar when the content of a selected application feature is visible. Figure 4–3 illustrates this option, showing how the navigation bar illustrated in Figure 4–2 becomes hidden by the application feature content.

Figure 4–3 Hiding the Navigation Bar



This option is selected by default; the navigation bar is shown by default if the show or hide state is not specified by the application feature.

4.2.2 How to Set the Display Behavior for the Springboard

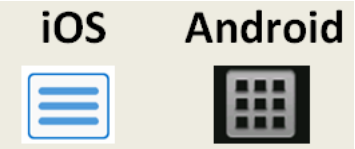
By default, a MAF application does not show a springboard on application launch. You can change this default behavior in the MAF Application editor.

To set the display behavior for the springboard:

1. In the MAF Application editor, select **Navigation** and in the Navigation tab, select **Show Springboard**.

2. to display the default springboard provided by MAF. The default springboard is implemented as a MAF AMX page.
3. Select **Show on application launch** to enable the MAF application to display the springboard to the end user after the MAF application has been launched.
4. To enable the display of the springboard button, select **Show toggle button**. Figure 4–4 shows this button within the context of an application feature.

Figure 4–4 The Springboard Toggle Button



4.2.3 How to Set the Slideout Behavior for the Springboard

If you configure your MAF application to use a springboard, you can set the slideout behavior of the springboard in the MAF Application Editor.

To set the slideout behavior for the springboard:

1. For the animation choose **Slide Right**. The springboard occupies an area determined by the number of pixels (or the percent) entered for the **Width** option. If you select **<none>** for the animation, then the springboard cannot slide from the right (that is, MAF does not provide the animation to enable this action). The springboard takes the entire display area.
2. Set the width (in pixels). The default width of a springboard on an iOS-powered device is 320 pixels. On Android-powered devices, the springboard occupies the entire screen by default, thereby taking up all of the available width.

> **Note:** If the springboard does not occupy the entire area of the display, then an active application feature occupies the remainder of the display. For more information, see Section 4.4, "What Happens When You Set the Animation for the Springboard."

4.2.4 How to Set the Display Order for Application Features

You set the display order for application features in the Feature References page of the maf-application.xml overview editor.

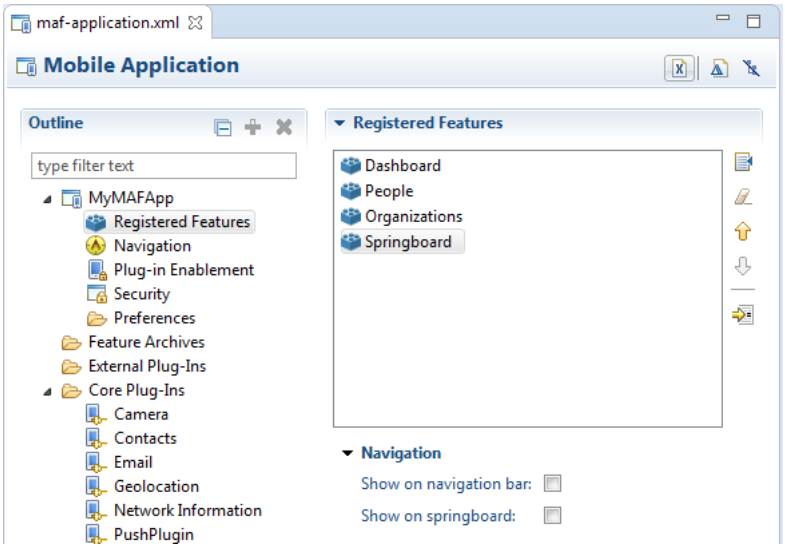
To set the display order for application features:

1. In the MAF Application Editor, select **Registered Features**.
2. Use the up- and down-arrows shown to arrange the display order of the registered features. The top-most application feature is the default application feature. Depending on the security configuration for this application, MAF can enable users to login anonymously to view unsecured content, or it can prompt users to provide their authentication credentials.

> **Tip:** The top-most ID in the Feature References table is the first application feature to display within the MAF application. See, for example, the Dashboard application feature in the WorkBetter sample application.

3. Set the springboard and navigation bar display behavior for the application feature using **Show on navigation bar** and **Show on springboard**. Figure 4–5 shows selecting these options to prevent an application feature from displaying in the navigation bar.

Figure 4–5 Changing the Navigation Options



The springboard and the navigation bar display by default. If both the navigation bar and springboard options are not selected, then the application feature only displays if it is in the first position.

Note: Because springboard applications do not display on the navigation bar or within the springboard of a MAF application, Show on navigation bar and Show on springboard must both be deselected for feature references used as custom springboard application features.

4.3 What Happens When You Configure the Navigation Options

Setting the springboard and navigation bar options updates or adds elements to the `adfmf:application.xml` file's `<adfmf:navigation>` element. For example, selecting `None` results in the code updated with `<springboard enabled="false">` as illustrated in the following example.

```
<adfmf:application>
  ...
  <adfmf:navigation>
     <adfmf:navigationBar enabled="true"/>
     <adfmf:springboard enabled="false"/>
  </adfmf:navigation>
</adfmf:application>
```

Tip: By default, the navigation bar is enabled, but the springboard is not. If you update the XML manually, you can enable the springboard as follows:

```
<adfmf:application>
  ...
<adfmf:navigation>
    <adfmf:springboard enabled="true"/>
  </adfmf:navigation>
  ...
</adfmf:application>
```

Figure 4–1 illustrates how the enabled attribute is set to true when you select the options.

Tip: Because the springboard fills the entire screen of the device, the navigation bar and the springboard do not appear simultaneously.

If you select Custom and then select the application feature used as the springboard, the editor populates the `<adfmf:navigation>` element as illustrated in Figure 4–2. The id attribute refers to an application feature defined in the `maf-feature.xml` file that is used as a custom springboard.

Example 4–1 Enabling the Display of the Default Springboard

```
<adfmf:application>
  ...
  <adfmf:navigation>
     <adfmf:navigationBar enabled="true"/>
     <adfmf:springboard enabled="true"/>
  </adfmf:navigation>
</adfmf:application>
```

Example 4–2 Configuring a Custom Springboard

```
<adfmf:navigation>
    <adfmf:springboard enabled="true">
      <adfmf:springboardFeatureReference id="springboard"/>
    </adfmf:springboard>
  </adfmf:navigation>
```

4.4 What Happens When You Set the Animation for the Springboard

Figure 4–3 shows the navigation block of the `maf-application.xml` file, where the springboard is set to slide out and occupy a specified area of the display (213 pixels).

The following line disables the animation:

```
<adfmf:springboard enabled="true" animation="none"/>
```

The following line sets the springboard to occupy 100 pixels from the left of the display area and also enables the active application feature to occupy the remaining portion of the display:

```
<adfmf:springboard enabled="true" animation="slideright" width="100px"/>
```

In addition to the animation, Figure 4–3 demonstrates the following:

- The use of the `showSpringboardAtStartup` attribute, which defines whether the springboard displays when the application starts. (By default, the springboard is displayed.)
- The use of the `navigationBar`'s `displayHideShowNavigationBarControl` attribute.

To prevent the springboard from displaying, set the enabled attribute to `false`.

Example 4–3 Configuring Springboard Animation

```
<adfmf:navigation>
   <adfmf:navigationBar enabled="true"
                        displayHideShowNavigationBarControl="true"/>
   <!-- default interpretation of width is pixels -->
   <adfmf:springboard enabled="true"
                      animation="slideright"
                      width="213"
                      showSpringbaordAtStartup="true"/>
</adfmf:navigation>
```

4.5 What You May Need to Know About Custom Springboard Application Features with HTML Content

The default HTML springboard page provided by MAF uses the following technologies, which you may also want to include in a customized login page:

- CSS—Defines the colors and layout.
- JavaScript—The `<script>` tag embedded within the springboard page contains references to the methods described in Appendix B, "Local HTML and Application Container APIs" that call the Apache Cordova APIs. In addition, the HTML page uses JavaScript to respond to the callbacks and to detect page swipes. When swipe events are detected, JavaScript enables the dynamic modification of the style sheets to animate the page motions.

  A springboard authored in HTML (or any custom HTML page) can leverage the Apache Cordova APIs by including a `<script>` tag that references the `base.js` library. You can determine the location of this library (or other JavaScript libraries) by first deploying a MAF application and then locating the `www/js` directory within platform-specific artifacts in the deploy directory. For an Android application, the `www/js` directory is located within the Android application package (`.apk`) file at:

  application workspace directory/deploy/*deployment profile name*/*deployment profile name*.apk/assets/www/js

  For iOS, this library is located at:

  application workspace directory/deploy/*deployment profile name*/temporary_xcode_project/www/js

  For more information, see Section B.1, "Using MAF APIs to Create a Custom HTML Springboard Application Feature."
- WebKit—Provides smooth animation of the icons during transitions between layouts as well as between different springboard pages. For more information on the WebKit rendering engine, see `http://www.webkit.org/`.

Springboards written in HTML are application features declared in the `maf-feature.xml` file and referenced in the `maf-application.xml` file.

4.6 What You May Need to Know About Custom Springboard Application Features with MAF AMX Content

Like their HTML counterparts, springboards written using MAF AMX are application features that are referenced by the MAF application. Because a springboard is typically written as a single MAF AMX page rather than as a task flow, it uses the `gotoFeature` method to launch the embedded application features.

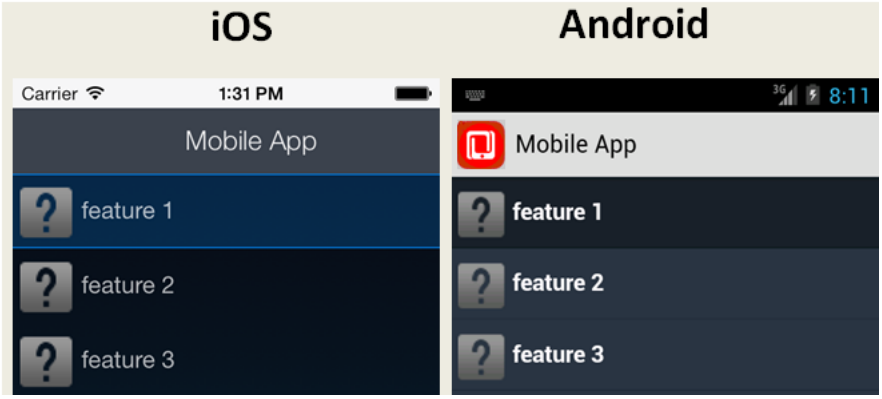
> **Note:** A custom springboard page (authored in either HTML or MAF AMX) must reside within a view project which also contains the `maf-feature.xml` file.

The default springboard (springboard.amx, illustrated in the following example) is implemented as a MAF AMX application feature.

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:panelPage id="pp1">>
      <amx:facet name="header">
         <amx:outputText value="#{bindings.name.inputValue}" id="ot3"/>
      </amx:facet>
      <amx:listView var="row"
                    value="#{bindings.features.collectionModel}"
                    fetchSize="#{bindings.features.rangeSize}"
T
                    id="lv1"
                    styleClass="amx-springboard">
         <amx:listItem showLinkIcon="false"
                       id="li1"
                       actionListener="#{bindings.gotoFeature.execute}">
            <amx:tableLayout id="tl1"
                             width="100%">
               <amx:rowLayout id="rl1">
                  <amx:cellFormat id="cf11"
                                  width="46px"
                                  halign="center">
                     <amx:image source="#{row.image}"
                                id="i1"
                                inlineStyle="width:36px;height:36px"/>
                  </amx:cellFormat>
                  <amx:cellFormat id="cf12"
                                  width="100%"
                                  height="43px">
                     <amx:outputText value="#{row.name}"
                                     id="ot2"/>
                  </amx:cellFormat>
               </amx:rowLayout>
            </amx:tableLayout>
               <amx:setPropertyListener from="#{row.id}"
                                        to="#{pageFlowScope.FeatureId}"/>
         </amx:listItem>
      </amx:listView>
   </amx:panelPage>
</amx:view>
```

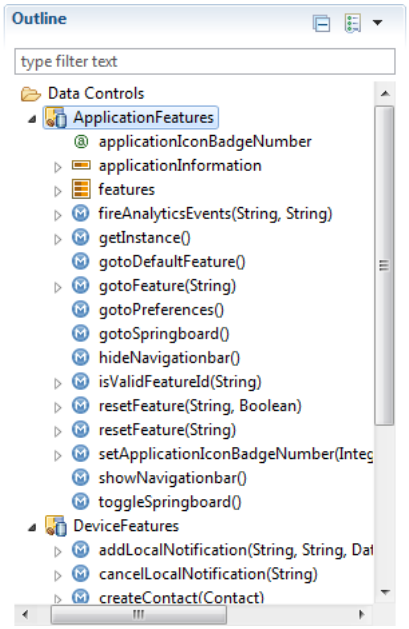
As shown in Figure 4–6, a MAF AMX file defines the springboard using a List View whose List Items are the MAF application's embedded application features. These application features, once deployed, are displayed by their names and associated icons. The `gotoFeature` method of the `AdfmfContainerUtilities` API provides the page's navigation functions. For a description of using this method to display a specific application feature, see Section B.2.5, "gotoFeature." See also Section 12.3.15, "How to Use List View and List Item Components."

Figure 4–6 The Default Springboard



MAF provides the basic tools to create a custom springboard (or augment the default one) in the ApplicationFeatures data control. This data control, illustrated in Figure 4–7, enables you to declaratively build a springboard page using its data collections of attributes that describe both the MAF application and its application features. For an example of a custom springboard page, see the APIDemo sample application. For more information on this application (and other samples that ship with MAF), see Section 1.4, "MAF Sample Applications.".

Figure 4–7 ApplicationFeatures Data Control



The ApplicationFeatures data control exposes methods that the AdfmfContainerUtilities class from the following package provides to implement navigation in a MAF application:

`oracle.adfmf.framework.api`

Table 4-1 describes some of the methods that you can drag from the ApplicationFeatures data control and drop on a MAF AMX page to navigate in your MAF application.

For more information about using data controls, see Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX." For more information about the `AdfmfContainerUtilities` class, see *Java API Reference for Oracle Mobile Application Framework*.

Table 4–1 Application Feature Methods

Method	Description
`gotoDefaultFeature`	Navigates to default application feature.
`gotoFeature`	Navigates to a specific application as designated by the parameter that is passed to this method.
`gotoPreferences`	Navigates to the preferences page.
`gotoSpringboard`	Navigates to the springboard.
`hideNavigationbar`	Hides the navigation bar.
`showNavigationbar`	Displays the navigation bar (if it is hidden).
`resetFeature`	Resets the application feature that is designated by the parameter passed to this method.
`hideSpringboard`	Hides the springboard.
`showSpringboard`	Shows the springboard.
`toggleSpringboard`	Toggles the display of the springboard.

4.7 What You May Need to Know About the Runtime Springboard Behavior

If you chose the **Show on application launch** option and defined the slideout width to full size of the screen, then MAF loads the default application feature in the background at startup. When the MAF application hibernates, MAF hides the springboard.

4.8 Navigating a MAF Application Using Android's Back Button

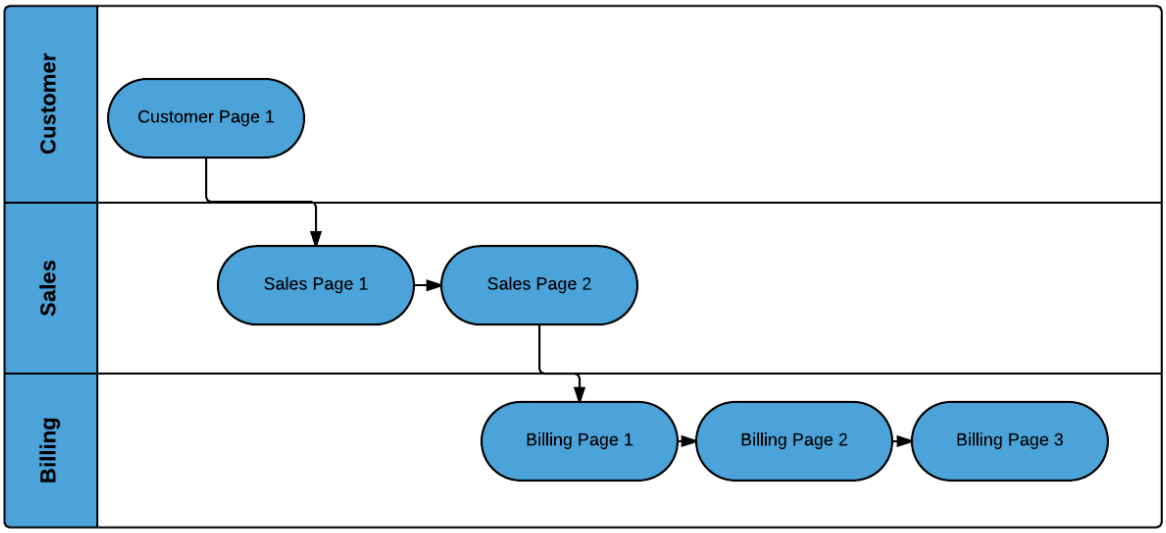
End users can navigate backwards on MAF applications using the Android system's Back button.

Figure 4–8 shows the Android system's Back button that appears on the Android navigation bar or on the Android device itself. Figure 4–8 shows the Android 4.x and Android 5.x versions of the navigation bar where this button appears.

Figure 4–8 Android's Back Button

Figure 4–9 shows a navigation flow on a MAF application where an end user has navigated between three application features (Customer, Sales, and Billing) to the Billing Page 3 page of the Billing application feature.

Figure 4–9 Navigation Flow Between Application Features and Pages in a MAF Application



The default MAF application behavior in response to an end user tapping Android's system Back button on:

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

An end user may choose to tap Android's system Back button instead of a MAF AMX button that you expose on the UI with a value of `__back` for the action attribute. The behavior is the same in both scenarios. Assume, for example, that all 3 pages in the Billing application feature expose a button component with an action attribute set to `__back`. The backward navigation flow in this scenario is from Page 3 to Page 2, Page 2 to Page 1, and for the MAF application to hibernate if the end user taps the command button on Page 1.

You can override the default MAF application behavior in response to an end user tapping the Android system Back button so that the MAF application navigates elsewhere or executes some logic prior to navigating backwards. MAF provides JavaScript APIs and the MAF AMX System Action Behavior (`systemActionBehavior`) component that you can use to override the default MAF application behavior. The `systemActionBehavior` component can only be used where your application feature's content is MAF AMX pages. JavaScript can be used to override the default behavior on

application features that use MAF AMX pages, local HTML or remote URLs. You can use the `registerSystemActionOverride` JavaScript method to register a handler to be invoked when an end user taps the Android Back button. Use the `unregisterSystemActionOverride` JavaScript method to remove a handler from being invoked. Both methods are in the `adf.mf.api` namespace. For more information, see *JSDoc Reference for Oracle Mobile Application Framework*.

For more information about using the `systemActionBehavior` component, see HowSection 4.8.1, "How to Configure Behavior of the Android System Back Button."

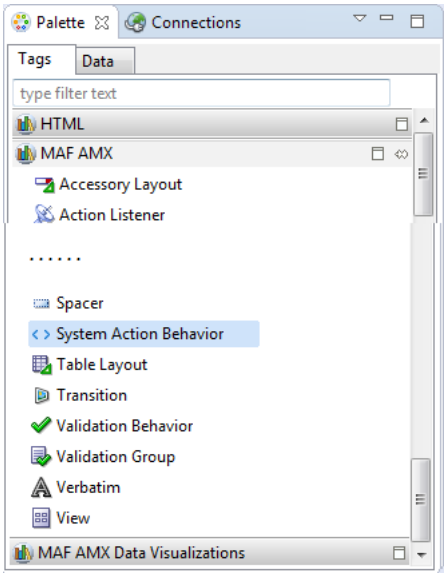
The default MAF application behavior in response to an end user tapping Android's system Back button in a MAF application created using a release prior to MAF 2.2.0 was to navigate back between application features. This meant that, for example in Figure 4–8, an end user navigates from the Billing application feature to Sales application feature and finally Customer application feature before hibernating the MAF application. You can implement this legacy behavior in MAF applications created using this release of MAF by configuring a parameter in the maf-config.xml file, as described in the Retaining Legacy Behavior When Navigating a MAF Application Using Android's Back Button section of the *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*. Implementing this legacy behavior causes the MAF application to ignore any usage of the `systemActionBehavior` component and the `registerSystemActionOverride` JavaScript method discussed here.

4.8.1 How to Configure Behavior of the Android System Back Button

The System Action Behavior (`systemActionBehavior`) operation allows you to override the default behavior of the Android-powered device Back button to perform processing of custom logic before the navigation proceeds to the previous page of the MAF AMX application feature as defined by the task flow.

In OEPE, the System Action Behavior is located under MAF AMX in the Palette window, as Figure 4–10 shows.

Figure 4–10 System Action Behavior in the Palette



The following example demonstrates the `systemActionBehavior` element defined in a MAF AMX file. This element can only be a child of the view element.

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:systemActionBehavior id="sab1"
                             type="back"
                             actionListener="#{MyBean.onBackButton}"
                             action="#{MyBean.getNavAction}"/>
   ...
</amx:view>
```

In the preceding example, the `actionListener` and action attributes of the `systemActionBehavior` invoke Java bean methods shown in the following example. The `onBackButton` method performs processing of custom logic before the back navigation occurs. The `getNavAction` method disables the back behavior.

In addition to the System Action Behavior MAF AMX component and Java beans, you can use JavaScript to configure behavior of the Android system back button. The following example demonstrates the `feature.js` file included with the application feature. It defines a handler for the Android system back button that enables some sort of processing to take place before the back navigation occurs.

```
handleSystemBack = function()
{
   // do some processing, invoke a Java bean
   adf.mf.api.amx.doNavigation("__back");
};
adf.mf.api.registerSystemActionOverride("back", handleSystemBack);
```

The handler demonstrated in the following example prevents the back navigation from occurring.

```
handleSystemBack = function()
{
   // do nothing
};
adf.mf.api.registerSystemActionOverride("back", handleSystemBack);
```

The handler demonstrated in the following example enables the standard back navigation.

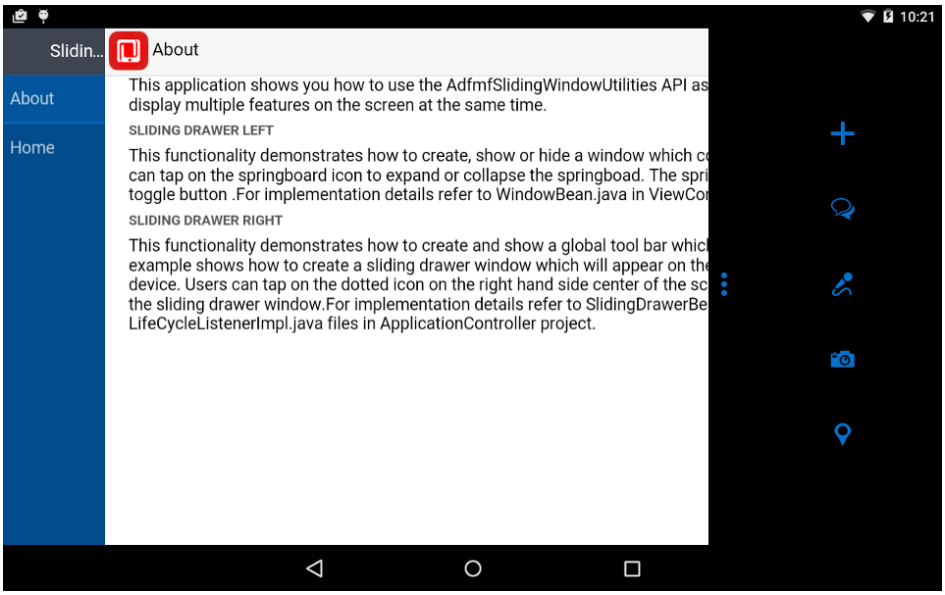
```
handleSystemBack = function()
{
   adf.mf.api.amx.doNavigation("__back");
};
adf.mf.api.registerSystemActionOverride("back", handleSystemBack);
```

4.9 Creating a Sliding Window in Your MAF Application

You can render an application feature as a sliding window. This makes the application feature display concurrently with the other application features that display within the navigation bar or springboard. You might use a sliding window to display content that is always present within the application, such as a global tool bar, or for temporary (pop-up) content, such as a help window.

Figure 4–11 shows the `SlidingDrawer` application feature from the SlidingWindow sample application, described in Appendix E, "MAF Sample Applications."." This application feature appears on the right of an application screen while overlaying other application features.

Figure 4–11 Sliding Window Overlaying Other Application Features



If you choose to render an application feature as a sliding window, you must set its **Show on Navigation Bar** and **Show on Springboard** properties to `false`.

You create a sliding window by invoking a combination of the `oracle.adfmf.framework.api.AdfmfSlidingWindowOptions` and `AdfmfSlidingWindowUtilities` classes, either from a managed bean or lifecycle listener within your application.

The following example demonstrates how the SlidingWindow sample application creates the sliding window shown in Figure 4–11 from the activate method of LifeCycleListenerImpl.java. After creating the sliding window, the SlidingWindow sample application uses SlidingDrawerBean.java to manage the display of the sliding window.

```
...
public void activate()  {
    // The argument you pass to the create method is the refId of the
    // feature in the maf-application.xml. For example,
    // <adfmf:featureReference id="fr4" refId="SlidingDrawer"
showOnNavigationBar="false"
    // showOnSpringboard="false"/>
      String slidingWindowDrawer =
AdfmfSlidingWindowUtilities.create("SlidingDrawer");

    // Note also that both showOn... values must be set to false in the config
    // file for the sliding window to appear

      SlidingDrawerBean.slidingDrawerWindow=slidingWindowDrawer;
      AdfmfSlidingWindowOptions options = new AdfmfSlidingWindowOptions();
      options.setDirection(AdfmfSlidingWindowOptions.DIRECTION_RIGHT);
      options.setStyle(AdfmfSlidingWindowOptions.STYLE_OVERLAID);
      options.setSize("0");

  }
```

For information about how to access the complete `SlidingWindow` sample application discussed here, see Appendix E, "MAF Sample Applications."

For more information about `AdfmfSlidingWindowUtilities` and `AdfmfSlidingWindowOptions`, see the *Java API Reference for Oracle Mobile Application Framework*. For more information about using lifecycle listeners, see Chapter 10, "Using Lifecycle Listeners in MAF Applications."

4.10 Using Custom URL Schemes in MAF Applications

A custom URL scheme can be used to invoke a native application from other applications.

To invoke a MAF mobile application from another application, perform the following steps:

1. Register a custom URL scheme. You configure this URL scheme in the Security tab of the MAF Application Editor using the URL Scheme field. The URL with this scheme can then be used to invoke the MAF mobile application and pass data to it.
2. In the assembly project, create a custom URL event listener class (for example, `CustomURLEventListener`) that is notified of the URL. This class must implement the `oracle.adfmf.framework.event.EventListener` interface that defines an event listener. For more information on the `oracle.adfmf.framework.event.EventListener` interface, see *Java API Reference for Oracle Mobile Application Framework*.

   Override and implement the `onMessage(Event e)` method that gets called with the URL that is used to invoke the MAF mobile application. The Event object can be used to retrieve useful information about URL payload and the application state. To get `URL` payload, use the `Event.getPayload` method. To get the application state at the time of URL event, use the `Event.getApplicationState` method. For more information, see the Event class in *Java API Reference for Oracle Mobile Application Framework*.
3. Register an application lifecycle event listener (ALCL) class.

   For more information, see Chapter 10, "Using Lifecycle Listeners in MAF Applications."

   Get an `EventSource` object in the `start` method of the ALCL class that represents the source of the custom URL event:

   ```
   EventSource openURLEventSource =
   EventSourceFactory.getEventSource(EventSourceFactory.OPEN_URL_EVENT_SOURCE_
   NAME);
   ```

   Create and add an object of the custom URL event listener class to the event source:

   ```
   openURLEventSource.addListener(new CustomURLEventListener());
   ```

A MAF application can invoke another native application in the following ways:

- Using an `amx:goLink` on a MAF AMX page whose URL begins with the custom URL scheme registered by the native application. For example:

  ```
  <amx:goLink text="Open App" id="gl1" url="mycustomurlscheme://somedata"/>
  ```

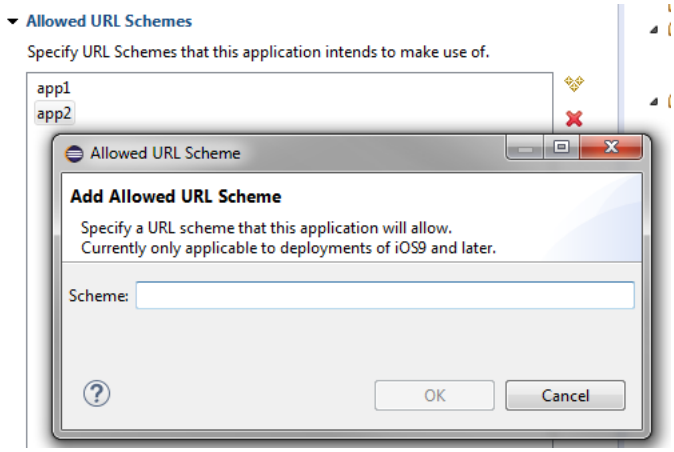
- Using an HTML link element on an HTML page whose `href` attribute value begins with the custom URL scheme registered by the native application. For example:

  ```
  <a href="mycustomurlscheme://somedata">Open App</a>
  ```

Add any custom URL schemes that your MAF application uses to invoke a native application to the Allowed Scheme list in the Security page of the MAF Application

Editor. This change addresses iOS 9's requirement that applications declare any URL schemes they use to invoke other applications. Click the Add icon in the Allow Schemes section of the Security page to add the custom URL scheme, as shown in Figure 4–12.

Figure 4–12 Registering a Custom URL Scheme that a MAF Applications Use to Invoke Another Application



5

Defining the Content Type of MAF Application Features

This chapter describes using the MAF Application Editor and MAF Features Editor to define the display behavior of the mobile application's springboard and navigation bar and how to designate content by embedding application features.

This chapter includes the following sections:

- Section 5.1, "Introduction to Content Types for an Application Feature"
- Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML"
- Section 5.3, "Defining the Application Feature Content as a MAF AMX Page or Task Flow"
- Section 5.4, "What You May Need to Know About Selecting External Resources"

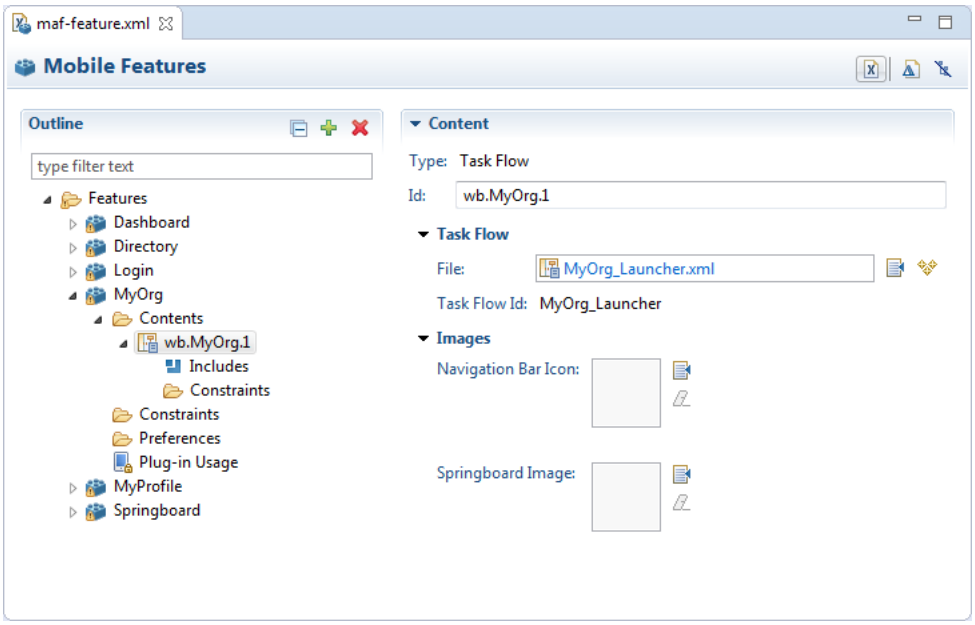
5.1 Introduction to Content Types for an Application Feature

The content type for an application feature describes the format of the user interface, which can be constructed using MAF AMX components or HTML(5) tags. An application feature can also derive its content from remotely hosted pages that contain content appropriate to a mobile context. These web pages might be a JavaServer page authored in Apache Trinidad for smartphones, or be comprised of ADF Faces components for applications that run on tablet devices. The application features embedded in a MAF application can each have different content types.

While a MAF application includes application features with different content types, applications features themselves may have different content types to respond to user- and device-specific requirements. For information on how the application feature delivers different content types, see Chapter 22, "Setting Constraints on Application Features." Adding a child element to the `<adfmf:content>` element, shown in Example 5-1, enables you to define how the application feature implements its user interface.

The Content tab of the MAF Features Editor, shown in Figure 5–1, provides you with dropdown lists and fields for defining the target content-related elements and attributes shown in Example 5–1. The fields within this tab enable you to set constraints that can control the type of content delivered for an application feature as well as the navigation and springboard icon images that it uses.

Figure 5–1 Defining the Implementation of the Application Feature



Example 5–1 The <adfmf:content> Element

```
<adfmf:content id="Feature1">
      <adfmf:amx file="FeatureContent.amx">
</adfmf:content>
```

5.2 Defining the Application Feature Content as Remote URL or Local HTML

The Content tab of the overview editor, shown in Figure 5–1, provides you with dropdown lists and fields for defining the target content-related elements and attributes shown in Example 5–1. The fields within this tab enable you to set constraints that can control the type of content delivered for an application feature as well as the navigation and springboard icon images that it uses.

Before you begin:

Each content type has its own prerequisites, as follows:

- Remote URL—A reference to a web application. You can enhance an existing web application for mobile usage and extend device services. Remote content can complement both MAF AMX and local HTML content by providing a local data cache and a full set of server-side data and functionality. The remote URL implementation requires a valid web address. For more information, see Chapter 20, "Implementing Application Feature Content Using Remote URLs."
- Local HTML—Reference a HTML page that is packaged within your MAF application. Such HTML pages can reference JavaScript, as demonstrated by the HelloWorld sample application described in MAF Sample Applications. Consider using this content type to implement application functionality through usage of the Cordova JavaScript APIs if the MAF is not best suited to implementing your application's functionality. For more information about JavaScript APIs and the MAF, see Appendix B, "Local HTML and Application Container APIs."

To define the application content as Remote URL or Local HTML:

1. Right-click an application feature listed in the MAF Features Editor and select **Add**.
2. In the New Object dialog, either **Local URL** or **Remote HTML** and click **OK**.
3. Define the content-specific parameters:
   - For local HTML content, enter the location of the local bundle or create the HTML page by clicking **Add** in the URL field, and entering a name for the HTML file in the New Mobile HTML Page dialog, and then building the page using OEPE's HTML editor. Because this is an application feature, this page is stored within the ViewContent folder of the view project.
   - For remote URL content, select the connection that represents address of the web pages on the server (and the location of the launch page).

     You can create this connection by first clicking **Add** and then completing the Create URL Connection dialog. This connection is stored in the `connections.xml` file.
4. If needed, do the following:
   - Enter constraints that describe the conditions under which this content is available to users. For more information, see Chapter 22, "Setting Constraints on Application Features."
   - Select navigation bar and springboard images.

5.3 Defining the Application Feature Content as a MAF AMX Page or Task Flow

The Content tab of the Overview editor, shown in Figure 5–1, provides you with dropdown lists and fields for defining the target content-related elements and attributes shown in Example 5–1. The fields within this tab enable you to set constraints that can control the type of content delivered for an application feature as well as the navigation and springboard icon images that it uses.

Before you begin:

Each content type has its own prerequisites, as follows:

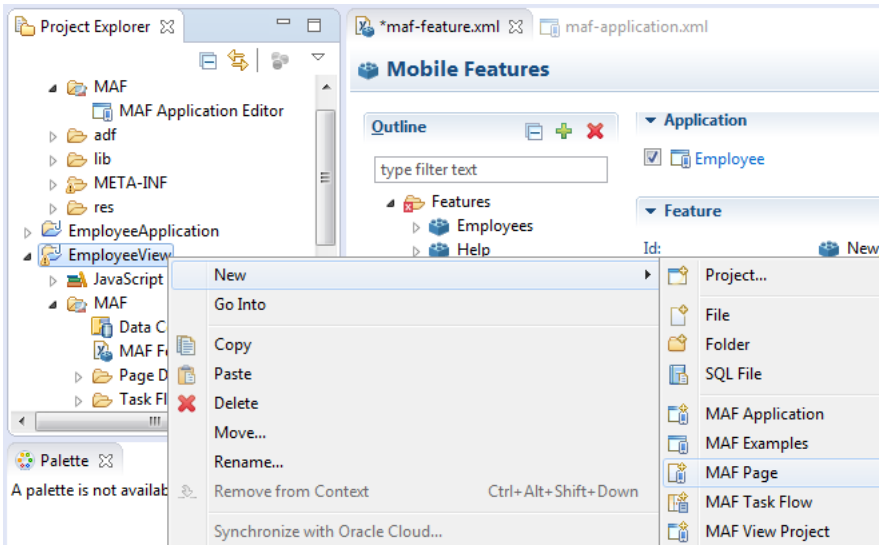
- **AMX Page**—The default content type for application features. For more information about MAF AMX pages, see Chapter 11, "Creating MAF AMX Pages."

  An application feature implemented as MAF AMX requires a view (that is, a single MAF AMX page) or a bounded or unbounded task flow. Including a JavaScript file provides rendering logic to the MAF AMX components or overrides the existing rendering logic. Including a style sheet (CSS) with selectors that specify a custom look and feel for the application feature, one that overrides the styles defined at the MAF application level that are used by default for application features. In other words, you ensure that the entire application feature has its own look and feel.

  If you create the MAF AMX pages as well as the MAF application that contains them, you can create both using the wizards in the New Gallery. You access these wizards from **File** > **New**. Alternatively, you can create an MAF AMX page using the context menu shown in Figure 5–2 that appears when you right-click the view project in the Project Explorer and then choose **New**.

Note: When manually editing references to task flows, MAF AMX pages, CSS and JavaScript files in MAF Features Editor, keep in mind that file systems used on devices may enforce case-sensitivity and may not allow special characters. To ensure that these files can be referenced, check the mobile device specification.

Figure 5–2 Context Menu for Creating an MAF Page

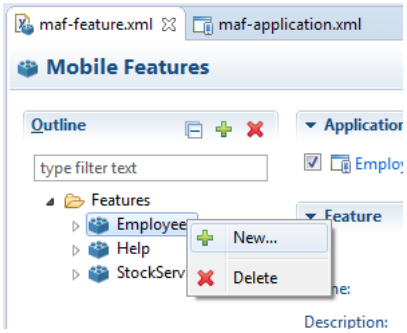


- **Task Flow**—Provides a modular approach to defining control flow in your application feature. Use a task flow to define a collection of activities that make up a task. Examples of activities that you can include in a task flow are views (use to display MAF AMX pages), method calls (use to invoke managed bean methods), and task flow calls (use to call other task flows). For more information about task flows, see Section 11.2, "Creating Task Flows."

To use a MAF AMX page or task flow as application feature content:

1. Select an application feature listed in the Outline of the MAF Feature Editor.
2. Click the right mouse button and select **New**, as shown in Figure 5–3.

Figure 5–3 Context Menu for New Feature



3. In the New Object dialog, expand the Content node, if necessary and choose one of the following content types and click **OK**:
   - **AMX Page**

- **Task Flow**

4. Define the content-specific parameters:
   - For an AMX Page, click **Browse** to specify an existing page, or click **Create** to open the **New MAF Page** dialog and select the parent folder for your new AMX page.
   - For a Task Flow, click **Browse** to specify an existing page, or click **Create** to open the **New MAF Task Flow** dialog.
5. Give your page a descriptive name that will make it easier to identify when working on your mobile application.
6. Click **Finish** to save the page.

Note: The images, style sheet, and JavaScript files must reside within the **ViewContent** folder to enable deployment. See Section 5.4, "What You May Need to Know About Selecting External Resources."

5.4 What You May Need to Know About Selecting External Resources

To enable deployment, all resources referenced by the following attributes must be located within the `ViewContent` folder of the view project.

- The `icon` and `image` attributes for `<adfmf:feature>` (for example, `<adfmf:feature id="PROD" name="Products" icon="feature_icon.png" image="springboard.png">`). See also, Section 3.4, "Setting Display Properties for an Application Feature."
- The `icon` and `image` attributes for `<adfmf:content>` (for example, `<adfmf:content id="PROD" icon="feature_icon.png" image="springboard_image.png">`). See also Section 5.1, "Introduction to Content Types for an Application Feature."
- The `file` attribute for `<adfmf:amx>` (for example, `<adfmf:amx file="PRODUCT/home.amx" />`). See also Section 5.1, "Introduction to Content Types for an Application Feature."
- The `url` attribute for `<adfmf: localHTML>` (for example, `<adfmf:localHTML url="oracle.hello/index.html"/>`). See also Section 5.1, "Introduction to Content Types for an Application Feature" and Section 29.6.4.2, "The Custom Login Page."
- The file attribute defined for `type=stylesheet` and `type=JavaScript` in `<adfmf:includes>` (for example, `<adfmf:include type="JavaScript" file="myotherfile.js"/>` or `<adfmf:include type="StyleSheet" file="resources/css/stylesheet.css" id="i3"/>`). See also Chapter 7, "Skinning MAF Applications."

MAF does not support resources referenced from another location, meaning that you cannot, for example, enter a value outside of the `ViewContent` directory using `../` as a prefix.

6 Localizing MAF Applications

This chapter describes how to use resource bundles where you can define text and image resources that render if your MAF application runs on devices that use multiple languages. The chapter also describes the design-time support that OEPE provides to create and edit resource bundles.

This chapter includes the following sections:

- Section 6.1, "Introduction to MAF Application Localization"
- Section 6.2, "Setting Resource Bundle Options for a MAF Application"
- Section 6.3, "Defining Text Resources in a Base Resource Bundle"
- Section 6.4, "Creating Locale-Specific Resource Bundles"
- Section 6.5, "Editing Resources in Resource Bundles"
- Section 6.6, "Localizing Image Files in a MAF Application"
- Section 6.7, "MAF Support of Languages"
- Section 6.8, "Localizable MAF Properties"

6.1 Introduction to MAF Application Localization

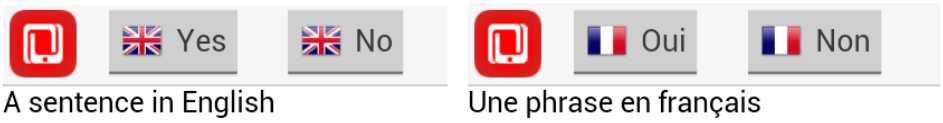
Localization is the process of adapting a product to a specific locale. With localization, a MAF application uses the same language as the mobile device on which it is deployed. For example, if French is set as the device language, the text resources of a localized MAF application appear in French.

MAF facilitates the localization process by providing design-time menus for the definition of text resources that appear on the user interface in one or more resource bundles. Define your MAF application's text resources in a base resource bundle. If your application does not include a locale-specific resource bundle for the locale of the device, these text resources are rendered on the user interface of the application on the user's device. Create locale-specific resource bundles for each locale to be supported. In these locale-specific resource bundles, provide translations of the text resources that you had defined in the base resource bundle.

Figure 6–1 shows an example where an application renders `commandButton` and `outputText` components. The text and image resources that appears in the components vary depending on the language set on the mobile device. On the left, the components render a text resource in English because the base resource bundle contains text resources with English values and the device where the application runs uses English (or a language that is not French). On the right, the same components render text resources in French because the MAF application contains a locale-specific resource

bundle (_fr) with translations of the values in the base resource bundle and the mobile device's language setting is French.

Figure 6–1 Localized Text Resources



Use the following steps to localize your MAF application:

1. Determine the number of resource bundles in your MAF application, as described in Section 6.2, "Setting Resource Bundle Options for a MAF Application."
2. Define the text resources in your base resource bundle to be rendered when your MAF application runs in a locale for which you do not provide a locale-specific resource bundle.

   For more information, see Section 6.3, "Defining Text Resources in a Base Resource Bundle."
3. Create locale-specific resource bundles where you provide translations of the text resources that you had defined in the base resource bundle.

   For more information, Section 6.4, "Creating Locale-Specific Resource Bundles."
4. Create locale-specific versions of image files and other resources that are needed to complete the localization of your MAF application.

   For more information, see Section 6.6, "Localizing Image Files in a MAF Application."

6.2 Setting Resource Bundle Options for a MAF Application

A MAF application's resource bundles use the XML Localization File format (XLIFF). OEPE creates a `.xlf` file resource bundle the first time that you enter a display value in the Externalize String dialog. Invoke Select Text Resource for each property that supports a localized value.

By default, a MAF application has two resource bundles:

- A project-level resource bundle (default name `ViewControllerController.xlf`)
- An application-level resource bundle (default name `mafapplication.xlf`)

The application-level resource bundle contains application-level text resources. For example, you specify the application name as a text resource in this resource bundle to translate the application name into different languages. Project-level resource bundles contain the text resources that render in the MAF AMX pages of a project, or the text resources of application feature properties. For more information about these properties, see Section 6.8, "Localizable MAF Properties."

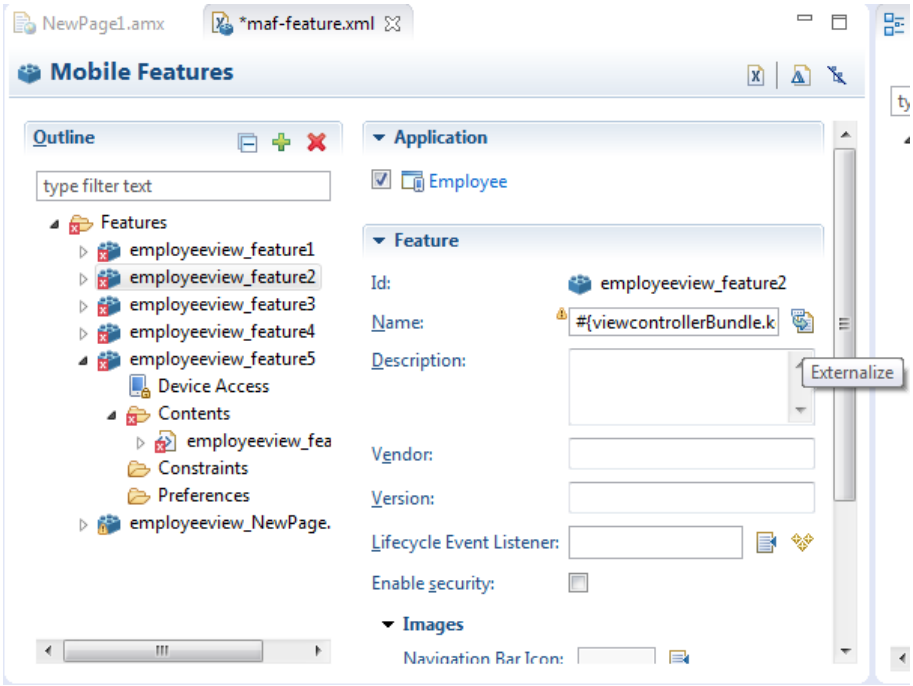
You can change the default behavior of a MAF application having one project-level resource bundle by setting the resource bundle options for the project. A MAF application can only have one application-level resource bundle.

6.3 Defining Text Resources in a Base Resource Bundle

OEPE provides the Externalized String dialog where you can define values in resource bundles for a MAF application, application feature, and MAF AMX UI component text resources that appear to end users. Access the Externalized String dialog by clicking the icon beside the property for which you want to define a text resource in the MAF Application Editor or MAF Features Editor. These properties are typically properties that display text which users can see. Examples include the name properties for the MAF application and application features, in addition to the label, text, and hintText properties that MAF AMX UI components such as button, link, and input text expose. For more information about these properties, see Section 6.8, "Localizable MAF Properties."

Figure 6–2 demonstrates how you display the Externalize context menu for a MAF feature's name. The same button is used for an application feature's name.

Figure 6–2 Selecting the Edit Externalized String Dialog

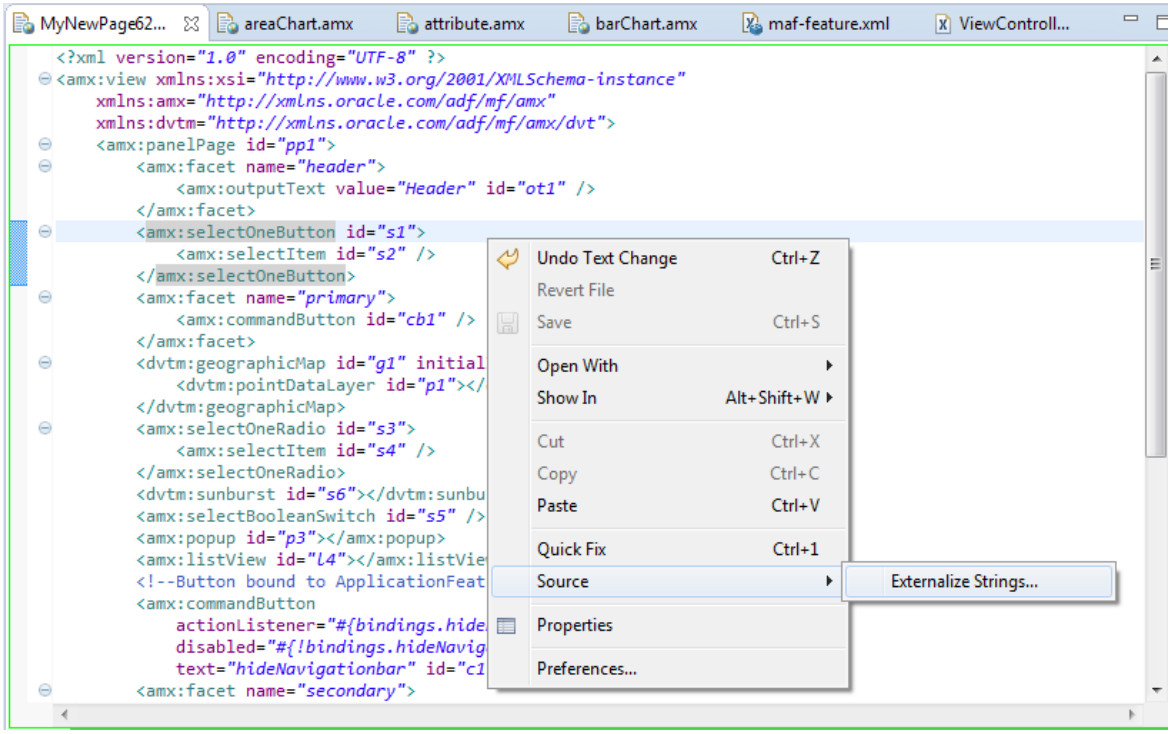


You can create resource bundles for attributes of such MAF AMX components as the `text` attribute of `<amx:commandButton>`. Table 6–4 lists these MAF AMX (`amx`) components.

To use strings in MAF AMX components:

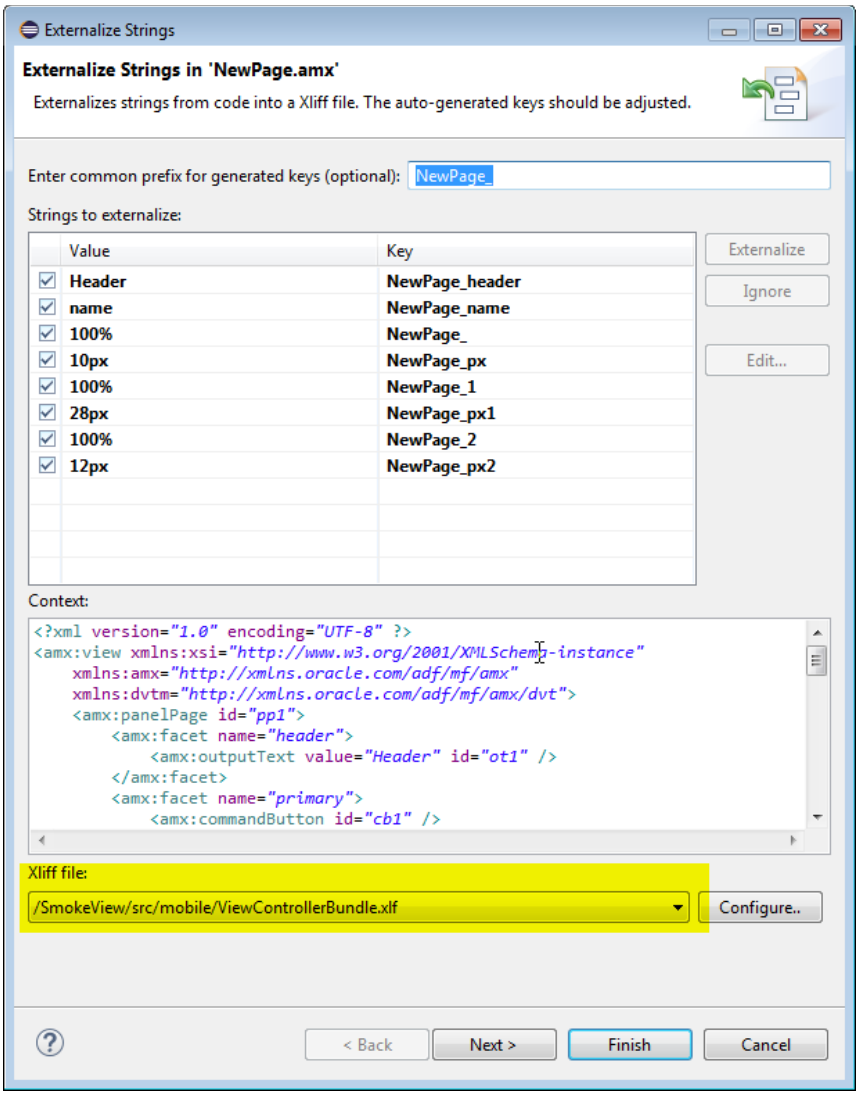
1. Select an attribute in an AMX editor, such as the value attribute defined for the `<amx:selectOneButton>` component in Figure 6–3.

Figure 6–3 Externalizing Strings by Properties



2. Click the right mouse button and select **Source > Externalize Strings**. This opens the Externalize Strings dialog, shown in Figure 6–4.
3. In the Externalize Strings dialog, edit the string resources by entering a display name, key, and then click **Finish**.

Figure 6–4 Adding a String to a Resource Bundle



6.3.1 How to Define a Text Resource in a Base Resource Bundle

You define a text resource in a resource bundle using the Externalize String dialog which can be invoked for properties that reference text resources defined in resource bundles using EL expressions.

To define a text resource in a resource bundle:

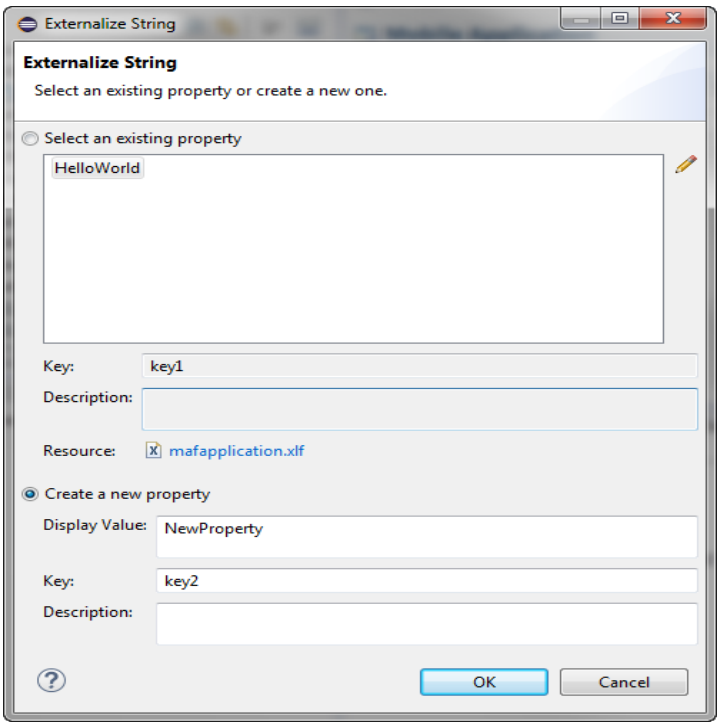
1. Choose the artifact that has a property for which you want to define a text resource. This could be the MAF application itself, an application feature or a MAF AMX UI component.

   For example, an attribute in the MAF Feature editor, such as `Name` in Figure 6–2, and click **Externalize**. This opens the Externalize String dialog.

2. In the Externalize String dialog, shown in Figure 6–5, create a new string resource by clicking Create a New Property. Enter a display value, a key, a description and then click **OK**.

The first time you create a new string resource and save the workspace, the resource file is created. When you subsequently open the Externalize String dialog, there is a link to the resource file.

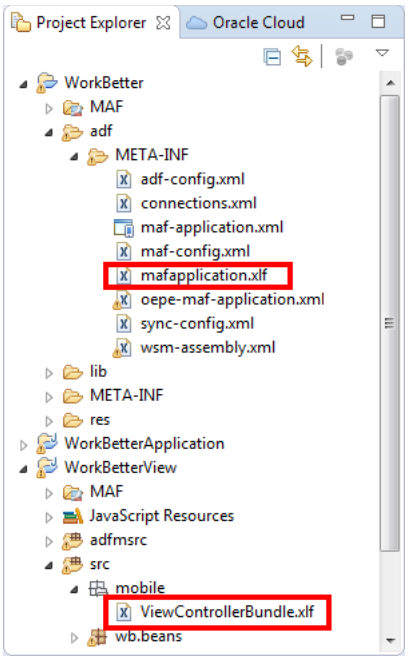
Figure 6–5 Select Edit Externalized String Dialog



6.3.2 What Happens When You Define a Text Resource in a Base Resource Bundle

OEPE writes the display value and key that you enter to a resource bundle. If this is the first time that you define a text resource for a MAF application or project, OEPE creates the resource bundle. If the text resource that you define is for an application-level property (for example, the MAF application's name property), OEPE creates an application-level resource bundle (`mafapplication.xlf`). If the text resource that you define is for a project-level property (for example, an application feature's name property), OEPE creates a project-level resource bundle (`ViewControllerBundle.xlf`), as shown in Figure 6–6.

Figure 6–6 Newly-Created Resource Bundles



OEPE creates one application-level resource bundle and one project-level resource bundle per MAF application.

The syntax of the XML that OEPE writes to the resource bundle for the key and display value is the same regardless of whether the resource bundle is an application or project-level resource bundle, as shown by the following example.

```
...
      <!-- The value of the id attribute is the value you enter in the Key input
field of the
          Select Text Resource Dialog -->
      <trans-unit id="FEATURE_ONE">
      <!-- The value of the source element is the value you enter in the Display
Value input
          field of the Select  Text Resource Dialog -->
        <source>Feature Name</source>
        <target/>
      </trans-unit>
      <trans-unit id="HEADER_VALUE_IN_PANEL">
        <source>Header Value in Panel Page</source>
        <target/>
      </trans-unit>
      <trans-unit id="COMMAND_BUTTON">
        <source>Text Display Value for a Command Button</source>
        <target/>
      </trans-unit>
    ...
```

OEPE makes the changes shown in Figure 6–1 for files where you configure a property to reference a text resource that you define in a resource bundle.

- If an attribute has been localized for the first time, OEPE adds an `<adfmf:loadbundle>` element whose `basename` attribute refers to the newly created resource bundle.

- OEPE changes the localized attribute string to an EL expression that refers to the key of the text resource defined in the resource bundle.

Example 6–1 Resource Bundle References in MAF AMX Page and MAF Configuration Files

```
<!-- maf-application.xml where a text resource has been defined for the MAF
application's name -->
        <adfmf:application ....name="#{mylocalizedmafappBundle.MY_LOCALIZED_MAF_
APPLICATION}"
        ...
                <adfmf:loadBundle basename="MyLocalizedMAFappBundle"
var="mylocalizedmafappBundle"/>

<!-- maf-feature.xml where a text resource has been defined for the application
feature's name -->
        <adfmf:loadBundle basename="mobile.ViewControllerBundle"
var="viewcontrollerBundle"/>
        ...
                <adfmf:feature id="feature1" name="#{viewcontrollerBundle.FEATURE_
ONE}">

<!-- MAF AMX page where a text resource has been defined for a command button's
text attribute -->
        <amx:loadBundle basename="mobile.ViewControllerBundle"
var="viewcontrollerBundle" id="lb1"/>
        ...
     <amx:commandButton id="cb1" text="#{viewcontrollerBundle.COMMAND_BUTTON}"/>
```

6.4 Creating Locale-Specific Resource Bundles

You create a locale-specific resource bundle if you want your MAF application to render different text resources in a specific locale. For example, if you want to provide translations of the text resources in the base resource bundle when the MAF application runs on a device that has the language set to French or Arabic, you need to create a locale-specific resource bundle for both the French and Arabic languages.

Figure 6–7 shows a MAF application where the locale-specific versions of the application-level (`mafapplication.xlf`) and project-level (`ViewControllerBundle.xlf`) resource bundles have been created to support the Arabic and French locales. You must create the locale-specific resource bundle in the same directory as the base resource bundle with a file name that uses the following format:

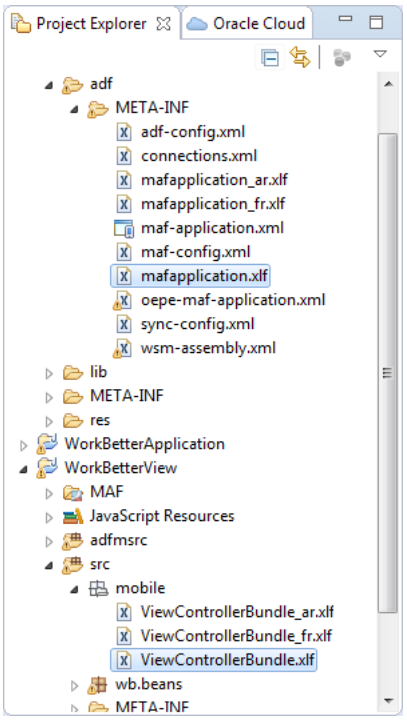
```
<BASE_RESOURCE_BUNDLE_NAME>_<LANGUAGE_TOKEN>.xlf
```

Where:

- `<BASE_RESOURCE_BUNDLE_NAME>` is the base resource bundle name
- `<LANGUAGE_TOKEN>` is in the following format:
  `<ISO-639-lowercase-language-code>`

  For a list of the languages that MAF supports, see Section 6.7, "MAF Support of Languages."

Figure 6–7 Base Resource Bundle and Locale-Specific Resource Bundles



6.4.1 How to Create a Locale-Specific Resource Bundle

Open the base bundle for which you want to create a locale-specific resource bundle and save a copy of the file with the appropriate language code in the file name. This allows you to:

- You create the locale-specific resource bundle in the same directory as the base resource bundle which is a requirement.
- A copy of all text resources in the base resource bundle appear in the locale-specific resource base bundle where you can provide translated values.

To create a locale-specific resource bundle:

1. Open the base resource bundle for which you want to create a locale-specific version:
   - Application-level base resource bundle: in the assembly project, double-click the `.xlf` file in the `ADF > META-INF` node.
   - Project-level base resource bundle: in the view project, double-click the `.xlf` file in the `src > mobile` node.
2. In OEPE, click `File > Save As` and append the language code for the locale that you want to support.

   For example, append `_fr` if you want your MAF application to support the French locale.
3. Edit the newly-created locale-specific resource bundle so that it includes the appropriate language codes.

   The following example demonstrates edits you need to make to support the French locale for an application-level and project-level resource bundle.

```
<!-- Application-level French locale-specific resource bundle -->
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="fr" original="this" datatype="x-oracle-adf">

<!-- Project-level French locale-specific resource bundle -->
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="fr" original="mobile.ViewControllerBundle_fr"
                                                    datatype="x-oracle-adf">
```

4. Edit the resource bundle to provide translations of each text resource that you want to appear in the target locale.

   For more information about editing resource bundles, see Section 6.5, "Editing Resources in Resource Bundles."

6.5 Editing Resources in Resource Bundles

After you have created the XLIFF file, or Java class file, you can edit it using the source editor. If you are using one of the MAF editors, you can invoke the Externalize command on the file.

6.6 Localizing Image Files in a MAF Application

You may want to render different image files in a MAF application depending on the locale. For example, an image may contain text or a picture of a flag, as shown in Figure 6–8.

Figure 6–8 Rendering Different Images Based on Locale

Write an EL expression for the component attribute that references the image to an entry in the resource bundle. The resource bundle entry contains the path to the image. For example, the `commandButton` components shown in Figure 6–8 define the following value for the icon attribute in the MAF AMX page that renders the components:

```
<amx:commandButton id="cb1" text="#{viewcontrollerBundle.YES}"
                            icon="#{viewcontrollerBundle.IMAGE_PATH}"/>
```

The base resource bundle (`ViewControllerBundle.xlf`) contains the following entry with the path to the image to appear when the MAF application runs in a locale that is not French.

```
<trans-unit id="IMAGE_PATH">
        <source>/images/uk.png</source>
        <target/>
        <note>Path to image file</note>
      </trans-unit>
```

The French resource bundle (`ViewControllerBundle_fr.xlf`) specifies a different image to render when the MAF application runs in a French locale, as shown in the following example.

```
<trans-unit id="IMAGE_PATH">
```

```
<source>/images/fr.png</source>
<target/>
```

Manually write the entries in the resource bundle that define the path to the image.

Once you have defined the path to the image in the source bundle, you can use the Expression Builder. For more information, see Section 13.3.2, "How to Create an EL Expression."

6.7 MAF Support of Languages

Figure 6–1 lists the languages that are supported by MAF. System messages, such as error messages, appear in these languages if the user's device is configured to use one of these languages. MAF supports language tokens for countries and regions. However, full support is only available to the extent that they appear in the list of languages. Error messages from MAF use the closest of languages provided in the list. The language is usually English unless the device is configured to use one of the languages in the table, or a more specific variant of one of these languages.

If you want an application to render additional resource strings in these languages, you must define them. Create a locale-specific resource bundle using the language token listed in Table 6-1. For example, to define resource strings for Brazilian Portuguese, create a resource `bundle named BASE_RESOURCE_BUNDLE_NAME_pt_BR.xlf`. To create resource bundles for languages that are not included in the list, create resource bundles with the following name format: `BASE_RESOURCE_BUNDLE_NAME_lowercase-ISO-639-1-language-code.xlf`. For more information about the naming convention, see Section 6.4, "Creating Locale-Specific Resource Bundles."

During deployment, OEPE transforms the language tokens to the expected values on the platform on which the application is deployed. For example, a MAF application that you deploy on the iOS platform transforms the language token `pt_BR` to `pt`, the expected language ID on the iOS platform.

Table 6–1 Resource Bundle Names for Languages Supported by MAF

Language	Resource bundle name to provide additional UI strings
Arabic	BASE_RESOURCE_BUNDLE_NAME_ar.xlf
Brazilian Portuguese	BASE_RESOURCE_BUNDLE_NAME_pt_BR.xlf
Catalan	BASE_RESOURCE_BUNDLE_NAME_ca.xlf
Czech	BASE_RESOURCE_BUNDLE_NAME_cs.xlf
Danish	BASE_RESOURCE_BUNDLE_NAME_da.xlf
Dutch	BASE_RESOURCE_BUNDLE_NAME_nl.xlf
Finnish	BASE_RESOURCE_BUNDLE_NAME_fit.xlf
French	BASE_RESOURCE_BUNDLE_NAME_fr.xlf
German	BASE_RESOURCE_BUNDLE_NAME_de.xlf
Greek (Modern)	BASE_RESOURCE_BUNDLE_NAME_el.xlf
Hebrew	BASE_RESOURCE_BUNDLE_NAME_iw.xlf
Hungarian	BASE_RESOURCE_BUNDLE_NAME_hu.xlf
Italian	BASE_RESOURCE_BUNDLE_NAME_it.xlf
Japanese	BASE_RESOURCE_BUNDLE_NAME_ja.xlf
Korean	BASE_RESOURCE_BUNDLE_NAME_ko.xlf

Table 6–1 (Cont.) Resource Bundle Names for Languages Supported by MAF

Language	Resource bundle name to provide additional UI strings
Norwegian	BASE_RESOURCE_BUNDLE_NAME_no.xlf
Polish	BASE_RESOURCE_BUNDLE_NAME_pl.xlf
Portuguese	BASE_RESOURCE_BUNDLE_NAME_pt.xlf
Romanian	BASE_RESOURCE_BUNDLE_NAME_ro.xlf
Russian	BASE_RESOURCE_BUNDLE_NAME_ru.xlf
Simplified Chinese	BASE_RESOURCE_BUNDLE_NAME_zh_CN.xlf
Slovak	BASE_RESOURCE_BUNDLE_NAME_sk.xlf
Spanish	BASE_RESOURCE_BUNDLE_NAME_es.xlf
Swedish	BASE_RESOURCE_BUNDLE_NAME_sv.xlf
Traditional Chinese	BASE_RESOURCE_BUNDLE_NAME_zh_TW.xlf
Turkish	BASE_RESOURCE_BUNDLE_NAME_tr.xlf
Thai	BASE_RESOURCE_BUNDLE_NAME_th.xlf

6.8 Localizable MAF Properties

The `maf-application.xml` and `maf-feature.xml` files both expose properties that can reference text resources in resource bundles. Table 6–2 and Table 6–3 list these properties. Because these configuration files are read early in the application lifecycle, these strings are not evaluated as EL statements at runtime. Instead, these strings are taken as the full key for the translated string in the native device translation infrastructure.

Table 6–4 lists the attributes of those MAF AMX UI components that can reference text resources.

At the application level, you can localize strings for such attributes as application name or preference page labels, which are listed in Table 6–2.

Table 6–2 Localizable MAF Application Attributes

Element	Attribute(s)
`<adfmf:Application>`	`name`
`<adfmf:PreferenceGroup>`	`label`
`<adfmf:PreferencePage>`	`label`
`<adfmf:PreferenceBoolean>`	`label`
`<adfmf:PreferenceText>`	`label`
`<adfmf:PreferenceNumber>`	`label`
`<adfmf:PreferenceList>`	`label`
`<adfmf:PreferenceValue>`	`name`

At the project (view controller) level, you can use the MAF Feature editor to localize application feature-related attributes listed in Table 6–3.

Table 6–3 Localizable Application Feature Attributes

Element	Attribute(s)
<adfmf:Feature>	name
<adfmf:Constraint>	value
<adfmf:Parameter>	value
<adfmf:PreferencePage>	label
<adfmf:PrefrenceGroup>	label
<adfmf:PreferenceBoolean>	label
<adfmf:PreferenceText>	label
<adfmf:PreferenceNumber>	label
<adfmf:PreferenceList>	label
<adfmf:PreferenceValue>	name

You can create resource bundles for attributes of such MAF AMX UI components as the text attribute of the Button component (<amx:commandButton>). Figure 6–4 lists these MAF AMX UI components.

Table 6–4 Localizable Attributes of MAF AMX UI Components

Component	Attribute(s)
<amx:inputDate>	label
<amx:inputNumberSlider>	label
<amx:panelLabelAndMessage>	label
<amx:selectBooleanCheckBox>	label
<amx:selectBooleanSwitch>	label
<amx:selectItem>	label
<amx:selectManyCheckBox>	label
<amx:selectManyChoice>	label
<amx:selectOneButton>	label
<amx:selectOneChoice>	label
<amx:selectOneRadio>	label
<amx:commandButton>	text
<amx:commandLink>	text
<amx:goLink>	text
<amx:inputText>	label, value, hintText
<amx:outputText>	value

7 Skinning MAF Applications

This chapter describes how to customize the appearance of a MAF application by using skins.

This chapter includes the following sections:

- Section 7.1, "Introduction to MAF Application Skins"
- Section 7.2, "Adding a Custom Skin to an Application"
- Section 7.3, "Specifying a Skin for an Application to Use"
- Section 7.4, "Registering a Custom Skin"
- Section 7.5, "Versioning MAF Skins"
- Section 7.6, "What Happens When You Version Skins"
- Section 7.7, "Overriding the Default Skin Styles"
- Section 7.8, "What You May Need to Know About Skinning"
- Section 7.9, "Adding a New Style Sheet to a Skin"
- Section 7.10, "Enabling End Users Change an Application's Skin at Runtime"
- Section 7.11, "What Happens at Runtime: How End Users Change an Application's Skin"

7.1 Introduction to MAF Application Skins

MAF uses cascading style sheet (CSS) language-based skins to make sure that all application components within a MAF application (including those used in its constituent application features) share a consistent look and feel. Rather than change how a MAF application looks by re-configuring MAF AMX or HTML components, you can create, or extend, a skin that changes how components display.

Creating or editing a skin to change the look and feel of your MAF application is an iterative process. You can create a skin and deploy it to a device to view the result. You can continue this process until your skin renders the result that you want. The developer tools that the Android and iOS platforms provide to inspect and debug user interface code like CSS, HTML, and JavaScript are an invaluable resource for this task. For more information about how you can use these tools with your MAF application, see Section 30.4.2, "How to Debug UI Code on the Android Platform" and Section 30.5.2, "How to Debug UI Code on the iOS Platform."

The following are the supported skin families and versions that MAF uses to define the selectors that determine the appearance of MAF AMX pages:

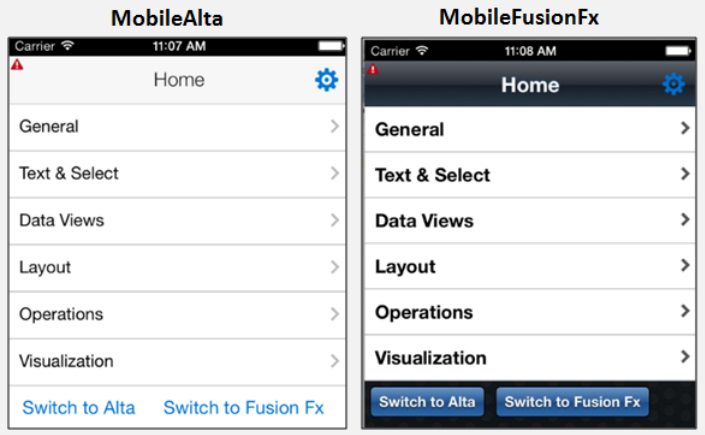
`amx`

```
mobileAlta-1.0
  mobileAlta-1.1
    mobileAlta-1.2
       mobileAlta-1.3
          mobileAlta-1.4
             mobileAlta-1.5
```

By default, a new MAF application that you create uses the latest version of the `mobileAlta` skin family. An application that you migrate from a previous release to the current release continues to use the skin that it was configured to use prior to migration. If you want the migrated application to use another skin (for example, the latest version of `mobileAlta`), you need to edit the `maf-config.xml` file, as described in Section 7.3, "Specifying a Skin for an Application to Use."

Figure 7–1 demonstrates the difference in look and feel between the `mobileAlta` and `mobileFusionFx` skin families by showing the same application screen rendering using the different skins.

Figure 7–1 Comparison of Look and Feel Provided by mobileAlta and mobileFusionFx



You can view all the resources (CSS files and images) that the skin for your MAF application uses by deploying your MAF application to a device, emulator or simulator. Deployment moves these resources to a `www\css` directory that it creates within the platform-specific artifacts that the deployment process generates. For iOS deployments, the `www\css` directory is located within the `temporary_xcode_project` directory. The iOS deployment packages these resources into an `Oracle_ADFmc_Container_Template.zip` file that is added to the created `.IPA` file. For Android deployments, the directory path is `%app%\deploy\Android1\framework\build\java_res\assets\www\css` where `Android1` is the name of the deployment profile. Android deployment packages these resources into an assets.zip file that is added to the created `.APK` file. Note that OEPE's **Project > Clean >Clean All** command removes the deploy directory and its sub-directories, including the `www\css` directory.

> **Caution:** Do not write styles that rely on the MAF DOM structures. Further, some of the selectors defined in these files may not be supported.

You use the `maf-config.xml` file, described in Section 7.1.1, "About the maf-config.xml File," and the `maf-skins.xml` file, described in Section 7.1.2, "About the maf-skins.xml File," to control the skinning of the MAF application. The `maf-config.xml` file

designates the default skin family used to render application components and the `maf-skins.xml` file enables you to customize the default skin family or to define a new skin family.

7.1.1 About the maf-config.xml File

After you create a MAF application, OEPE populates the `maf-config.xml` file to the MAF application's **META-INF** node. The file itself is populated with the base MAF skin family, `mobileAlta`, illustrated below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-family>mobileAlta</skin-family>
  <skin-version>v1.3</skin-version>
  ....
</adfmf-config>
```

> **Note:** You can determine the skin value at runtime using EL expressions. For more information, see Section 7.10, "Enabling End Users Change an Application's Skin at Runtime."
>
> If you do not specify values for the `<skin-family>` or `<skin-version>` tags, the MAF application automatically uses the latest skin family or skin version.

MAF applies skins as a hierarchy, with device-specific skins being applied first, followed by platform-specific skins, and then the base skin, `mobileAlta`. In terms of MAF's `mobileAlta` skin family, this hierarchy is expressed as follows:

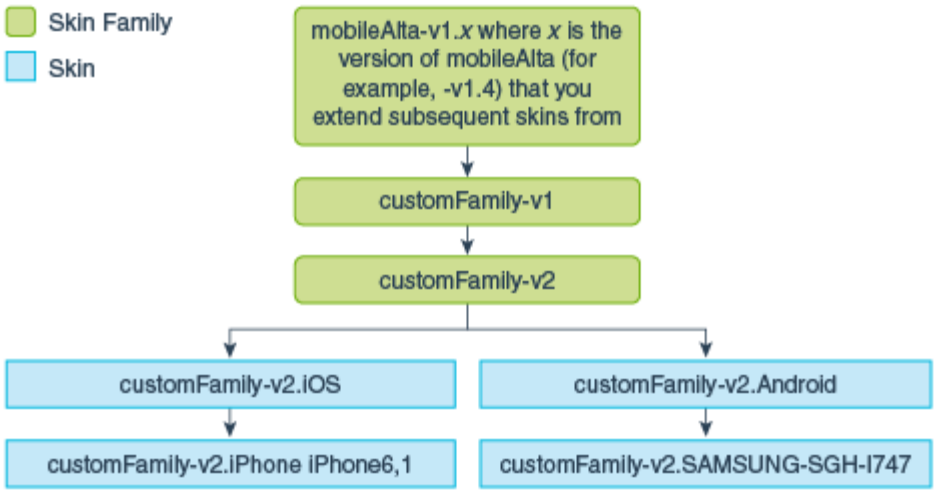
1. `mobileAlta.<DeviceModel>` (for example, `mobileAlta.iPhone5,3`)
2. `mobileAlta.iOS` or `mobileAlta.Android`
3. `mobileAlta`

> **Tip:** Deploy the DeviceDemo sample application to the device or platform you want to retrieve the values for if you plan to create a device-specific or platform-specific skin. The Properties application feature in the DeviceDemo sample application displays the values for the device model and platform the application runs on. For more information, see Appendix E, "MAF Sample Applications."

Figure 7–2 provides a visual illustration of how MAF applies this hierarchy of skins at runtime. Note also that the `SkinningDemo` sample application, described in Appendix E, "MAF Sample Applications," demonstrates this implementation.

MAF gives precedence to selectors defined at the device-specific level of this hierarchy. In other words, MAF overwrites a selector defined in `mobileAlta.iOS` with the `mobileAlta.iPhone` definition for the same selector. The `<extends>` element, described in Section 7.1.2, "About the maf-skins.xml File," defines this hierarchy for the MAF runtime. For more information on how skins are applied at various levels, see Section 7.8, "What You May Need to Know About Skinning."

Figure 7–2 MAF Skin Hierarchy Application at Runtime



7.1.2 About the maf-skins.xml File

The `maf-skins.xml` file located in the **META-INF** node of the application controller project allows you to either define a new skin by extending an existing skin, or, add a new style sheet to an existing skin.

By default, this file is empty, but the elements listed in Table 7–1 describe the child elements that you can use to populate this file to extend `mobileAlta` or to define the CSS files that are available to the application. You use the `<skin>` element to create new skins or to extend an existing skin.

Table 7–1 Child Elements of the <skin> Element

Elements	Description
`<id>`	A required element that identifies the skin in the `maf-skins.xml` file. The value you specify must adhere to one of the following formats: ▪ skinFamily-version ▪ skinFamily-version.platform For example, specify `mySkin-v1.iOS` if you want to register a skin for your application that defines the appearance of your application when deployed to an Apple iPad or iPhone. Substitute `iOS` by `iPad` or `iPhone` if the skin that you register defines the appearance of your application on one or other of the latter devices. Specify `.android` if you want to register a skin that defines the appearance of your application when deployed to the Android platform.
`<family>`	A required element that identifies the skin family.

Table 7–1 (Cont.) Child Elements of the <skin> Element

Elements	Description
<extends>	Use this element to extend an existing skin by specifying the skin id of the skin you want to extend. <skin> <id>mySkin-v1</id> <family>mySkin</family> <extends>mobileAlta-v1.4</extends> <style-sheet-name>styles/myskin.css</style-sheet-name> <version> <name>v1</name> </version> </skin>
<style-sheet-name>	Use a relative URL to specify the location of the CSS file within your MAF application's project. For example, the maf-skins.xml file in the SkinningDemo sample application contains the following reference to the v1.css style sheet in the css directory of the application controller project: <style-sheet-name>css/v1.css</style-sheet-name>
<version>	Specify different versions of a skin. For more information, see Section 7.5, "Versioning MAF Skins."

Table 7–2 lists elements that you can use to define the <skin-addition> element in a MAF CSS when you integrate a style sheet into an existing skin.

Table 7–2 The <skin-addition> Child Elements

Element	Description
<skin-id>	Specify the ID of the skin that you need to add an additional style sheet to. Possible values include the skins provided by MAF (for example, mobileAlta-v1.4.iOS) or a custom skin that you create.
<style-sheet-name>	Use a relative URL to specify the location of the CSS file within your MAF application's project. For example, the maf-skins.xml file in the SkinningDemo sample application contains the following reference to the v1.css style sheet in the css directory of the application controller project: <style-sheet-name>css/v1.css</style-sheet-name>

The example below illustrates designating the location of the CSS file in the <style-sheet-name> element and the target skin family in <skin-id>.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-addition>
    <skin-id>mobileAlta-v1.4.iOS</skin-id>
    <style-sheet-name>skins/mystyles.iphone.addition1.css</style-sheet-name>
  </skin-addition>
</adfmf-skins>
```

You can use the <skin-id> and <style-sheet-name> elements to render to a particular iOS or Android device, or alternatively, you can define these elements to handle the styling for all of the devices of a platform. Table 7–3 provides examples of using these elements to target all of the devices belonging to the iOS platform, as well as specific iOS device types (tablets, phones, and simulators).

Tip: Consider using the DeviceDemo sample application, described in Appendix E, "MAF Sample Applications," to retrieve information about the device model.

Table 7–3 Platform- and Device-Specific Styling

Device	Example

iPhone	`<skin-addition>`
	`<skin-id>mobileAlta-v1.4.iPhone5,1</skin-id>`
	`<style-sheet-name>iPhoneStylesheet.css</style-sheet-name>`
	`</skin-addition>`

iPad	`<skin-addition>`
	`<skin-id>mobileAlta-v1.4.iPad4,2</skin-id>`
	`<style-sheet-name>iPadStylesheet.css</style-sheet-name>`
	`</skin-addition>`

iPhone Simulator	`<skin-addition>`
	`<skin-id>mobileAlta-v1.4.iPhone Simulator x86_64</skin-id>`
	`<style-sheet-name>iPhoneSimStylesheet.css</style-sheet-name>`
	`</skin-addition>`

All iOS Devices	`<skin-addition>`
	`<skin-id>mobileAlta-v1.4.iOS</skin-id>`
	`<style-sheet-name>iOSSimStylesheet.css</style-sheet-name>`
	`</skin-addition>`

7.2 Adding a Custom Skin to an Application

To add a custom skin to your application, create a CSS file within OEPE which places the CSS in a project's source file for deployment with the application.

To add a custom skin to an application:

1. In the Applications window, expand **ViewContent** and right-click the **css** folder and choose **New > Other**. In the New Gallery dialog, expand **Web** and choose **CSS File**. Click **Next**.
2. In the New CSS File page of the wizard, specify a name for the CSS file.
3. Click **Finish**.

The new CSS file opens in Eclipse, where you can define styles for your application.

7.3 Specifying a Skin for an Application to Use

You configure values in the `maf-config.xml` file that determine what skin the application uses.

To specify a skin for an application to use:

1. In the Project Explorer, expand the assembly project, then `adf`, then `META-INF`, then double-click `maf-config.xml` file to open it in the XML Editor.
2. In the `maf-config.xml` file, specify the value of the `<skin-family>` element for the skin you want to use and, optionally, the `<skin-version>` element, as shown in the example below.

To see the file in the Source Editor, click the Source tab at the bottom of the XML Editor. The example below shows the configuration required to make a mobile application use the `mobileAlta-v1.4` skin.

```
<adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-family>mobileAlta</skin-family>
  <skin-version>v1.2</skin-version>
</adfmf-config>
```

Note: Set an EL expression as the value for the `<skin-family>` element if you want to dynamically select the skin the application uses at runtime. For more information, see Section 7.10, "Enabling End Users Change an Application's Skin at Runtime."

7.4 Registering a Custom Skin

You register a custom skin by adding the property values to the `maf-skins.xml` file that identify the custom skin to your application.

To register a custom skin:

1. In the Project Explorer, expand **Application Project** > **src** > **META-INF** and double-click **maf-skins.xml** to open it in the XML Editor.
2. Switch to the Design tab. Select and right click **adfmf-skins** tag. Select **Add Child > skin**.
3. Expand the skin node and notice that id and family tags are created by default. You can right click and select **Add Child > extends** or **Add Child > style-sheet-name** to create `extends` and `style-sheet-name` respectively.
4. Select any of the following fields and go to the Content column in order to edit the values as follows:
   - **family**—Enter a value for the family name of your skin.

     You can enter a new name or specify an existing family name. If you specify an existing family name, you need to version skins, as described in Section 7.5, "Versioning MAF Skins," to distinguish between skins that have the same value for family.

     The value you enter is set as the value for a `<family>` element in the `maf-skins.xml` where you register the skin that you create. At runtime, the `<skin-family>` element in the application's `maf-config.xml` uses this value to identify the skin that an application uses.
   - **id**—Enter an ID for the skin that uses one of the following naming formats: `skinFamily-version` or `skinFamily-version.platform`. For example, `mySkinFamily-v1.2.android`.
   - **extends**—Enter the name of the parent skin that you want to extend. For example, if you want your custom skin to extend the `mobileAlta-v1.4` skin, enter `mobileAlta-v1.4`.
   - **style-sheet-name**—Enter or select the name of the style sheet.
5. Save the file `maf-skins.xml`.

7.5 Versioning MAF Skins

You can specify version numbers for your skins in the `maf-skins.xml` file using the `<version>` element. Use this optional capability if you want to distinguish between skins that have the same value for the `<family>` element in the `maf-skins.xml` file. This capability is useful in scenarios where you want to create a new version of an existing skin in order to change some existing behavior. Note that when you configure an application to use a particular skin, you do so by specifying values in the `maf-config.xml` file, as described in section Section 7.3, "Specifying a Skin for an Application to Use."

You specify a version for your skin by entering a value for the `<version>` element in the `maf-skins.xml` file.

> **Best Practice:** Specify version information for each skin that you register in the application's `maf-skins.xml` file.

To version a MAF skin:

1. In the Project Explorer, expand the **application project > src > META-INF**, then double-click `maf-skins.xml` to open it in the XML Editor.
2. Switch to the Design tab. Select and right click skin tag. Select **Add Child > version**.
3. Right-click on **version** and select **Add Child > default**. Select default tag, go to the Content column and set the value of default as `true` if you want your application to use this version of the skin when no value is specified in the `<skin-version>` element of the `maf-config.xml` file, as described in Section 7.5, "Versioning MAF Skins."
4. Select name tag under skin tag and go to the Content column. Enter the value of the name field. For example, enter `v1` if this is the first version of the skin.
5. Save the file `maf-skins.xml`.

7.6 What Happens When You Version Skins

The version information that you configure for skins takes precedence over platform and device values when an application applies a skin at runtime. At runtime, a MAF application applies a device-specific skin before it applies a platform-specific skin. If skin version information is specified, the application first searches for a skin that matches the specified skin version value. If the application finds a skin that matches the skin version and device values, it applies this skin. If the application cannot find a skin with the specified skin version in the device-specific skins, it searches for a skin with the specified version in the platform-specific skins. If it does not find a skin that matches the specified version in the available platform-specific skins, it searches the base skins.

The example below shows an example `maf-skins.xml` that references three skins (`customFamily-v1.iphone5,3`, `customFamily-v2.iPhone5,3` and `customFamily-v3.iPhone5,3`). Each of these skins have the same value for the `<family>` element (`customFamily`). The values for the child elements of the `<version>` elements distinguish between each of these skins.

At runtime, an application that specifies `customFamily` as the value for the `<skin-family>` element in the application's `maf-config.xml` file uses `customFamily-v1.iphone5,3` because this skin is configured as the default skin in the `maf-skins.xml` file (`<default>true</default>`). You can override this behavior by

specifying a value for the <skin-version> element in the maf-config.xml file, as described in Section 7.3, "Specifying a Skin for an Application to Use." For example, if you specify v2 as a value for the <skin-version> element in the maf-config.xml file, the application uses customFamily-v2.iPhone instead of customFamily-v1.iphone5,3 that is defined as the default in the maf-skins.xml file.

If you do not specify the skin version to pick (using the <skin-version> element in the maf-config.xml file), then the application uses the skin that is defined as the default using the <default>true</default> element in the maf-skins.xml file. If you do not specify a default skin, the application uses the last skin defined in the maf-skins.xml file. In the example below, the last skin to be defined is customFamily-v3.iPhone.

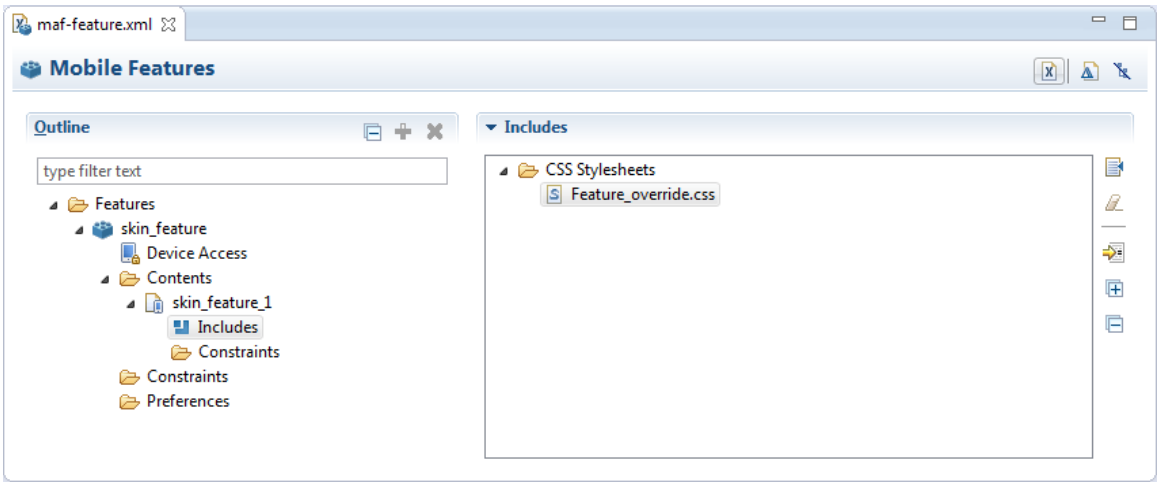
```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/skin">
  <skin id="s1">
    <family>customFamily</family>
    <id>customFamily-v1.iphone5,3</id>
    <extends>customFamily-v1.iOS</extends>
    <style-sheet-name>iphone.css</style-sheet-name>
    <version>
      <default>true</default>
      <name>v1</name>
    </version>
  </skin>
  <skin id="s2">
    <family>customFamily</family>
    <id>customFamily-v2.iPhone</id>
    <extends>customFamily-v1.iOS</extends>
    <style-sheet-name>iphone-v2.css</style-sheet-name>
    <version>
      <name>v2</name>
    </version>
  </skin>
  <skin id="s3">
    <family>customFamily</family>
    <id>customFamily-v3.iPhone</id>
    <extends>customFamily-v1.iOS</extends>
    <style-sheet-name>iphone-v3.css</style-sheet-name>
    <version>
      <name>v3</name>
    </version>
  </skin>
</adfmf-skins>
```

7.7 Overriding the Default Skin Styles

For a MAF AMX application, you can designate a specific style for the application feature implemented as MAF AMX, thereby overriding the default skin styles set at the application-level within the maf-config.xml and maf-skins.xml files. You add individual styles to the application feature using a CSS file as the *Includes* file.

The Includes table in the overview editor for the maf-feature.xml file enables you to add a CSS to a MAF AMX application feature.

Figure 7–3 The Includes Section



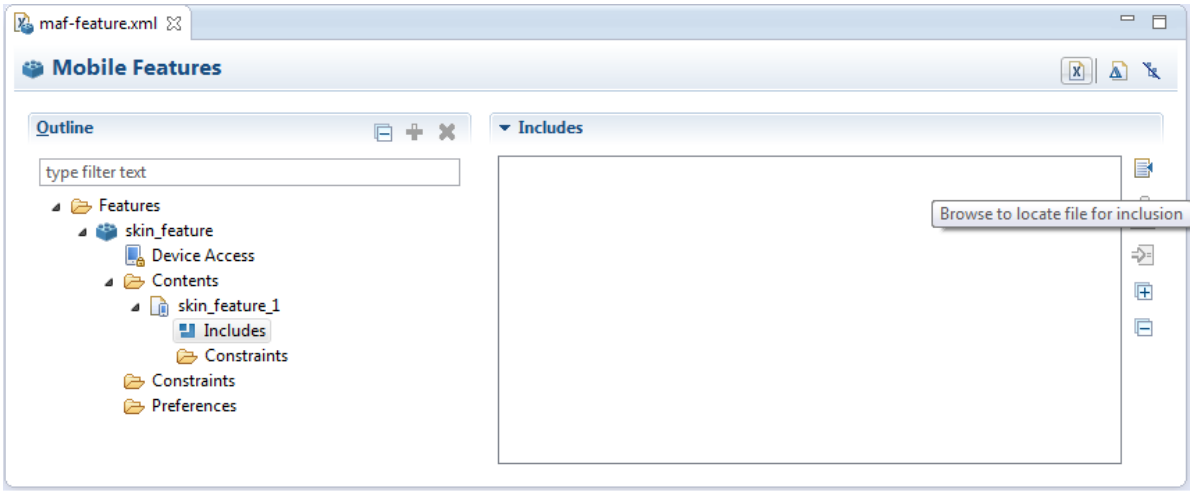
Before you begin:

Create a MAF task flow as described in Section 11.2, "Creating Task Flows." Create or add a Cascading Style Sheet (CSS) file for the skin. You can create the CSS file by right clicking the **ViewContent** folder of the view controller project, and then select **File > New > Other** and type CSS in the filter. Then, select **Web > CSS File**.

How to add a style to an application feature:

1. Open the MAF Features Editor. Ensure that there is a feature that has **AMX Page** as the content type.
2. In the Outline section, expand **Features > feature name > Contents > Feature Id** and select **Includes**. On selection, the Includes section will appear on the right side of the editor, as shown in Figure 7–4.

Figure 7–4 Selecting the StyleSheet Option



3. Select the **Browse** button to select a file for inclusion in the Includes section. This opens the **Select CSS Stylesheet or JavaScript File** dialog, as shown in Figure 7–5.
4. Select the CSS file that you created earlier under the **ViewContent** directory of the view controller project.

Note: The `.css` file must reside within the view controller project; you cannot include a `.css` file that resides in the application controller project.

Figure 7–5 Selecting the CSS for the Application Feature

5. Save the changes in the Mobile Features Editor.

7.8 What You May Need to Know About Skinning

The CSS files defined in the `maf-skins.xml` file, illustrated in the example below, show how to extend a skin to accommodate the different display requirements of the Apple iPhone and iPad. These styles are applied in a descending fashion. The SkinningDemo sample application provides a demonstration of how customized styles can be applied when the application is deployed to different devices. You can find this example by selecting **File > New > MAF Examples**, then selecting **SkinningDemo** from the MAF Examples page.

For example, at the iOS level, the stylesheet (`mobileAlta` in the example below) is applied to both an iPhone or an iPad. For device-specific styling, define the `<skin-id>` elements for the iPhone and iPad skins. The skinning demo application illustrates the use of custom skins defined through this element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/skins">
    <skin>
        <id>mobileAlta-v1.4.iPhone5,3</id>
        <family>mobileAlta</family>
        <extends>mobileAlta-v1.4.iOS</extends>
```

```
        <style-sheet-name>skins/mobileAlta-v1.4.iphone.css</style-sheet-name>
    </skin>
    <skin>
        <id>mobileAlta-v1.4.iPad iPad4,1</id>
        <family>mobileAlta</family>
        <extends>mobileAlta-v1.4.iOS</extends>
        <style-sheet-name>skins/mobileAlta-v1.4.ipad.css</style-sheet-name>
    </skin>
    <!--  Skin Additions -->
    <skin-addition>
        <skin-id>mobileAlta-v1.4.iPhone5,3</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition1.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileAlta-v1.4.iPhone5,3</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition2.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileAlta-v1.4.iOS</skin-id>
        <style-sheet-name>skins/mystyles.ios.addition2.css</style-sheet-name>
    </skin-addition>
</adfmf-skins>
```

7.9 Adding a New Style Sheet to a Skin

You can add a CSS file to an existing skin instead of extending a skin.

To add a new style sheet to a skin

1. In the Project Explorer, expand the **application project > src, > META-INF** folders, then double-click the file `maf-skins.xml` to open it in the XML Editor.
2. Switch to the Design tab. Select and right-click the `adfmf-skins` tag. **Select Add Child > skin-addition**.
   - Select **skin-id** tag under **skin-addition** tag. Go to the Content column and enter the identifier of the skin to which you want to add a new style (for example, `mobileAlta-v1.4`).
   - Right click **skin-addition tag** and select **Add Child > style-sheet-name**. Select the style-sheet-name tag and go to the Content column. Specify the path of the css file relative to the ViewContent folder. For example, if you created the css file at `ViewContent/css/myaddedcss.css`, then enter `css/myaddedcss.css` as the value
3. Save the file `maf-skins.xml`.

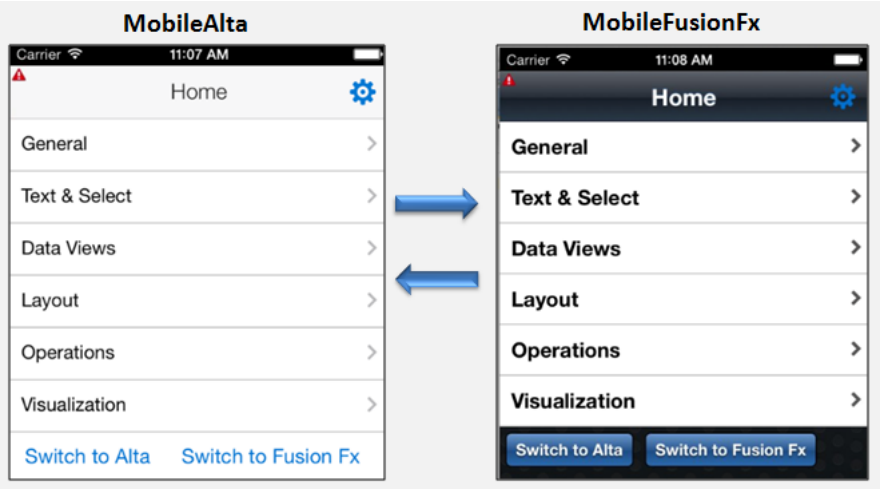
Caution: Creating custom styles that use DOM-altering structures can cause MAF applications to hang. Specifically, the `display` property causes rendering problems in the HTML that is converted from MAF AMX. This property, which uses such values as `table`, `table-row`, and `table-cell` to convert components into a table, may result in table-related structures that are not contained within the appropriate parent table objects. Although this problem may not be visible within the application user interface itself, the logging console reports it through a Signal 10 exception.

7.10 Enabling End Users Change an Application's Skin at Runtime

You can configure your application to enable end users select an alternative skin at runtime. You might configure this functionality when you want end users to render the application using a skin that is more suitable for their needs.

Figure 7–6 shows how you might implement this functionality by displaying buttons to allow end users to change the skin the application uses at runtime. Configure the buttons on the page to set a scope value that can later be evaluated by the `skin-family` property in the application's Mobile Application Editor.

Figure 7–6 Changing an Application's Skin at Runtime (on iOS)



You enable end users change an application's skin by exposing a component that allows them to update the value of the `skin-family` property in the application's Mobile Application Editor.

To enable end users change an application's skin at runtime:

1. Open the page where you want to configure the component(s) that you use to set the skin family property in the Mobile Application Editor.
2. Configure a number of components (for example, button components) that allow end users to choose one of a number of available skins at runtime, as shown in Figure 7–6.

   The example below shows how you configure `amx:commandButton` components that allow end users to choose available skins at runtime, as shown in Figure 7–6. Each `amx:commandButton` component specifies a value for the `actionListener` attribute. This attribute passes an `actionEvent` to a method (`skinMenuAction`) on a managed bean named `skins` if an end user clicks the button.

   ```
   ...
   <amx:commandButton text="Switch to Alta"
        actionListener="#{applicationScope.SkinBean.switchToMobileAlta}"
   id="cb1"/>
   <amx:commandButton text="Switch to Fusion Fx"
        actionListener="#{applicationScope.SkinBean.switchToMobileFusionFx}"
   id="cb2"/>
   ...
   ```

3. Write a managed bean in the application's view controller project to store the value of the skin selected by the end user. Example 7–1 shows a method that takes the

value the end user selected and uses it to set the value of `skinFamily` in the managed bean. Example 7–1 also shows a method that resets all features in the application to use the new skin. Example 7–1 also makes use of the `PropertyChangeSupport` and `PropertyChangeListener` objects described in Section 13.12, "About Data Change Events."

4. In the Project Explorer, expand the **Application Resources** panel, expand **Descriptors > ADF Meta-INF** node and double-click the `maf.config.xml` file.
5. In the `maf-config.xml` file, write an EL expression to dynamically evaluate the skin family:

```
<skin-family>#{applicationScope.SkinBean.skinFamily}</skin-family>
```

Example 7–1 Managed Bean to Change an Application's Skin

```
package application;

import javax.el.ValueExpression;
import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.FeatureInformation;
import oracle.adfmf.framework.api.AdfmfContainerUtilities;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;

public class SkinBean {

    private String skinFamily = "mobileAlta";
    private PropertyChangeSupport propertyChangeSupport = new
PropertyChangeSupport(this);

    public void setSkinFamily(String skinFamily) {
        String oldSkinFamily = this.skinFamily;
        this.skinFamily = skinFamily;
        propertyChangeSupport.firePropertyChange("skinFamily", oldSkinFamily,
skinFamily);
    }

    public String getSkinFamily() {
        return skinFamily;
    }

    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public void switchToMobileAlta(ActionEvent ev){
        this.switchSkinFamily("mobileAlta");
    }

    public void switchToMobileFusionFx(ActionEvent ev) {
        this.switchSkinFamily("mobileFusionFx");
    }

    public void switchSkinFamily(String family) {
        this.setSkinFamily(family);
```

```
            // reset all the features individually as follows to load the new skin
            FeatureInformation[] features = AdfmfContainerUtilities.getFeatures();
            for (int i = 0; i < features.length; i++) {
                AdfmfContainerUtilities.resetFeature(features[i].getId());
            }
        }
    }
```

7.11 What Happens at Runtime: How End Users Change an Application's Skin

At runtime, the end user uses the component that you exposed to select another skin. This component submits the value that the end user selected to a managed bean that, in turn, sets the value of a managed bean property (**skinFamily**). At runtime, the **<skin-family>** property in the **maf-config.xml** file reads the value from the managed bean using an EL expression. The managed bean in Example 7–1 also reloads the features in the application to use the newly-specified skin.

> **Tip:** Similar to the `<skin-family>` property, you can use an EL expression to set the value of the `<skin-version>` property in the `maf-config.xml` file at runtime.

8

Reusing the MAF Application Content

This chapter introduces Feature Archive (FAR) files and describes how you can package application feature content into these files for reuse in one or more MAF applications.

This chapter includes the following sections:

- Section 8.1, "Introduction to Feature Archive Files"
- Section 8.2, "Using FAR Content in a MAF Application"
- Section 8.3, "What Happens When You Add a FAR"
- Section 8.4, "What You May Need to Know About Enabling the Reuse of Feature Archive Resources"

8.1 Introduction to Feature Archive Files

Application features, when packaged into a JAR file known as a Feature Archive file (FAR), provide reusable content that can be consumed by other MAF applications. A MAF application can consume one or more FAR files. A FAR file contains everything that an application feature requires, such as icon images, resource bundles, HTML files, JavaScript files, and other implementation-specific files.

A FAR also contains one `maf-feature.xml` file, which identifies each of the packaged application features by a unique ID. You can edit this file to update application feature properties, such as content implementation (MAF AMX, Local HTML, Remote URL), display properties based on such factors as user roles and privileges, or device properties.

You can add a FAR as either an application library or as a view controller project. You cannot customize the FAR's contents when you add it as project library, nor can you reuse its individual artifacts. A MAF application consumes the FAR in its entirety when it is added as a library file. For example, a FAR's task flow cannot be the target of a task flow call activity.

8.2 Using FAR Content in a MAF Application

You can import a FAR to make its features accessible to your MAF application.

Before you begin:

Deploy the application feature as a Feature Archive file, as described in Section 27.6.2, "How to Deploy the Feature Archive.".

How to add application feature content to a MAF application:

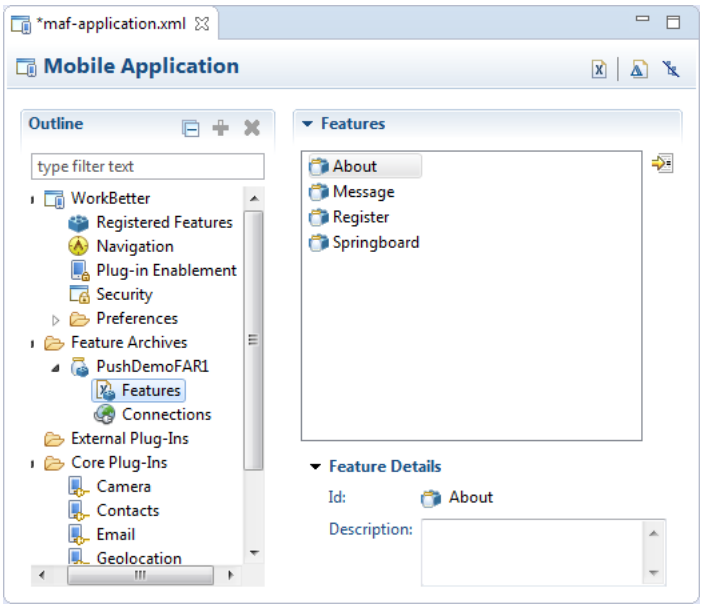
1. In the Project Explorer, expand the folder for the top-level assembly project, then expand the MAF folder.
2. Double-click MAF Application Editor to open it.
3. Right-click on the Feature Archives folder and choose **New > Feature Archive**.
4. Click the Browse button for the **URI** field to open the Mobile Feature Archive Location dialog, then type in the URI at which the FAR can be located. OEPE verifies that the selected file contains feature definitions, and that the location is accessible.
5. When you have specified the URI and OEPE has verified it, click **OK**.

8.3 What Happens When You Add a FAR

After you add a FAR to a MAF application:

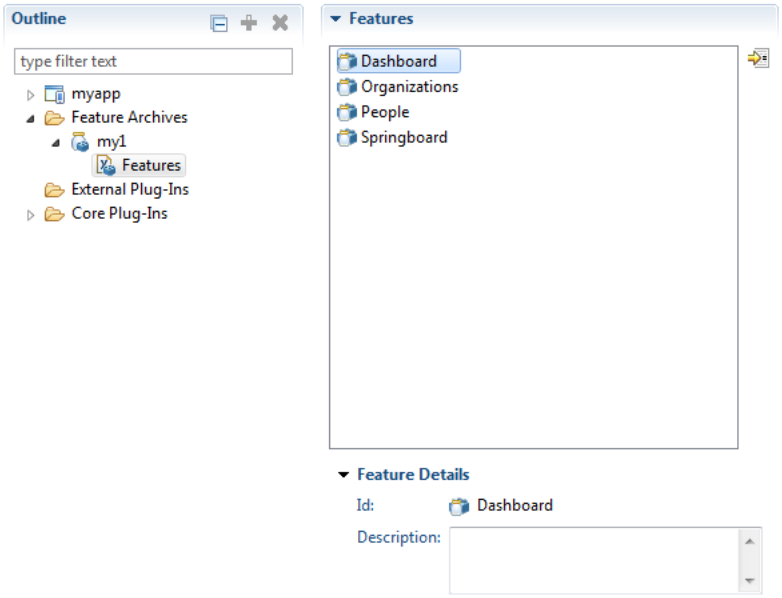
- The FAR appears in the MAF Application Editor, as shown in Figure 8–1.

Figure 8–1 Feature Archive Imported into MAF Application



- Every application feature declared in the `maf-feature.xml` files included in the JARs becomes available to the consuming application, as illustrated by Figure 8–2 where the dropdown lists IDs of the available application features in the JAR in addition to one that has already been defined in the application.

Figure 8–2 FAR Features Available



- The information in the `connections.xml` file located in the Feature Archive JAR is merged into the consuming application's `connections.xml file`. If there are any conflicts, these are displayed in the Log window.

8.4 What You May Need to Know About Enabling the Reuse of Feature Archive Resources

To ensure that the resources of a FAR can be used by an application, both the name of the FAR and its feature reference IDs must be globally unique; ensure there are no duplicate feature reference IDs in the `maf-application.xml` file. Within the FAR itself, the `DataControl.dcx` file must be in a unique package directory. Rather than accepting the default names for these package directories, you should instead create a unique package hierarchy for the project. You should likewise use a similar package naming system for the feature reference IDs.

9

Using Plugins in MAF Applications

This chapter describes how to enable the core plugins that MAF provides for use in MAF applications, how to register additional plugins, how to import a plugin from a FAR, and how to package plugins in your MAF application for deployment.

This chapter includes the following sections:

- Section 9.1, "Introduction to Using Plugins in MAF Applications"
- Section 9.2, "Enabling a Core Plugin in Your MAF Application"
- Section 9.3, "Registering Additional Plugins in Your MAF Application"
- Section 9.4, "Deploying Plugins with Your MAF Application"
- Section 9.5, "Importing Plugins from a Feature Archive File"
- Section 9.6, "Using a Plugin in a MAF Application"

9.1 Introduction to Using Plugins in MAF Applications

MAF packages a number of Cordova plugins. A MAF application uses these plugins to interact with the device on which it is deployed. Core plugins are the plugins that MAF provides by default. View these plugins in the MAF Application Editor. Examples include the Email and Contacts plugins that MAF applications use to access email and contact functionality from a device.

View the Cordova versions used by the Android, iOS, and Windows platforms by selecting **External Plug-ins** in the MAF Application Editor.

Select a plugin in the Core Plugins list, as shown in Figure 9–1, to view a description of the individual plugins. By default, a newly-created MAF application enables only one core plugin (Network Information plugin). You enable these core plugins, as described in Section 9.2, "Enabling a Core Plugin in Your MAF Application."

> **Note:** All applications on iOS devices have network access by default. You cannot change this behavior. If an application that is deployed to an Android device does not require network access, disable the Network Information plugin. The Network Information plugin must be enabled to facilitate remote debugging of an application running on an Android emulator or device.

You can register additional plugins if the core plugins that MAF provides by default do not meet the requirements of your MAF application. For more information, see "Introduction to custom Cordova plugin development" at
http://blogs.oracle.com/mobile/entry/introduction_to_custom_cordova_plugin

and Section 9.3, "Registering Additional Plugins in Your MAF Application." Once you have either enabled the core plugin or registered any additional plugins for your MAF application, you create content in an application feature that accesses the functionality of the plugin. For more information, see Section 9.6, "Using a Plugin in a MAF Application."

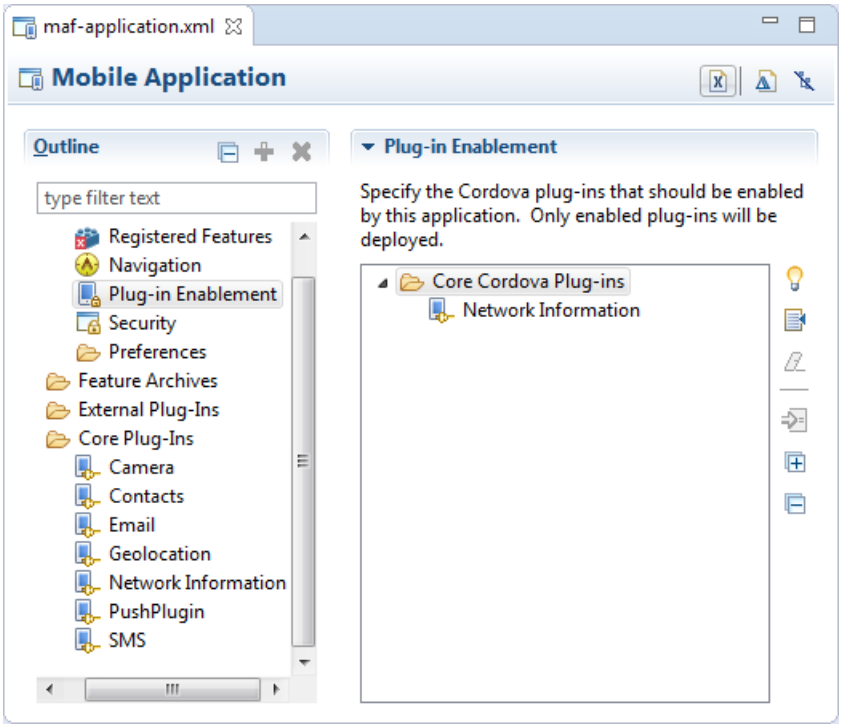
The deployment of a MAF application may fail, after the registration of additional plugins, for the following reasons:

- Filename conflicts between plugins that your MAF application uses.
- The additional plugins that were registered require dependent plugins for proper functioning.

For more information, see Deploying Plugins with Your MAF Application Section 9.4, "Deploying Plugins with Your MAF Application."

If you want to migrate a MAF application created with an earlier release of MAF, you need to register any plugins your application uses in the MAF Application Editor. For more information, refer to "Migrating Plugins from Earlier Releases." section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

Figure 9–1 Plugins in the MAF Application Editor



9.2 Enabling a Core Plugin in Your MAF Application

By default, newly-created MAF applications enables only one core plugin (Network Information plugin). Enable or disable additional core plugins so that your MAF application can access the associated device functionality.

9.2.1 How to Enable a Core Plugin in Your MAF Application

You enable a core plugin using the MAF Application Editor.

To enable a core plugin in your MAF application:

1. From the Project Explorer, expand the assembly project folder, then expand MAF and double-click MAF Application Editor.
2. In the editor under Outline, expand **Core plugins** and then select **Plugin Enablement**. The Plug -ins already enabled for the application are listed.
3. Click to open the Mobile Plugin Selection dialog and select the core plugin you want to use. Click **OK**.

   For example, if you want your MAF application to be able to send an SMS message, select the SMS plugin.
4. With the core plugin selected in the Plugin Enablement area of the MAF Application Editor, choose whether the plugin is to be available for Android, iOS, or both. Save your changes.

   You can also add notes, for example, if there are plugins which are enabled but which do not have features associated with them.

9.2.2 What Happens When You Enable a Core Plugin in Your MAF Application

Once you enable a core plugin in the MAF Application Editor the plugin is listed in Plugin Enablement in the editor. You can see which features in the application use the selected plugin in the Registered Features Declaring Usage in the Plugin Enablement area. Example 9-1 shows the entries for a MAF application where the Email and Network Information plugins have been enabled. Enabling these plugins is a prerequisite to your MAF application using the device's email client and accessing the internet.

Example 9–1 Enabled Core Plugins in maf-plugins.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<maf-plugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/adf/mf">
  <cordova-plugins>
    <core-cordova-plugin id="c1" pluginId="cordova-plugin-network-information"/>
    <core-cordova-plugin id="c2" pluginId="com.oracle.maf.email"/>
 </cordova-plugins>
</maf-plugins>
```

9.3 Registering Additional Plugins in Your MAF Application

Register additional plugins in your MAF application when you require functionality in your MAF application not provided by the core plugins that MAF delivers.

9.3.1 How to Register an Additional Plugin

You use the MAF Application Editor to register the additional plugin you want your MAF application to use.

Before you begin, ensure that the application, and the plugin to be registered with the application, are stored on the same drive. If, for example, you store your application on the `C:` drive in a Windows environment, you must also store the plugin that you want to register with the application on the `C:` drive. This ensures that OEPE, using a relative path, successfully registers the plugin with your application.

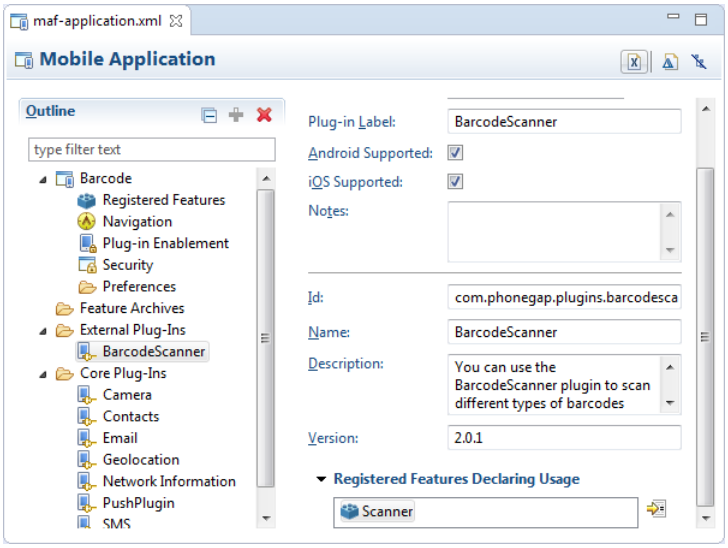
To register an additional plugin for a MAF application:

1. From the Project Explorer, expand the assembly project folder, then expand MAF and double-click MAF Application Editor.
2. In the editor under Outline, right-click **External plugins** and choose **New > External Cordova Plugin**.
3. Enter the location of the plugin in **URI**. Click to open the Mobile Plugin location dialog. Enter either an Eclipse workspace container, or navigate to an external folder containing the plugin.
4. To re-inspect the plugin and update the information if the content of the URI has changed, click . This gets the information available from the metadata of the current plugin. This can include:
   - Id
   - Name
   - Description
   - Version
   - Platform support. This can be edited so you can override the platform availability if you need to.
   - Notes. You can add information that may be useful to understand the history of this registration in the application, or for any other purpose.

9.3.2 What Happens When You Register an External Plugin for Your MAF Application

Once you select the source files for the plugin you want your MAF application to use, OEPE edits the application's maf-plugins.xml file with entries that identify the enabled plugins in your MAF application. Example 9–2 shows the entries in a maf-plugins.xml file where the Globalization plugin shown in Figure 9–2 has been registered with the MAF application.

Figure 9–2 Additional Plugin in the MAF Application Editor



Example 9–2 Additional Plugin in maf-plugins.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<maf-plugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns="http://xmlns.oracle.com/adf/mf">
  <cordova-plugins>
    <core-cordova-plugin id="c1" pluginId="cordova-plugin-network-information"/>
    <cordova-plugin id="c2" pluginId="cordova-plugin-globalization"

path="../../../../CordovaPlugins/cordova-plugin-globalization/">
      <platform id="p1" name="android" enabled="true"/>
      <platform id="p2" name="ios" enabled="true"/>
      <platform id="p3" name="windows" enabled="true"/>
    </cordova-plugin>
  </cordova-plugins>
</maf-plugins>
```

9.4 Deploying Plugins with Your MAF Application

The deployment of a plugin with your MAF application depends on the chosen method of deployment.

Deployment to a FAR

A deployment to a FAR includes a copy of the `maf-plugins.xml` file of the application named `jar-maf-plugins.xml`. It is identical to the `maf-plugins.xml` file of the application with the exception that the path attribute value of each plugin is an empty string. A FAR deployment does not include the source files for the plugin.

Deployment to a Mobile Application Archive File

A deployment to a Mobile Application Archive File includes a copy of the `maf-plugins.xml` file of the application with all path attributes set to an empty string.

Deployment Using an Android, iOS, or Windows Deployment Configuration

During deployments using an Android, iOS or Windows deployment configuration, OEPE invokes tools that build and deploy the application. These tools, in turn, invoke the Cordova plugman tool to install the configured plugins from their source location to the deployment folder.

Resolving Naming Conflicts Between Plugins

Deployment can fail due to naming conflicts if more than one plugin used by your MAF application contains resource files with the same name. For example, deployment fails if a MAF application uses two plugins that both have a resource file name arrays.xml.

To resolve these naming conflicts, rename the resource file in the plugin that conflicts with the resource file name in the other plugin. Update the reference to the resource file in the plugin.xml file of the first plugin. In our example, this requires you to rename the array.xml resource file name of the first plugin to pluginone_arrays.xml and edit the plugin.xml file of the plugin as follows:

```
<source-file src="src/android/LibraryProject/res/values/pluginone_arrays.xml"
                                              target-dir="res/values"/>
```

Adding Missing Dependent Plugins

Deployment can fail if an additional plugin that your MAF application uses does not locate the plugins that it requires (dependent plugins). This scenario can arise if you work behind a firewall. At deployment time, OEPE invokes the tools of Apache Cordova to manage plugins dependencies. These tools may fail to download dependent plugins if their proxy settings are not configured to allow the download of

dependent plugins. To work around this scenario, download the missing dependent plugin, and add it to your MAF application. You add the missing dependent plugin the same way as other plugins that you want to add to your MAF application. For more information, see Section 9.3, "Registering Additional Plugins in Your MAF Application." After you add the dependent plugin, make sure that it appears before the plugin that requires it in the `maf-plugins.xml` file, as demonstrated in Example 9–3.

Example 9–3 Adding Dependent Plugins to the MAF Application

```
<maf-plugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/adf/mf">
   <cordova-plugins>
     ....
    <cordova-plugin id="c2" pluginId="com.example.dependent.dependentPlugin"

path="../../../../../plugins/Dependent-Plugin-Required-By-PluginWithID_c3/">
      ...
    <cordova-plugin id="c3" pluginId="com.example.plugin"
                    path="../../../../../plugins/AdditionalPlugin/">
      ...
   </cordova-plugins>
</maf-plugins>
```

9.5 Importing Plugins from a Feature Archive File

When you import a FAR that contains a `jar-maf-plugins.xml` file to your application, the content in the `jar-maf-plugins.xml` file merges with the `maf-plugins.xml` file of the consuming application. OEPE logs information about the merge to its Messages window.

If the plugin to import from the FAR already exists in the `maf-plugins.xml` file of the consumer application, OEPE logs a message that the plugin exists in the application, and will not be merged.

If the plugin to import from the FAR does not exist in the `maf-plugins.xml` file of the consumer application, OEPE adds the plugin to the `maf-plugins.xml` file of the application. In this scenario, you need to set the path to the newly-imported plugin, as described in Section 9.3, "Registering Additional Plugins in Your MAF Application."

9.6 Using a Plugin in a MAF Application

Once you register a plugin or enable a core plugin in your MAF application, you can create content in the MAF application that uses the plugin.

See "Integrating a custom Cordova plugin into a MAF app" at `http://blogs.oracle.com/mobile/entry/integrating_a_custom_cordova_plugin` for information about how you can invoke a plugin from Java, from a MAF AMX page, and from local HTML.

The `BarcodeDemo` sample application also demonstrates how you can accomplish this task.

Figure 9–3 Platform-Specific Content To Access a Plugin

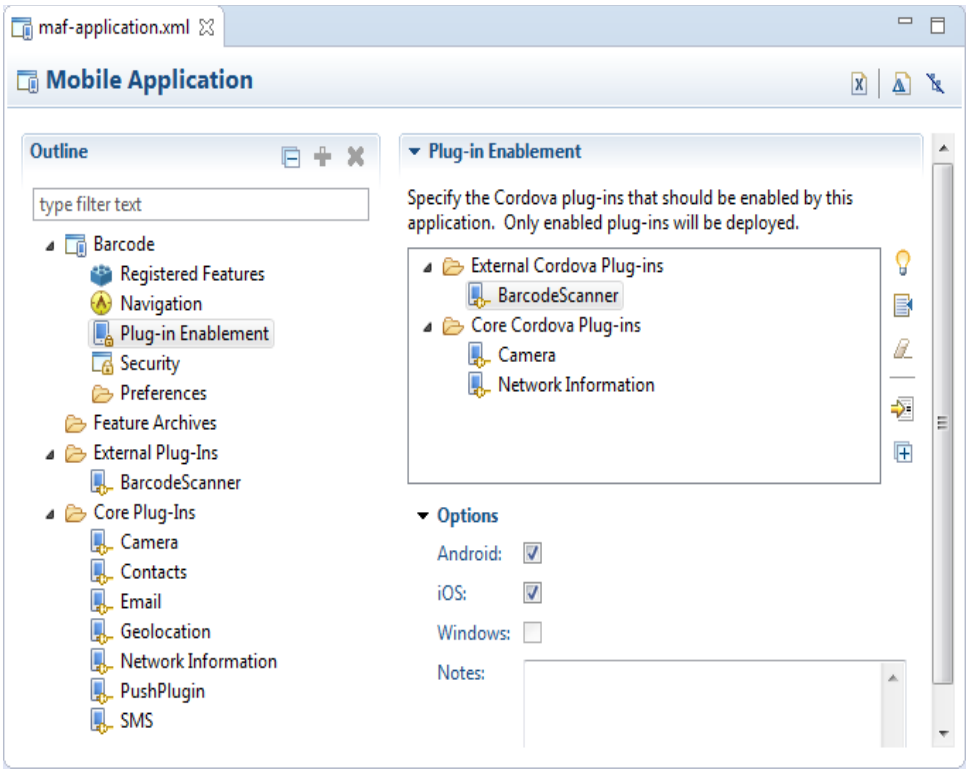
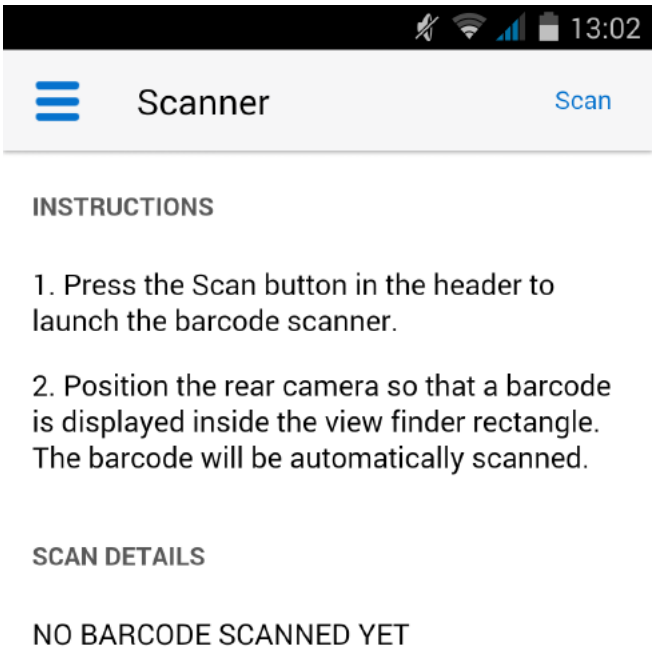


Figure 9–3 shows a button (Scan) in the MAF AMX page that the BarcodeDemo sample application renders on the end user's device at runtime. This button invokes a managed bean method and the managed bean method invokes a JavaScript function that calls the BarcodeScanner plugin.

Figure 9–4 Command Button Invoking Managed Bean Method to Access Plugin



Example 9–4 shows a number of code extracts from the `BarcodeDemo` sample application.

Other sample applications, apart from the `BarcodeDemo` sample application, that demonstrate how to use additional plugins in MAF applications are `BeaconDemo`, and `DatePicker`. For information about how to access and use these sample applications, see Appendix E, "MAF Sample Applications".

Example 9–4 Using the Barcode Scanner Plugin

```
<!--  The following code snippet from the scanner.amx file shows how the Scan
button invokes the scanBarcode method in the managed bean -->
<amx:commandButton text="Scan" id="cl2"
actionListener="#{viewScope.BarcodeBean.scanBarcode}"/>

<!--  The following code snippet from the BarcodeBean.java file shows how the
scanBarcode managed bean method invokes a JavaScript function -->
  public void scanBarcode (ActionEvent event)
  {
    // Invokes a JavaScript function named "scanBarcodeFromJavaBean"

AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.getFe
atureId(),

"scanBarcodeFromJavaBean",
                                                                  new Object[] { });
  }

<!--  The following code snippet from the scanner.js file shows how the JavaScript
function accesses the barcode scanner and sets the resulting value in a managed
bean field.-->
function scanBarcodeFromJavaBean(options)
        {
            cordova.plugins.barcodeScanner.scan(
              function(result)
               function onSuccess(result) {
        adf.mf.api.setValue( { "name": "#{viewScope.BarcodeBean.barcodeError}",
                               "value": ""},
                               function() {},
                               function() {});

        adf.mf.api.setValue( { "name": "#{viewScope.BarcodeBean.barcodeResult}",
                               "value": result.text},
                               function() {},
                               function() {});

        adf.mf.api.setValue( { "name": "#{viewScope.BarcodeBean.barcodeFormat}",
                               "value": result.format},
                               function() {},
                               function() {});

        adf.mf.api.setValue( { "name":
"#{viewScope.BarcodeBean.barcodeCancelled}",
                               "value": result.cancelled == 1 ? "Yes" : "No"},
                               function() {},
                               function() {});
    }

    function onError(error) {
        adf.mf.api.setValue( { "name": "#{viewScope.BarcodeBean.barcodeError}",
```

```
                            "value": "ERROR: " + error.text},
                            function() {},
                            function() {});
}

// Callable externally
scanBarcodeFromJavaBean = function() {
    cordova.plugins.barcodeScanner.scan(onSuccess, onError);
}
```

10

Using Lifecycle Listeners in MAF Applications

This chapter describes using the MAF Application Editor and MAF Features Editor to define the display behavior of the mobile application's springboard and navigation bar and how to designate content by embedding application features.

This chapter includes the following sections:

- Section 10.1, "Introduction to Lifecycle Listeners in MAF Applications"
- Section 10.2, "Registering a Lifecycle Listener for a MAF Application or an Application Feature"
- Section 10.3, "What Happens When You Register a Lifecycle Listener"

10.1 Introduction to Lifecycle Listeners in MAF Applications

Lifecycle listeners are useful locations to write code that executes in response to specific events in your application. MAF provides lifecycle listeners where you can write code in response to application or application feature events. A typical implementation of an application lifecycle listener method may be to write code that initializes your application's database when the application starts, as described in Chapter 19, "Using the Local Database in MAF AMX," or to update a security configuration from URL parameters, as described in HowSection 29.5.6, "How to Update Connection Attributes of a Named Connection at Runtime."

MAF provides the following two interfaces that you can implement to communicate with event notifications:

- `oracle.adfmf.application.LifeCycleListener`

  This interface specifies the following methods that an application lifecycle listener must implement:

  - `activate()`
  - `deactivate()`
  - `start()`
  - `stop()`

- `oracle.adfmf.feature.LifeCycleListener`

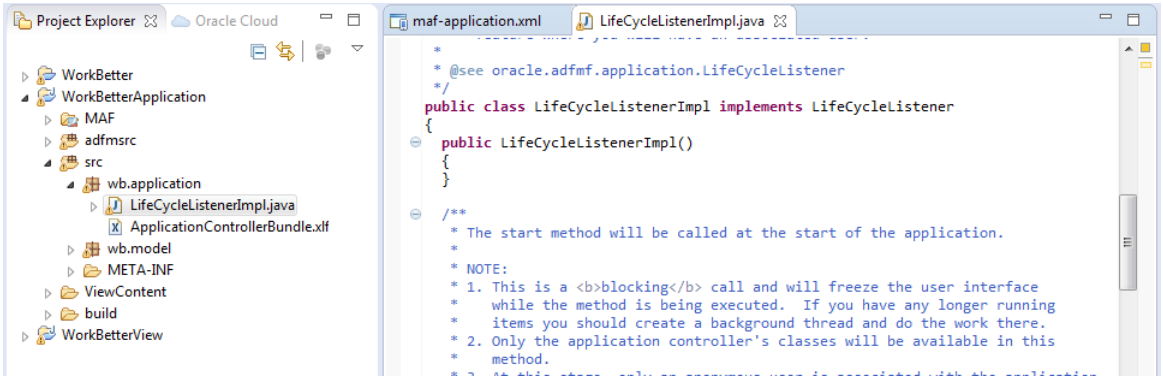
  This interface specifies the following methods that a feature lifecycle listener must implement:

  - `activate()`
  - `deactivate()`

You create a lifecycle listener by creating a Java class that implements the appropriate interface and registering the implementation in your MAF application, as described in Section 10.2, "Registering a Lifecycle Listener for a MAF Application or an Application Feature."

A new MAF application that you create implements the `oracle.adfmf.application.LifeCycleListener` interface through the default creation of the `application.LifeCycleListenerImpl.java` class in the application project, as shown in Figure 10–1.

Figure 10–1 Implementation of Application Lifecycle Listener



Note that the application lifecycle listener is executed with an anonymous user (that is, there is no user associated with any of its methods and no secure web service is called).

Table 10–1 describes the specific times that MAF invokes application lifecycle methods during an application's startup, shutdown, and hibernation.

Table 10–1 Timing of MAF's Invocation of Application Lifecycle Methods

Method	Timing	When Called	Usage
`start`	Called after the MAF application has completely loaded the application features and immediately before presenting the user with the initial application feature or the springboard. This is a blocking call.	When the application process starts.	Uses include: ▪ Determining if there are updates to the MAF application. ▪ Requesting a remote server to download data to the local database.
`stop`	Called as the MAF application begins its shutdown.	When the application process terminates.	Uses include: ▪ Logging off from any remote services. ▪ Uploading any data change to the server before the application is closed.
`activate`	Called as the MAF application activates from being situated in the background (hibernating). This is a blocking call.	After the `start` method is called.	Uses include: ▪ Reading and re-populating cache stores. ▪ Processing web service requests. ▪ Obtaining required resources.
deactivate	Called as the MAF application deactivates and moves into the background (hibernating). This is a blocking call.	Before the `stop` method is called.	Uses include: ▪ Writing the restorable state. ▪ Closing the database cursor and the database connection.

Table 10–2 describes the specific times that MAF invokes feature lifecycle methods during a feature's activation and deactivation.

Table 10–2 Timing of MAF's Invocation of Feature Lifecycle Methods

Method	Timing	When Called	Usage
`activate`	Called before the current application feature is activated.	Called when a user selects the application feature for the first time after launching a MAF application, or when the application has been re-selected (that is, brought back to the foreground).	Uses include: ▪ Reading the `applicationScope` variable. ▪ Setting the current row on the first MAF AMX view.
`deactivate`	Called before the next application feature is activated, or before the application feature exits.	Called when the user selects another application feature.	You can, for example, use the deactivate event to write the `applicationScope` variable, or any other state information, for the next application feature to consume.

For more information about the `oracle.adfmf.application.LifeCycleListener` and `oracle.adfmf.feature.LifeCycleListener` interfaces, see the *Java API Reference for Oracle Mobile Application Framework*.

The `LifecycleEvents` sample application demonstrates declaring listener classes that implement both the application and feature interfaces. It registers these listener classes in the MAF application's `maf-application.xml` and `maf-feature.xml files`. For more information about this and other sample applications, see Section 1.4, "MAF Sample Applications.".

10.2 Registering a Lifecycle Listener for a MAF Application or an Application Feature

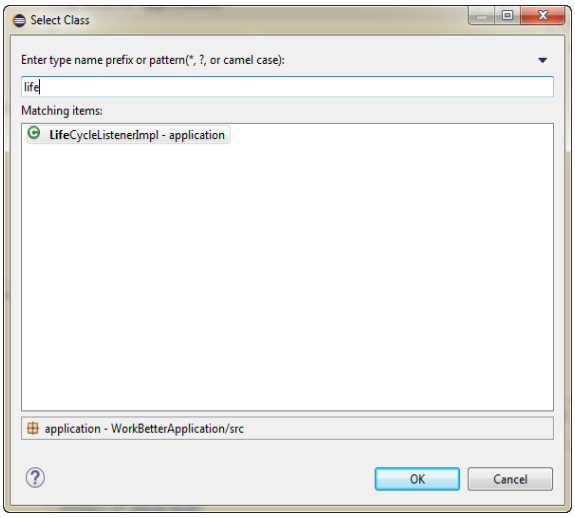
You register an application lifecycle listener using the MAF Application Editor and a feature lifecycle listener using the MAF Features Editor.

To register an application lifecycle listener:

1. In the Project Explorer, expand the assembly project, then MAF and double-click **MAF Application Editor**.
2. In the Outline, select the application node, and in the Application tab specify the Java class that implements the `oracle.adfmf.application.LifeCycleListener` interface in the **Lifecycle Event Listener** field.

> **Note:** If you place the mouse pointer over the label for **Lifecycle Event Listener**, OEPE displays a tooltip describing the listener class.

Figure 10–2 Retrieving the Application Event Listener



To register an application feature lifecycle listener:

1. In the Project Explorer, expand the view project, then MAF and double-click **MAF Features Editor**.
2. Select the feature in the Features list for which you want to register a feature lifecycle listener.
3. In the Lifecycle Event Listener field, specify the Java class that implements the `oracle.adfmf.feature.LifeCycleListener` interface.

10.3 What Happens When You Register a Lifecycle Listener

By default, a MAF application that you create implements an application lifecycle listener through the creation of the `application.LifeCycleListenerImpl.java` class in your application's application project. The `listener-class` attribute in the `maf-application.xml` file registers this class, as shown in the following example.

```
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                   version="1.0" name="NewMAFapp" id="com.company.NewMAFapp"
    appControllerFolder="ApplicationController"
listener-class="application.LifeCycleListenerImpl">
...
</adfmf:application>
```

OEPE writes an entry to the `maf-feature.xml file` for the `listener-class` attribute when you register a feature lifecycle listener. The following example shows an entry in the `LifecycleEvents` sample application described in Appendix E, "MAF Sample Applications.".

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
  <adfmf:feature id="Feature1" name="Feature1"
listener-class="mobile.Feature1Handler">
    <adfmf:description>This is a sample feature to show the feature lifecycle
handlers.
    </adfmf:description>
    <adfmf:content id="Feature1.1">
```

```
      <adfmf:amx file="Feature1/feature1.amx"/>
    </adfmf:content>
  </adfmf:feature>
...
</adfmf:features>
```

11

Creating MAF AMX Pages

This chapter describes how to create the MAF AMX application feature, including views and task flows.

This chapter includes the following sections:

- Section 11.1, "Introduction to the MAF AMX Application Feature"
- Section 11.2, "Creating Task Flows"
- Section 11.3, "Creating Views"

11.1 Introduction to the MAF AMX Application Feature

MAF AMX is a subframework within Mobile Application Framework (MAF) that provides a set of UI components that enable you to create an application feature whose behavior is identical on all supported platforms. MAF AMX allows you to use UI components declaratively by dragging them onto a page editor. A typical MAF AMX application feature includes several interconnected pages that can be navigated through various paths.

> **Note:** When developing interfaces for mobile devices, always be aware of the fact that the screen space is very limited. In addition, touchscreen support is not available on some mobile devices.

For more information, see the following:

- Chapter 2, "Getting Started with MAF Application Development"
- Chapter 12, "Creating the MAF AMX User Interface"
- Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX"

> **Note:** When using the MAF Feature editor to create objects as part of an application, OEPE creates an ID for the object that cannot be changed later.

11.2 Creating Task Flows

Task flows allow you to define the navigation between MAF AMX pages. Using your application workspace in OEPE (see Section 2.3, "Creating a MAF Application"), you can start creating the user interface for your MAF AMX application feature by designing task flows. MAF AMX uses navigation cases and rules to define the task

flow. These definitions are stored in a file with the default name of `task-flow-definition.xml` (see Section 11.2.3, "What You May Need to Know About the task-flow-definition.xml File").

A MAF sample application called Navigation (select **File > New > MAF Examples**, then select **Navigation**) demonstrates how to use various navigation techniques, such as circular navigation, routers, and so on.

MAF enables you to create MAF AMX application features that have both bounded and unbounded task flows. As described in Section 11.2.12, "What You May Need to Know About Bounded and Unbounded Task Flows," a bounded task flow is also known as a task flow definition and represents the reusable portion of an application. In MAF, bounded task flows have a single entry point and no exit points. They have their own collections of activities and control-flow rules, as well as their own memory scope and managed-bean life span. Other characteristics of bounded task flows include accepting input parameters (see Section 11.2.12.3.1, "Passing Parameters to a Bounded Task Flow") and generating return values (see Section 11.2.12.3.2, "Configuring a Return Value from a Bounded Task Flow").

You use the MAF AMX Task Flow Designer to create bounded task flows for your application feature. This tool includes a diagrammer (see Section 11.2.4, "What You May Need to Know About the MAF Task Flow Diagrammer") in which you build the task flow by dragging and dropping activities and control flows (see Section 11.2.2, "What You May Need to Know About Task Flow Activities and Control Flows") from the Palette. You then define these activities and the transitions between them using the Properties window.

Unless a task flow has already been created, MAF automatically generates a default unbounded task flow (`adfc-mobile-config.xml` file) when a new MAF AMX page is created.

11.2.1 How to Create a Task Flow

A task flow is composed of the task flow itself and a number of activities with control flow rules between those activities (see Section 11.2.2, "What You May Need to Know About Task Flow Activities and Control Flows"). Typically, the majority of the activities are view activities which represent different pages in the flow. When a method or operation needs to be called (for example, before a page is rendered), you use a method call activity with a control flow case from that activity to the appropriate next activity. When you want to call another task flow, you use a task flow call activity. If the flow requires branching, you use a router activity. At the end of a bounded task flow, you use a return activity which allows the flow to exit and control is sent back to the flow that called this bounded task flow.

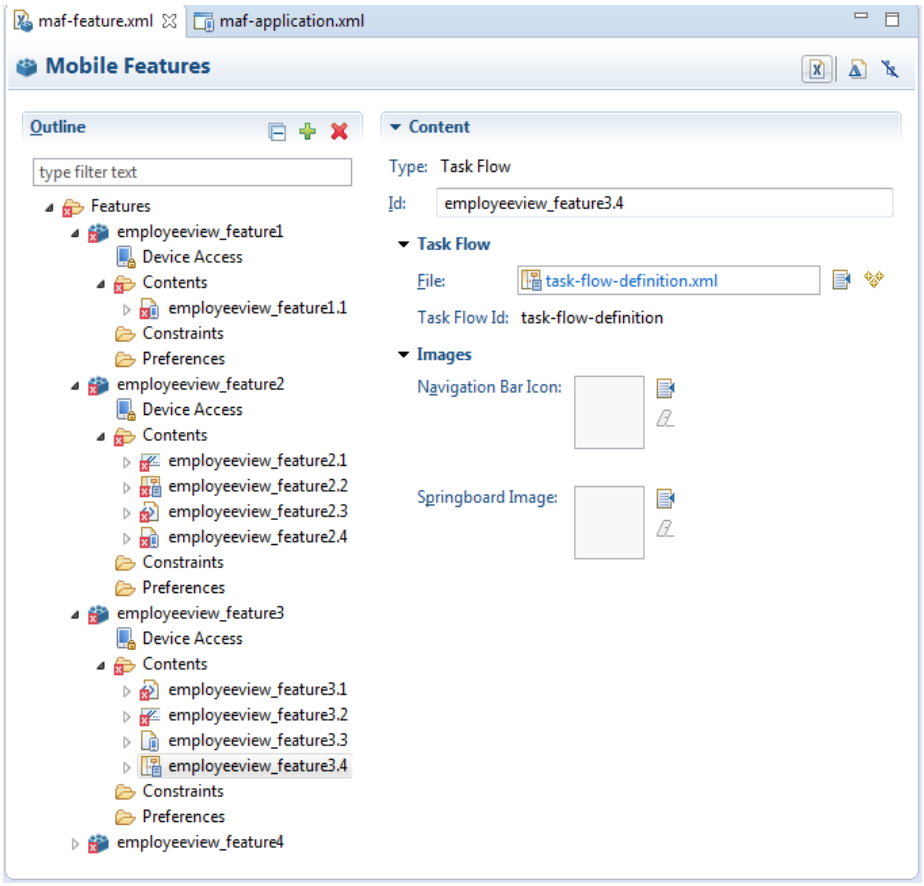
You can use the Palette to declaratively create a bounded task flow for your MAF AMX application feature by dragging and placing task flow elements in the flow structure. When you place these elements in relation to one another, OEPE creates the XML metadata needed for navigation to work in your MAF AMX application feature in the `task-flow-definition.xml` file (default). You can also edit the task flow definition file's source.

Before you begin:

To design a task flow, the MAF application must include a View Controller project file (see Chapter 2, "Getting Started with MAF Application Development").

To create a task flow in OEPE, create a bounded task flow from the MAF Features Editor. For more information, see Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML."

Figure 11–1 Creating the Task Flow File from the MAF Features editor



Additionally, you can manually add elements to the task flow file by directly editing the page in the Source editor. To open the file in the Source editor, click the **Source** tab.

Note: When manually editing the task flow file, keep in mind that all the document file names referring to MAF AMX pages, Java Script files, and CSS files are case-sensitive.

If special characters (such as an underscore, for example) are used in a file name, you should consult the mobile device specification to verify whether or not the usage of this character is supported.

Once the navigation for your MAF AMX application feature is defined, you can create the pages and add the components that will execute the navigation. For more information about using navigation components on a page, see Section 11.2.6, "How to Define Control Flows."

After you define the task flow for the MAF AMX application feature, you can double-click a view file to access the MAF AMX view. For more information, see Section 11.3, "Creating Views."

11.2.2 What You May Need to Know About Task Flow Activities and Control Flows

A task flow consists of activities and control flow cases that define the transitions between activities.

The MAF Task Flow designer supports activities listed in Table 11–1.

Table 11–1 Task Flow Activities

Activity	Description
View	Displays an MAF AMX page. For more information, see Section 11.2.5.1, "Adding View Activities."
Method Call	Invokes a method (typically a method on a managed bean). You can place a method call activity anywhere in the control flow of an MAF AMX application feature to invoke logic based on control flow rules. For additional information, see Section 11.2.5.3, "Adding Method Call Activities." You can also specify parameters that you pass into a method call that is included in a task flow. These include standard parameters for a method call action in an MAF AMX task flow. When you use the designer to generate a method, it adds the required arguments and type. At runtime, you can define parameters for a method call in a task flow and pass parameters into the method call itself for its usage. For more information, see Part 11.2.5, "How to Add and Use Task Flow Activities."
Router	Evaluates an Expression Language (EL) expression and returns an outcome based on the value of the expression. These outcomes can then be used to route control to other activities in the task flow. For more information, see Section 11.2.5.2, "Adding Router Activities."
Task Flow Call	Calls a bounded task flow from either an unbounded or bounded task flow. While a task flow call activity allows you to call a bounded task flow located within the same MAF AMX application feature, you can also call a bounded task flow from a different MAF AMX application feature or from a Feature Archive file (FAR) that has been added to a library. The task flow call activity supports task flow input parameters and return values. For more information, see Section 11.2.5.4, "Adding Task Flow Call Activities."
Task Flow Return	Identifies the point in an application's control flow where a bounded task flow completes and sends control flow back to the caller. You can use a task flow return only within a bounded task flow. For more information, see Section 11.2.5.5, "Adding Task Flow Return Activities."

The MAF Task Flow designer supports control flows listed in Table 11–2.

Table 11–2 Control Flows

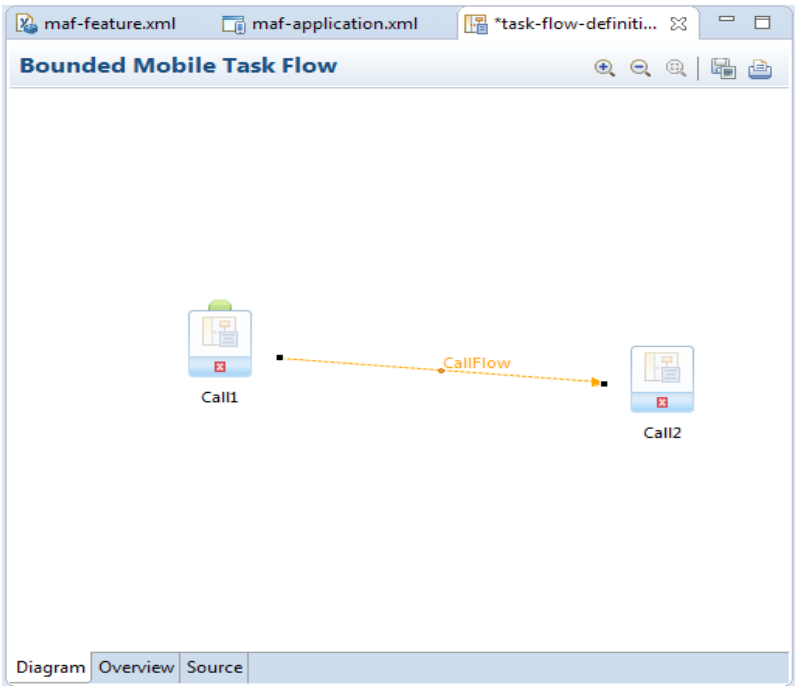
Control Flow	Description
Control Flow Case	Identifies how control passes from one activity to the next in the MAF AMX application feature. For more information, see Section 11.2.6.1, "Defining a Control Flow Case."
Wildcard Control Flow Rule	Represents a control flow case that can originate from any activities whose IDs match a wildcard expression.For more information, see Section 11.2.6.2, "Adding a Wildcard Control Flow Rule."

11.2.3 What You May Need to Know About the task-flow-definition.xml File

The `task-flow-definition.xml` file enables you to design the interactions between views (MAF AMX pages) by dragging and dropping task flow components from the Palette onto the diagrammer.

Figure 11–2 shows a sample task flow file called `task-flow-definition.xml`. In this file, the control flow is directed from `Call1` to `Call2` by the task flow call `CallFlow`. (See Section 11.2.7, "What You May Need to Know About MAF Support for Back Navigation").

Figure 11–2 Task Flow File



11.2.4 What You May Need to Know About the MAF Task Flow Diagrammer

As illustrated in Figure 11–2, the task flow diagram and Palette display automatically after you create a task flow using the MAF Task Flow Creation utility. The task flow diagram is a visual editor onto which you can drag and drop activities and task flows from the Palette. You can also edit the task flow file directly, by clicking the Source tab shown in Figure 11–2. For more information, see Section 11.2.5, "How to Add and Use Task Flow Activities."

11.2.5 How to Add and Use Task Flow Activities

You use the task flow diagrammer, selecting options from the Palette, to drag and drop activities, views, and control flows.

To add an activity to an MAF task flow:

1. In the Project Explorer, double-click a task flow source file (such as `task-flow-definition.xml`) to display the task flow diagram and the Palette. The diagrammer displays the task flow editor. The Palette automatically displays the components available for an MAF task flow.
2. Drag an activity from the Palette onto the diagram.

Note: There is a default activity that is associated with each bounded task flow.

11.2.5.1 Adding View Activities

One of the primary types of task flow activity is the view activity which displays an MAF AMX page.

XML metadata in the source file of the task flow associates a view activity with a physical MAF AMX page. An `id` attribute identifies the view activity.

You can configure view activities in your task flow to pass control to each other at runtime. For example, to pass control from one view activity (view activity A) to a second view activity (view activity B), you could configure a command component, such as a Button or a Link on the page associated with view activity A. To do so, you set the command component's Action attribute to the control flow case `from-outcome` that corresponds to the task flow activity that you want to invoke (for example, view activity B). At runtime, the end user initiates the control flow case by invoking the command component. It is possible to navigate from a view activity to another activity using either a constant or dynamic value on the Action attribute of the UI component:

- A constant value of the component's Action attribute is an action outcome that always triggers the same control flow case. When an end user clicks the component, the activity specified in the control flow case is performed. There are no alternative control flows.
- A dynamic value of the component's Action attribute is bound to a managed bean or a method. The value returned by the method binding determines the next control flow case to invoke. For example, a method might verify user input on a page and return one value if the input is valid and another value if the input is invalid. Each of these different action values trigger different navigation cases, causing the application to navigate to one of two possible target pages.

For more information, see Section 11.2.9, "How to Specify Action Outcomes Using UI Components."

You can also write an EL expression that must evaluate to `true` before control passes to the target view activity. You write the EL expression as a value for the `<if>` child element of the control flow case in the task flow.

Example 11–1 and Example 11–2 demonstrate what happens when you pass control between View activities:

Example 11–1 shows a control flow case defined in the XML source file for a bounded or unbounded task flow.

Example 11–1 Control Flow Case Defined in XML Source File

```
<control-flow-rule>
   <from-activity-id>Start</from-activity-id>
      <control-flow-case>
         <from-outcome>toOffices</from-outcome>
         <to-activity-id>WesternOffices</to-activity-id>
      </control-flow-case>
</control-flow-rule>
```

As Example 11–2 shows, a Button on an MAF AMX page associated with the Start view activity specifies `toOffices` as the `action` attribute. When the end user clicks the

button, control flow passes to the `WesternOffices` activity specified as the `to-activity-id` in the control flow metadata.

Example 11–2 Static Navigation Button Defined in a View Activity

```
<amx:commandButton text="Go" id="b1" action="toOffices">
```

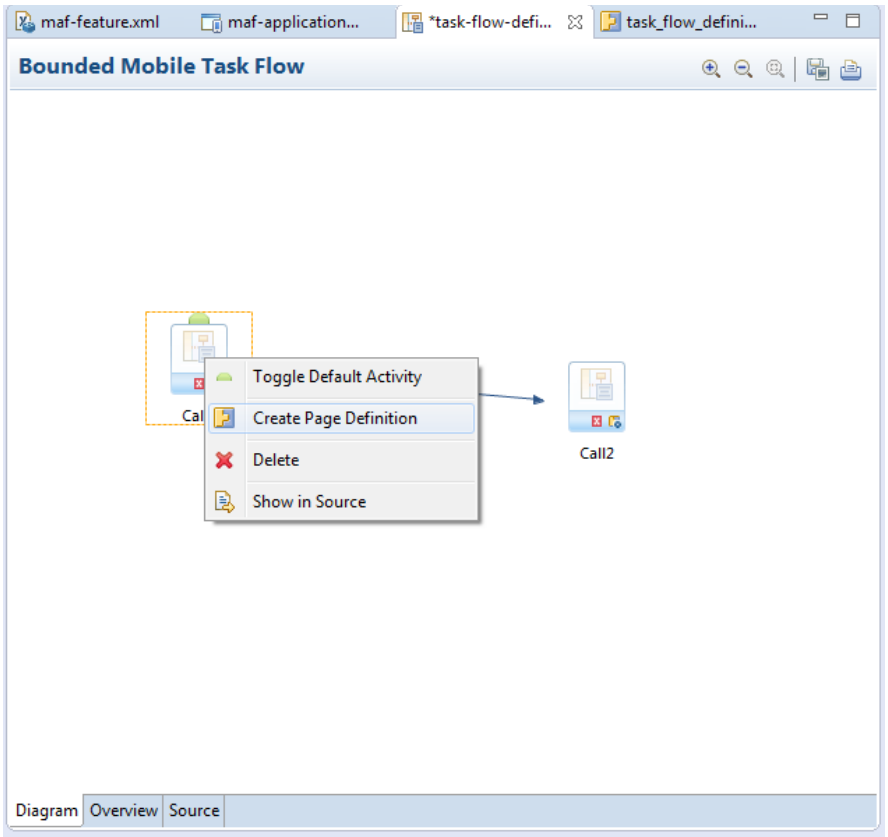
For more information, see the following:

- Section 11.2.5.1, "Adding View Activities"
- Section 11.3.1.2, "Creating MAF AMX Pages"

As previously stated, the view activity is associated in metadata with an actual MAF AMX page which it displays when added to a task flow. You add a view activity by dragging and dropping it from the Palette. You can create an actual MAF AMX page by right-clicking the View activity in the Diagram window and then define characteristics for the page through the displayed dialog (see Figure 11–15, "Create New MAF Page Dialog").

If you are creating a bounded task flow, you may want to designate a specific activity as the default activity (see Section 11.2.12, "What You May Need to Know About Bounded and Unbounded Task Flows"). This allows the specific activity to execute first whenever the bounded task flow runs. By default, OEPE makes the first activity you add to the task flow the default. To change to a different activity, right-click the appropriate activity in the Diagram window and choose **Create Page Definition** (see Figure 11–3).

Figure 11–3 Defining Default Activity



11.2.5.2 Adding Router Activities

You use a router activity to route control to activities based on the runtime evaluation of EL expressions.

Each control flow corresponds to a different router case. Each router case uses the following elements to choose the activity to which control is next routed:

- `expression`: an EL expression that evaluates to `true` or `false` at runtime.

  The router activity returns the outcome that corresponds to the EL expression that returns `true`.

- `outcome`: a value returned by the router activity if the EL expression evaluates to `true`.

  If the router case `outcome` matches a `from-outcome` on a control flow case, control passes to the activity that the control flow case points to. If none of the cases for the router activity evaluate to `true`, or if no router activity cases are specified, the `outcome` specified in the router Default Outcome field (if any) is used.

Consider using a router activity if your routing condition can be expressed in an EL expression: the router activity allows you to show more information about the condition on the task flow.

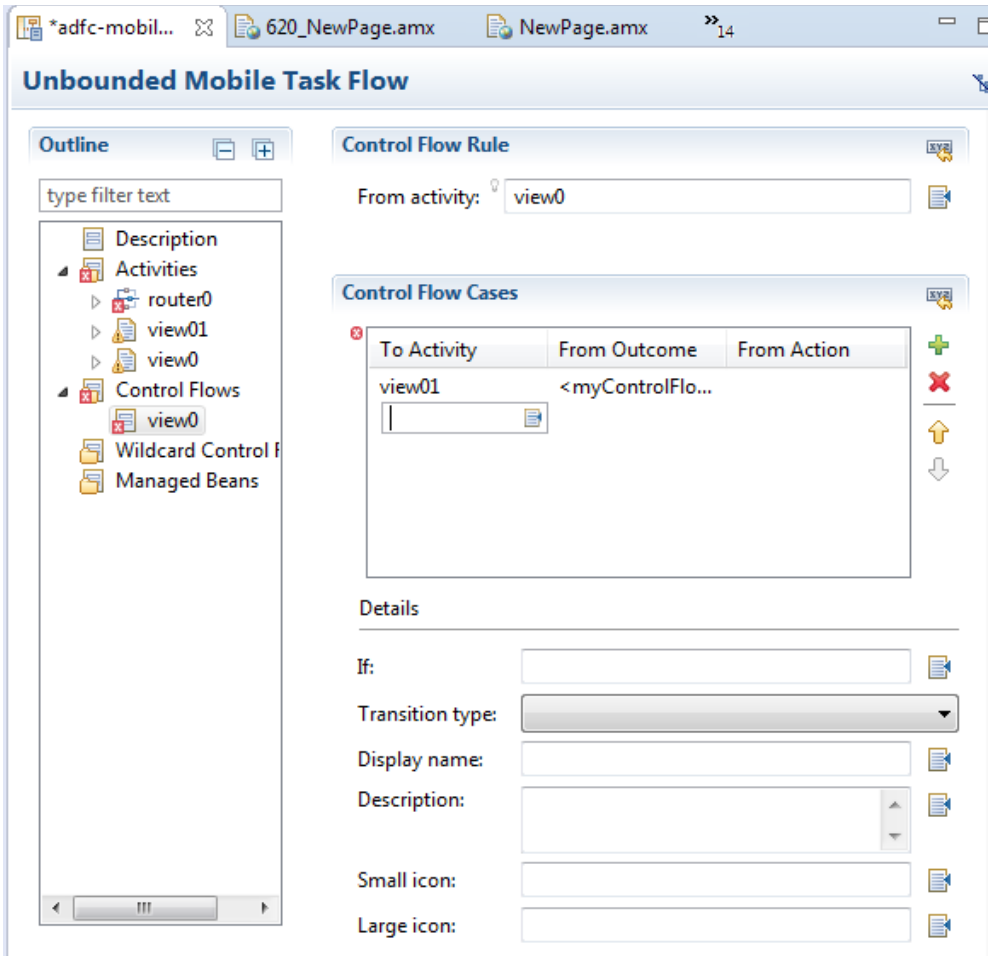
When you drag a Router activity onto the diagram, you can use the Properties window to create an expression whose evaluation determines which control flow rule to follow. Using the Properties window, you configure the **Activity ID** and **Default Outcome** properties of the router activity and add router cases to the router activity.

When defining the Activity ID attribute, write a value that identifies the router activity in the task flow's source file.

When defining the Default Outcome attribute, specify an activity that the router activity passes control to if no control flow cases evaluate to true or if no control flow case is specified.

For each router case that you add, specify values by clicking Add (+) in the **Control Flow Cases** section, as shown in Figure 11–4.

Figure 11–4 Configuring Router Activity



- **Expression**: An EL expression that evaluates to `true` or `false` at runtime.

  For example, to reference the value in an input text field of a view activity, write an EL expression similar to the following:

  ```
  #{pageFlowScope.value=='view2'}
  ```

  If this EL expression returns `true`, the router activity invokes the outcome that you specify in the Outcome field.

- **Outcome**: The outcome the router activity invokes if the EL expression specified by Expression returns `true`.

  Create a control flow case or a wildcard control flow rule for each outcome in the diagram of your task flow. For example, for each outcome in a control flow case, ensure that there is a corresponding `from-outcome`.

When you configure the control flow using a router activity, OEPE writes values to the source file of the task flow based on the values that you specify for the properties of the router activity.

11.2.5.3 Adding Method Call Activities

When you drag a Method Call activity onto the diagram, you can use the New Object dialog to choose the method to call.

Use a method call activity to call a custom or built-in method that invokes the MAF AMX application feature logic from anywhere within the application feature's control flow. You can specify methods to perform tasks such as initialization before displaying a page, cleanup after exiting a page, exception handling, and so on.

You can set an outcome for the method that specifies a control flow case to pass control to after the method finishes. You can specify the outcome as one of the following:

- `fixed-outcome`: upon successful completion, the method always returns this single outcome, for example, `success`. If the method does not complete successfully, an outcome is not returned. If the method type is void, you must specify a `fixed-outcome` and cannot specify `to-string`.

  You define this outcome by setting the **Fixed Outcome** field in the Properties window.

- `to-string`: if specified as `true`, the outcome is based on calling the `toString` method on the Java object returned by the method. For example, if the `toString` method returns `editBasicInfo`, navigation goes to a control flow case named `editBasicInfo`.

  You define this outcome by setting the **toString()** field in the Properties window.

You can associate the method call activity with an existing method by dropping a data control operation from the Data Controls window directly onto the method call activity in the task flow diagram. You can also drag methods and operations directly to the task flow diagram: a new method call activity is created automatically when you do so. You can specify an EL expression and other options for the method.

You configure the method call by modifying its activity identifier in the **Activity ID** field if you want to change the default value. If you enter a new value, the new value appears under the method call activity in the diagram.

In the **Method** field, enter an EL expression that identifies the method to call. Note that the bindings variable in the EL expression references a binding from the current binding container. In order to specify the bindings variable, you must specify a binding container definition or page definition. For more information, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts."

You can also use the Expression Builder to build the EL expression for the method:

- Choose Method Expression Builder from the Property Editor for the Method field.
- In the Expression Builder dialog, navigate to the method that you want to invoke and select it.

  If the method call activity is to invoke a managed bean method, double-click the method call activity in the diagram. This invokes a dialog where you can specify the managed bean method you want to invoke.

You can specify parameters and return values for a method by using the **Parameters** section of the Properties window. If parameters have not already been created by associating the method call activity to an existing method, add the parameters by clicking Add (+) and setting the following:

- **Class**: enter the parameter class. For example, `java.lang.Double`.
- **Value**: enter an EL expression that retrieves the value of the parameter. For example:

  ```
  #{pageFlowScope.shoppingCart.totalPurchasePrice}
  ```

- **Return Value**: enter an EL expression that identifies where to store the method return value. For example:

  ```
  #{pageFlowScope.Return}
  ```

11.2.5.4 Adding Task Flow Call Activities

You can use a task flow call activity to call a bounded task flow from either the unbounded task flow (see Section 11.2.12.1, "Unbounded Task Flows") or a bounded task flow (see Section 11.2.12.2, "Bounded Task Flows"). This activity allows you to call a bounded task flow located within the same or a different MAF AMX application feature.

The called bounded task flow executes its default activity first. There is no limit to the number of bounded task flows that can be called. For example, a called bounded task flow can call another bounded task flow, which can call another, and so on resulting in the creation of chained task flows where each task flow is a link in a chain of tasks.

To pass parameters into a bounded task flow, you must specify input parameter values on the task flow call activity. These values must correspond to the input parameter definitions on the called bounded task flow. For more information, see Section 11.2.5.4.2, "Specifying Input Parameters on a Task Flow Call Activity."

Note: The value on the task flow call activity input parameter specifies the location within the calling task flow from which the value is to be retrieved.

The value on the input parameter definition for the called task flow specifies where the value is to be stored within the called bounded task flow once it is passed.

Note: When a bounded task flow is associated with a task flow call activity, input parameters are automatically inserted on the task flow call activity based on the input parameter definitions set on the bounded task flow. Therefore, you only need to assign values to the task flow call activity input parameters.

By default, all objects are passed by reference. Primitive types (for example, `int`, `long`, or `boolean`) are always passed by value.

The technique for passing return values out of the bounded task flow to the caller is similar to the way that input parameters are passed. For more information, see Section 11.2.12.3.2, "Configuring a Return Value from a Bounded Task Flow."

To use a task flow call activity:

1. Call a bounded task flow using a task flow call activity (see Section 11.2.5.4.1, "Calling a Bounded Task Flow Using a Task Flow Call Activity")
2. Specify input parameters on a task flow call activity if you want to pass parameters into the bounded task flow (see Section 11.2.5.4.2, "Specifying Input Parameters on a Task Flow Call Activity").

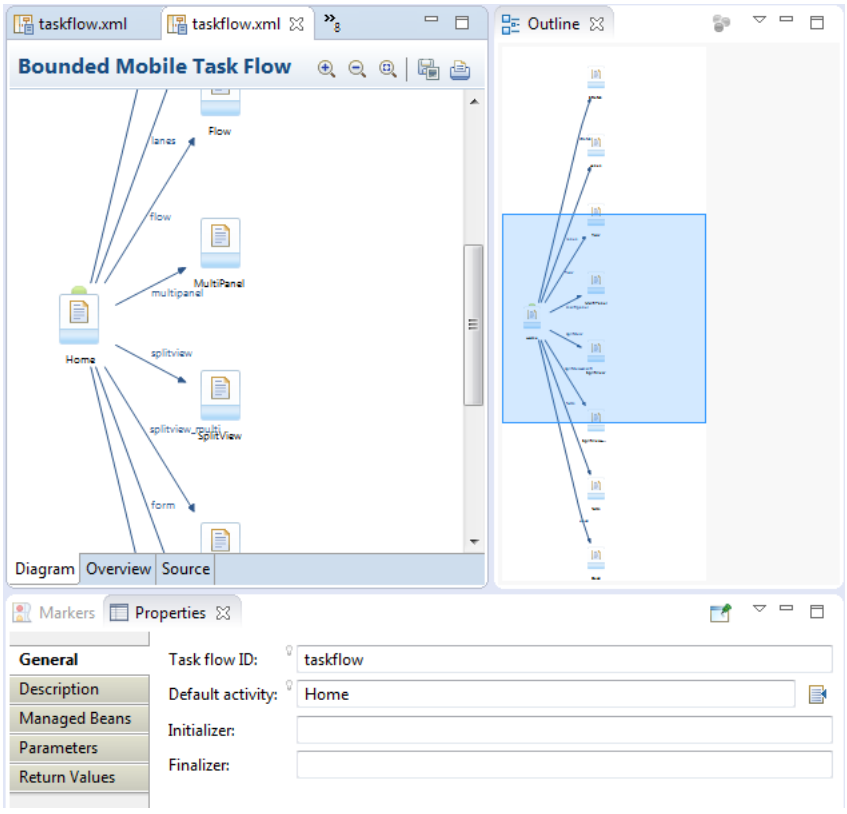
11.2.5.4.1 Calling a Bounded Task Flow Using a Task Flow Call Activity

Add a task flow call activity to the calling bounded or unbounded task flow to call a bounded task flow.

To call a bounded task flow:

1. Open a bounded task flow file in the Diagram view.
2. From the Activities pane of the Palette, drag and drop a Task Flow Call activity onto the diagram.
3. Choose one of the following options to identify the called task flow:
   - Double-click the newly-dropped task flow call activity to open the Create MAF Task Flow dialog where you define settings for a new bounded task flow.
   - Drag an existing bounded task flow from the Project Explorer and drop it on the task flow call activity.
   - If you know the name of the bounded task flow that you want to invoke, perform the following steps:
     a. In the Diagram view, select the Task Flow Call activity.
     b. In the Properties window, expand the General section, and select **Static** from the **Task Flow Reference** list.
     c. In the **Document** field, enter the name of the source file for the bounded task flow to call.
     d. In the **ID** field, enter the bounded task flow ID contained in the XML source file for the called bounded task flow (see Figure 11–5).

Figure 11–5 Task Flow Call Activity That Invokes a Bounded Task Flow



   - If you do not know the name of the bounded task flow to invoke and it is dependent on an end user selection at runtime, perform the following steps:
     a. In the Diagram view, select the Task Flow Call activity.

b. In the Properties window, expand the General section, and select **Dynamic** from the **Task Flow Reference** list.

c. Use the Expression Builder to set the value of the **Dynamic Task Flow Reference** property field: write an EL expression that identifies the ID of the bounded task flow to invoke at runtime.

11.2.5.4.2 Specifying Input Parameters on a Task Flow Call Activity The suggested method for mapping parameters between a task flow call activity and its called bounded task flow is to first specify input parameter definitions for the called bounded task flow. Then you can drag the bounded task flow from the Project Explorer and drop it on the task flow call activity. The task flow call activity input parameters are created automatically based on the bounded task flow's input parameter definition. For more information, see Section 11.2.12.3.1, "Passing Parameters to a Bounded Task Flow."

You can, of course, first specify input parameters on the task flow call activity. Even if you have defined them first, they will automatically be replaced based on the input parameter definitions of the called bounded task flow, once it is associated with the task flow call activity.

If you have not yet created the called bounded task flow, you may still find it useful to specify input parameters on the task flow call activity. Doing so at this point allows you to identify any input parameters you expect the task flow call activity to eventually map when calling a bounded task flow.

To specify input parameters:

1. Open a task flow file in the Diagram view and select a Task Flow Call activity.
2. In the Properties window, expand the Parameters section, and click Add (+) to specify a new input parameter in the Input Parameters list as follows:
   - **Name**: enter a name to identify the input parameter.
   - **Value**: enter an EL expression that identifies the parameter value. The EL expression identifies the location within the calling task flow from which the parameter value is to be retrieved. For example, enter an EL expression similar to the following:

     ```
     #{pageFlowScope.callingTaskflowParm}
     ```

     By default, all objects are passed by reference. Primitive types (for example, `int`, `long`, or `boolean`) are always passed by value.
3. After you have specified an input parameter, you can specify a corresponding input parameter definition for the called bounded task flow. For more information, see Section 11.2.12.3.1, "Passing Parameters to a Bounded Task Flow."

11.2.5.5 Adding Task Flow Return Activities

A task flow return activity identifies the point in an MAF AMX application feature's control flow where a bounded task flow completes and sends control flow back to the caller. You can use a task flow return activity only within a bounded task flow.

A gray circle around a task flow return activity icon indicates that the activity is an exit point for a bounded task flow. A bounded task flow can have zero or more task flow return activities.

Each task flow return activity specifies an `outcome` that it returns to the calling task flow.

The `outcome` returned to an invoking task flow depends on the end user action. You can configure control flow cases in the invoking task flow to determine the next action by the invoking task flow. Set the **From Outcome** property of a control flow case to the value returned by the task flow return activity's `outcome` to invoke an action based on that outcome. This determines control flow upon return from the called task flow.

To add a task flow return activity:

1. Open a bounded task flow file in the Diagram view.
2. From the Activities pane of the Palette, drag and drop a Task Flow Return activity onto the diagram.
3. In the Properties window, expand the General section and enter an outcome in the **Name** field.

   The task flow return activity returns this outcome to the calling task flow when the current bounded task flow exits. You can specify only one outcome per task flow return activity. The calling task flow should define control flow rules to handle control flow upon return. For more information, see Section 11.2.6, "How to Define Control Flows."
4. Expand the Behavior section and choose one of the options in the Reentry list.

   If you specify **reentry-not-allowed** on a bounded task flow, an end user can still click the back button on the mobile device and return to a page within the bounded task flow. However, if the end user does anything on the page such as activating a button, an exception (for example, `InvalidTaskFlowReentry`) is thrown indicating the bounded task flow was reentered improperly.
5. In the End Transaction drop down list, choose one of the following options:
   - **commit**: select to commit the existing transaction to the database.
   - **rollback**: select to roll back the transaction to what it was on entry of the called task flow. This has the same effect as canceling the transaction, since it rolls back a new transaction to its initial state when it was started on entry of the bounded task flow.

     If you do not specify commit or rollback, the transaction is left open to be closed by the calling bounded task flow.

11.2.5.6 Using Task Flow Activities with Page Definition Files

Page definition files define the binding objects that populate data at runtime. They are typically used in an MAF AMX application feature to bind page UI components to data controls. The following task flow activities can also use page definition files to bind to data controls:

- **Method call**: You can drag and drop a data control operation from the Data Controls window onto a task flow to create a method call activity or onto an existing method call activity. In both cases, the method call activity binds to the data control operation.
- **Router**: associating a page definition file with a router activity creates a binding container. At runtime, this binding container makes sure that the router activity references the correct binding values when it evaluates the router activity cases' EL expressions. Each router activity case specifies an outcome to return if its EL expression evaluates to `true`. For this reason, only add data control operations to the page definition file that evaluate to `true` or `false`.
- **Task flow call**: associating a page definition file with a task flow call activity creates a binding container. At runtime, the binding container is in context when

the task flow call activity passes input parameters. The binding container makes sure that the task flow call activity references the correct values if it references binding values when passing input parameters from a calling task flow to a called task flow.

- **View**: you cannot directly associate a view activity with a page definition file. Instead, you associate the page that the view activity references.

If you right-click any of the preceding task flow activities, except view activity, in the Diagram window for a task flow, OEPE displays an option on the context menu that enables you to create a page definition file if one does not yet exist. If a page definition file does exist, OEPE displays a context menu option for all task flow activities to go to the page definition file (see Section 11.3.1.5, "Accessing the Page Definition File"). OEPE also displays the Edit Binding context menu option when you right-click a method call activity that is associated with a page definition file.

A task flow activity that is associated with a page definition file displays an icon in the lower-right section of the task flow activity icon.

To associate a page definition file with a task flow activity:

1. In the Diagram view, right-click the task flow activity for which you want to create a page definition file and select Create Page Definition from the context menu.
2. In the resulting page definition file, add the bindings that you want your task flow activity to reference at runtime.

For more information about page definition files, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts."

When the preceding steps are completed, OEPE generates a page definition file for the task flow activity at design time. The file name of the page definition file comprises the originating task flow and either the name of the task flow activity or the data control operation to invoke. OEPE also generates an EL expression from the task flow activity to the binding in the created page definition file. At runtime, a binding container ensures that a task flow activity's EL expressions reference the correct value.

11.2.6 How to Define Control Flows

You use the following task flow components to define the control flow in your MAF AMX application feature:

- Control Flow Case (see Section 11.2.6.1, "Defining a Control Flow Case")
- Wildcard Control Flow Rule (see Section 11.2.6.2, "Adding a Wildcard Control Flow Rule")

11.2.6.1 Defining a Control Flow Case

You can create navigation using the Control Flow Case component, which identifies how control passes from one activity to the next. To create a control flow, select **Control Flow Case** from the Palette. Next, connect the Control Flow Case to the source activity, and then to the destination activity. OEPE creates the following after you connect a source and target activity:

- `control-flow-rule`: Identifies the source activity using a `from-activity-id`.
- `control-flow-case`: Identifies the destination activity using a `to-activity-id`.

To define a control flow case directly in the MAF task flow diagram:

1. Open the task flow source file in the Diagram view.

2. Select **Control Flow Case** from the Palette.
3. On the diagram, click a source activity and then a destination activity. OEPE adds the control flow case to the diagram. Each line that OEPE adds between an activity represents a control flow case. The `from-outcome` contains a value that can be matched against values specified in the action attribute of the MAF AMX UI components.
4. To change the `from-outcome`, select the text next to the control flow in the diagram. By default, this is a wildcard character.
5. To change the `from-activity-id` (the identifier of the source activity), or the `to-activity-id` (the identifier for the destination activity), drag either end of the arrow in the diagram to a new activity.

11.2.6.2 Adding a Wildcard Control Flow Rule

MAF task flows support the wildcard control flow rule, which represents a control flow `from-activity-id` that contains a trailing wildcard (`foo*`) or a single wildcard character. You can add a wildcard control flow rule to an unbounded or bounded task flow by dragging and dropping it from the Palette. To configure your wildcard control flow rule, use the Properties window.

To add a wildcard control flow rule:

1. Open the task flow source file in the Diagram view.
2. Select **Wildcard Control Flow Rule** in the Palette and drop it onto the diagram.
3. Select **Control Flow Case** in the Palette.
4. In the task flow diagram, drag the control flow case from the wildcard control flow rule to the target activity, which can be any activity type.
5. By default, the label below the wildcard control flow rule is *. This is the value for the **From Activity ID** element. To change this value, select the wildcard control flow rule in the diagram. In the Properties window for the wildcard control flow rule, enter a new value in the **From Activity ID** field. A useful convention is to cast the wildcard control flow rule in a form that describes its purpose. For example, enter `project*`. The wildcard must be a trailing character in the new label.

   Tip: You can also change the From Activity ID value in the task flow diagram.

6. Optionally, in the Properties window expand the **Behavior** section and write an EL expression in the **If** field that must evaluate to `true` before control can pass to the activity identified by **To Activity ID**.

11.2.6.3 What You May Need to Know About Control Flow Rule Metadata

Example 11–3 shows the general syntax of a control flow rule element in the task flow source file.

Example 11–3 Control Flow Rule Definition

```
<control-flow-rule>
   <from-activity-id>from-view-activity</from-activity-id>
   <control-flow-case>
      <from-action>actionmethod</from-action>
      <from-outcome>outcome</from-outcome>
      <to-activity-id>destinationActivity</to-activity-id>
```

```
      <if>#{myBean.someCondition}</if>
   </control-flow-case>
   <control-flow-case>
   ...
   </control_flow-case>
</control-flow-rule>
```

Control flow rules can consist of the following metadata:

- `control-flow-rule`: a mandatory wrapper element for control flow case elements.
- `from-activity-id`: the identifier of the activity where the control flow rule originates (for example, `source`).

  A trailing wildcard (*) character in `from-activity-id` is supported. The rule applies to all activities that match the wildcard pattern. For example, `login*` matches any logical activity ID name beginning with the literal `login`. If you specify a single wildcard character in the metadata (not a trailing wildcard), the control flow automatically converts to a wildcard control flow rule activity in the diagram. For more information, see Section 11.2.6.2, "Adding a Wildcard Control Flow Rule."
- `control-flow-case`: a mandatory wrapper element for each case in the control flow rule. Each case defines a different control flow for the same source activity. A control flow rule must have at least one control flow case.
- `from-action`: an optional element that limits the application of the rule to outcomes from the specified action method. The action method is specified as an EL binding expression, such as, for example, `#{backing_bean.cancelButton_action}`.

  In Example 11–3, control passes to `destinationActivity` only if outcome is returned from `actionmethod`.

  The value in `from-action` applies only to a control flow originating from a view activity, not from any other activity types. Wildcards are not supported in `from-action`.
- `from-outcome`: identifies a control flow case that will be followed based on a specific originating activity outcome. All possible originating activity outcomes should be accommodated with control flow cases.

  If you leave both the `from-action` and the `from-outcome` elements empty, the case applies to all outcomes not identified in any other control flow cases defined for the activity, thus creating a default case for the activity. Wildcards are not supported in `from-outcome`.
- `to-activity-id`: a mandatory element that contains the complete identifier of the activity to which the navigation is routed if the control flow case is performed. Each control flow case can specify a different `to-activity-id`.
- `if`: an optional element that accepts an EL expression as a value. If the EL expression evaluates to `true` at runtime, control flow passes to the activity identified by the `to-activity-id` element.

11.2.6.4 What You May Need to Know About Control Flow Rule Evaluation

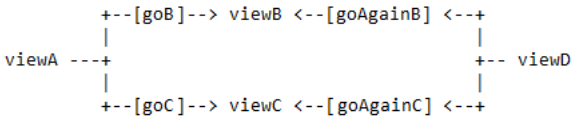
At runtime, task flows evaluate control flow rules from the most specific to the least specific match to determine the next transition between activities. Evaluation is based on the following priority:

1. `from-activity-id`, `from-action`, `from-outcome`: first, searches for a match in all three elements is performed.
2. `from-activity-id`, `from-outcome`: the search is performed in these elements if no match in all three elements is found.
3. `from-activity-id`: if search in the preceding combinations did not result in a match, search is performed in this element only.

11.2.7 What You May Need to Know About MAF Support for Back Navigation

In the task flow example that Figure 11–6 shows, it is possible to take two separate paths to reach **viewD** based on the action outcome value (see Section 11.2.9, "How to Specify Action Outcomes Using UI Components"): either from **viewA** to **viewB** to **viewD**, or from **viewA** to **viewC** to **viewD**.

Figure 11–6 Task Flow with Back Navigation



While you could theoretically keep track of which navigation paths had been followed and then directly implement the `__back` navigation flow, it would be tedious and error-prone, especially considering the fact that due to the limited screen space on mobile devices transitions out of the navigation sequences occur very frequently. MAF provides support for a built-in `__back` navigation that enables moving back through optional paths across a task flow: by applying its "knowledge" of the path taken, MAF performs the navigation back through the same path. For example, if the initial navigation occurred from **viewA** to **viewC** to **viewD**, on exercising the `__back` option on **ViewD** MAF would automatically take the end user back to **ViewA** through **ViewC** rather than through **ViewB**.

For additional information, see the following:

- Section 11.2.3, "What You May Need to Know About the task-flow-definition.xml File"
- Section 11.2.10, "How to Create and Reference Managed Beans"
- Section 12.3.5.7, "Enabling the Back Button Navigation"

11.2.8 How to Enable Page Navigation by Dragging

You can enable navigation from one MAF AMX page to another through the use of the Navigation Drag Behavior operation. For more information, see Section 12.3.28, "How to Enable Drag Navigation."

11.2.9 How to Specify Action Outcomes Using UI Components

Using the Properties window, you can specify an action outcome by setting the `action` attribute of one of the following UI components to the corresponding control flow case `from-outcome` leading to the next task flow activity:

- Command Button (see Section 12.3.5, "How to Use Buttons")
- Command Link (see Section 12.3.6, "How to Use Links")

- List Item

You use the UI component's **Action** field to make a selection from a list of possible action outcomes defined in one or more task flow for a specific MAF AMX page.

A **Back** action (`__back`) is automatically added to every list to enable navigation to the previously visited page.

> **Note:** An MAF AMX page can be referenced in both bounded and unbounded task flows, in which case actions outcomes from both task flows are included in the Action selection list.

11.2.10 How to Create and Reference Managed Beans

You can create and use managed beans in your MAF application to store additional data or execute custom code. You can use usual editing mechanism to reference managed beans and create references to them for applicable fields. For more information, see Section 13.4, "Creating and Using Managed Beans."

Click on the name of the task flow (not the icon) to invoke the **Edit Property** dialog that Figure 11–7 shows.

Figure 11–7 Edit Property Dialog for Action

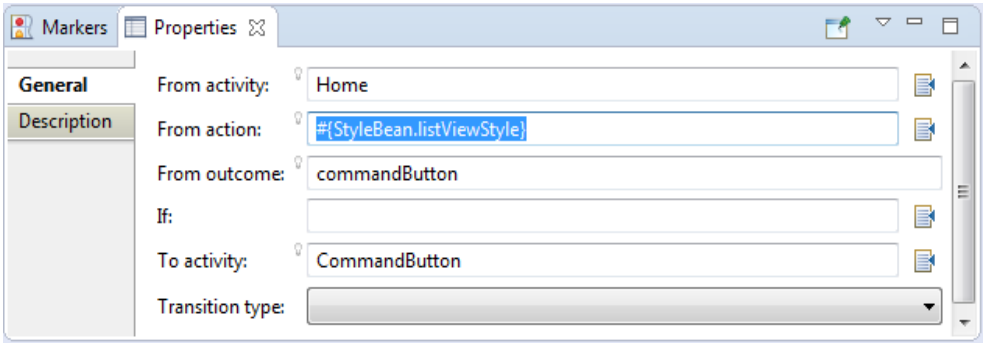


Table 11–5 lists MAF AMX attributes for which the Edit option in the Properties window is available.

Table 11–3 Editable Attributes

Property	Element
`action`	`amx:commandButton`
`action`	`amx:commandLink`
`action`	`amx:listItem`
`action`	`amx:navigationDragBehavior`
`action`	`dvtm:chartDataItem`
`action`	`dvtm:ieDataItem`
`action`	`dvtm:timelineItem`
`action`	`dvtm:area`
`action`	`dvtm:marker`
`actionListener`	`amx:listItem`

Table 11–3 (Cont.) Editable Attributes

Property	Element
actionListener	amx:commandButton
actionListener	amx:commandLink
binding	amx:actionListener
mapBoundsChangeListener	dvtm:geographicMap
mapInputListener	dvtm:geographicMap
moveListener	amx:listView
rangeChangeListener	amx:listView
selectionListener	amx:listView
selectionListener	amx:filmStrip
selectionListener	dvtm:areaDataLayer
selectionListener	dvtm:pointDataLayer
selectionListener	dvtm:treemap
selectionListener	dvtm:sunburst
selectionListener	dvtm:timelineSeries
selectionListener	dvtm:nBox
selectionListener	dvtm:areaChart
selectionListener	dvtm:barChart
selectionListener	dvtm:bubbleChart
selectionListener	dvtm:comboChart
selectionListener	dvtm:horizontalBarChart
selectionListener	dvtm:lineChart
selectionListener	dvtm:funnelChart
selectionListener	dvtm:pieChart
selectionListener	dvtm:scatterChart
valueChangeListener	amx:inputDate
valueChangeListener	amx:inputNumberSlider
valueChangeListener	amx:inputText
valueChangeListener	amx:selectBooleanCheckbox
valueChangeListener	amx:selectBooleanSwitch
valueChangeListener	amx:selectManyCheckbox
valueChangeListener	amx:selectManyChoice
valueChangeListener	amx:selectOneButton
valueChangeListener	amx:selectOneChoice
valueChangeListener	amx:selectOneRadio
valueChangeListener	dvtm:statusMeterGauge
valueChangeListener	dvtm:dialGauge
valueChangeListener	dvtm:ratingGauge

Table 11–3 (Cont.) Editable Attributes

Property	Element
viewportChangeListener	dvtm:areaChart
viewportChangeListener	dvtm:barChart
viewportChangeListener	dvtm:comboChart
viewportChangeListener	dvtm:horizontalBarChart
viewportChangeListener	dvtm:lineChart

Clicking **Edit** for all other properties invokes a similar dialog, but without the Action Outcome option

The preceding dialogs demonstrate that you can either create a managed bean, or select an available action outcome for the action property.

The **Action Outcome** list shown in Figure 11–7 contains the action outcomes from all task flows to which a specific MAF AMX page belongs. In addition, this list contains a `__back` navigation outcome to go back to the previously visited page (see Section 11.2.9, "How to Specify Action Outcomes Using UI Components" for more information). If a page is not part of any task flow, the only available outcome in the Action Outcome list is `__back`. When you select one of the available action outcomes and click OK, the action property value is updated with the appropriate EL expression, such as the following for a `commandButton`:

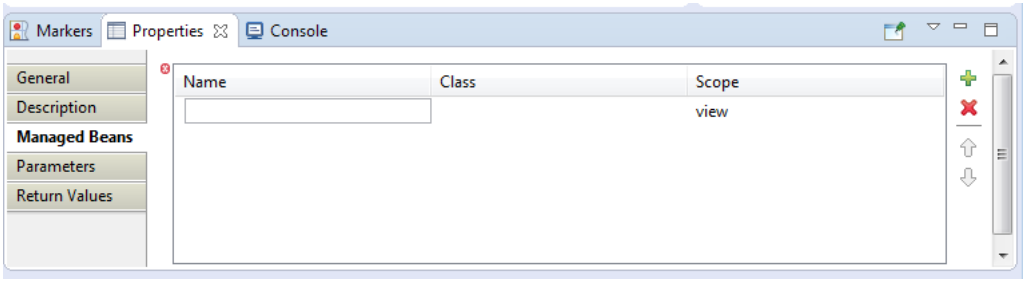
```
<amx:commandButton action="goHome"/>
```

The **Method Binding** option (see Figure 11–7) allows you to either create a new managed bean class or select an existing one.

To create a new managed bean class:

1. Click **New** next to the Managed Bean field to open the Create Managed Bean dialog that Figure 11–8 shows.

Figure 11–8 Create Managed Bean Dialog



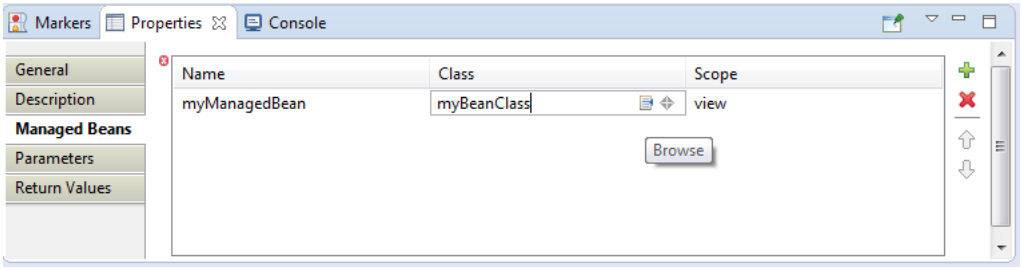
MAF supports the following scopes:

- `application`
- `view`
- `pageFlow`

When you declare a managed bean to an MAF application or the MAF AMX application feature, the managed bean is instantiated and identified in the proper scope, and the bean's properties are resolved and its methods are called through EL. For more information, see Section 13.3, "Creating EL Expressions."

2. Press the Tab key to proceed from the **Name** field to the **Class** name field (see Figure 11–9). You can either type in the class name or click the **Browse** button to select.

Figure 11–9 Setting Managed Bean Name and Class



Example 11–4 shows the newly created managed bean class. The task flow that this MAF AMX page is part of is updated to reference the bean.

Example 11–4 New Managed Bean Class

```
<managed-bean id="__3">
   <managed-bean-name>MyBean</managed-bean-name>
   <managed-bean-class>mobile.MyBean</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

> **Note:** If a given MAF AMX page is part of bounded as well as unbounded task flows, both of these task flows are updated with the managed bean entry.

3. Edit the selected property value with the appropriate EL expression, such as the following for a `commandButton`:

```
<amx:commandButton action="#{MyBean.getMeHome}"/>
```

The managed bean class is also updated to contain the newly created method, as Example 11–5 shows.

Example 11–5 New Method in Managed Bean Class

```
package mobile;

public class MyBean {
   public MyBean() {
   }

   public String getMeHome() {
      // Add event code here...
      return null;
   }
}
```

For more information, see the following:

- Section 13.3.5.2, "About the Managed Beans Category"

11.2.11 How to Specify the Page Transition Style

By defining the page transition style on the task flow, you can specify how MAF AMX pages transition from one view to another. The behavior of your MAF AMX page at transition can be as follows:

- fading in
- sliding in from left
- sliding in from right
- sliding up from bottom
- sliding down from top
- sliding in from start
- sliding in from end
- flipping up from bottom
- flipping down from top
- flipping from left
- flipping from right
- flipping from start
- flipping from end
- none

Sliding in from start and end, as well as flipping from start and end are used on the iOS platform to accommodate the right-to-left text direction. It is generally recommended to use the start and end transition style as opposed to left and right.

> **Note:** You cannot enable flipping on the Android platform.

You set the transition style by modifying the `transition` attribute of the `control-flow-case` (Control Flow Case component), as Example 11–6 shows.

Example 11–6 Setting Transition Style

```
<control-flow-rule id="__1">
   <from-activity-id>products</from-activity-id>
   <control-flow-case id="__2">
      <from-outcome>details</from-outcome>
      <to-activity-id>productdetails</to-activity-id>
      <transition>fade</transition>
   </control-flow-case>
</control-flow-rule>
```

In the Properties window, the `transition` attribute is located under **Behavior**. The default transition style is `slideLeft`.

> **Tip:** When defining the task flow, you should specify the `control-flow-case`'s `transition` value such that it is logical. For example, if the transition occurs from left to right with the purpose of navigating back, then the transition should return to the previous page by sliding right.

11.2.12 What You May Need to Know About Bounded and Unbounded Task Flows

Task flows provide a modular approach for defining control flow in an MAF AMX application feature. Instead of representing an application feature as a single large page flow, you can divide it into a collection of reusable task flows. Each task flow contains a portion of the application feature's navigational graph. The nodes in the task flows represent activities. An activity node represents a simple logical operation such as displaying a page, executing application logic, or calling another task flow. The transitions between the activities are called control flow cases.

There are two types of task flows in MAF AMX:

1. Unbounded Task Flows: a set of activities, control flow rules, and managed beans that interact to allow the end user to complete a task. The unbounded task flow consists of all activities and control flows in an MAF AMX application feature that are not included within a bounded task flow.
2. Bounded Task Flows: a specialized form of task flow that, in contrast to the unbounded task flow, has a single entry point and no exit points. It contains its own collections of activities and control-flow rules, as well as their own memory scope and managed-bean life span.

For a description of the activity types that you can add to unbounded or bounded task flows, see Section 11.2.2, "What You May Need to Know About Task Flow Activities and Control Flows."

A typical MAF AMX application feature contains a combination of one unbounded task flow created at the time when the application feature is created and one or more bounded task flows. At runtime, the MAF application can call bounded task flows from activities that you added to the unbounded task flow.

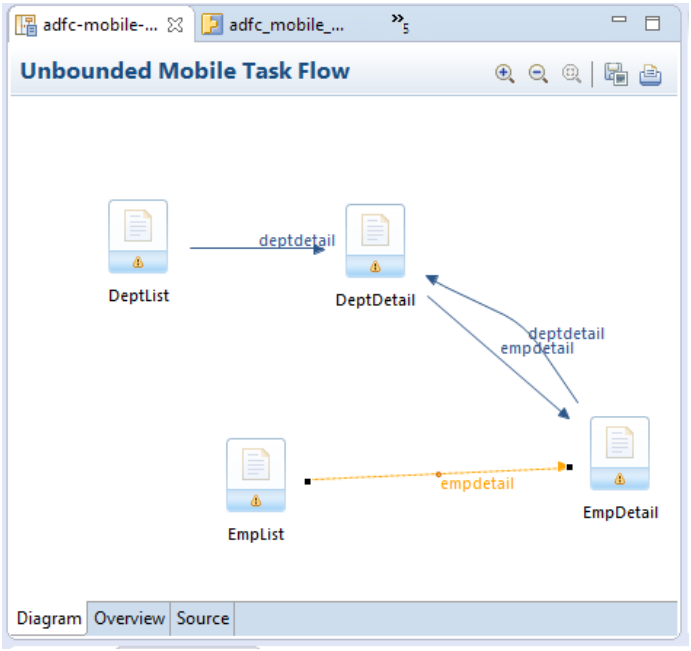
11.2.12.1 Unbounded Task Flows

An MAF AMX application feature always contains one unbounded task flow, which provides one or more entry points to that application feature. An entry point is represented by a view activity. By default, the source file for the unbounded task flow is the `adfc-mobile-config.xml` file.

> **Note:** Although it is possible create additional source files for unbounded task flows, the MAF AMX application feature combines all source files at runtime into the `adfc-mobile-config.xml` file.

Figure 11–10 displays the diagram for an unbounded task flow from an MAF AMX application feature. This task flow contains a number of view activities that are all entry points to the application feature.

Figure 11–10 Unbounded Mobile Task Flow Diagram



Consider using an unbounded task flow if the following applies:

- There is no need for the task flow to be called by another task flow.
- The MAF AMX application feature has multiple points of entry.
- There is no need for a specifically designated activity to run first in the task flow (default activity).

An unbounded task flow can call a bounded task flow, but cannot be called by another task flow.

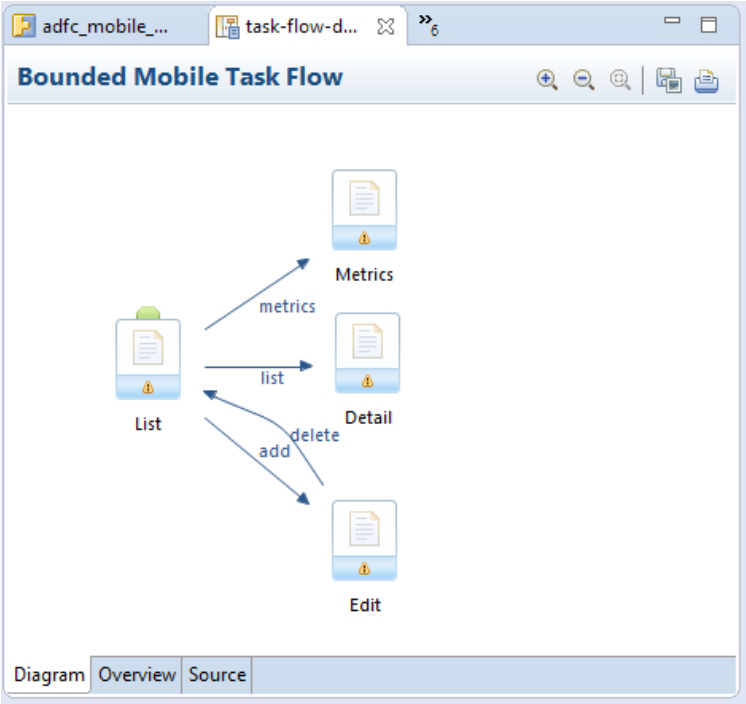
11.2.12.2 Bounded Task Flows

By default, the IDE proposes a file name for the source file of a bounded task flow (see Section 11.2.1, "How to Create a Task Flow"). You can modify this file name to reflect the purpose of the task to be performed.

A bounded task flow can call another bounded task flow, which can call another, and so on. There is no limit to the depth of the calls.

Figure 11–11 displays the diagram for a bounded task flow from an MAF AMX application feature.

Figure 11–11 Bounded Mobile Task Flow Diagram



The following are reasons for creating a bounded task flow:

- The bounded task flow always specifies a default activity, which is a single point of entry that must execute immediately upon entry of the bounded task flow.
- It is reusable within the same or other MAF AMX application features.
- Any managed beans you use within a bounded task flow can be specified in a page flow scope, making them isolated from the rest of the MAF AMX application feature. These managed beans (with page flow scope) are automatically released when the task flow completes.

The following is a summary of the main characteristics of a bounded task flow:

- Well-defined boundary: a bounded task flow consists of its own set of private control flow rules, activities, and managed beans. A caller requires no internal knowledge of page names, method calls, child bounded task flows, managed beans, and control flow rules within the bounded task flow boundary. Data controls can be shared between task flows.
- Single point of entry: a bounded task flow has a single point of entry—a default activity that executes before all other activities in the task flow.
- Page flow memory scope: you can specify page flow scope as the memory scope for passing data between activities within the bounded task flow. Page flow scope defines a unique storage area for each instance of a bounded task flow. Its lifespan is the bounded task flow, which is longer than request scope and shorter than session scope.
- Addressable: you can access a bounded task flow by specifying its unique identifier within the XML source file for the bounded task flow and the file name of the XML source file.

- Reusable: you can identify an entire group of activities as a single entity, a bounded task flow, and reuse the bounded task flow in another MAF AMX application feature within an MAF application.

  You can also reuse an existing bounded task flow by calling it.

  In addition, you can use task flow templates to capture common behaviors for reuse across different bounded task flows.

- Parameters and return values: a caller can pass input parameters to a bounded task flow and accept return values from it (see Section 11.2.12.3.1, "Passing Parameters to a Bounded Task Flow" and Section 11.2.12.3.2, "Configuring a Return Value from a Bounded Task Flow").

  In addition, you can share data controls between bounded task flows.

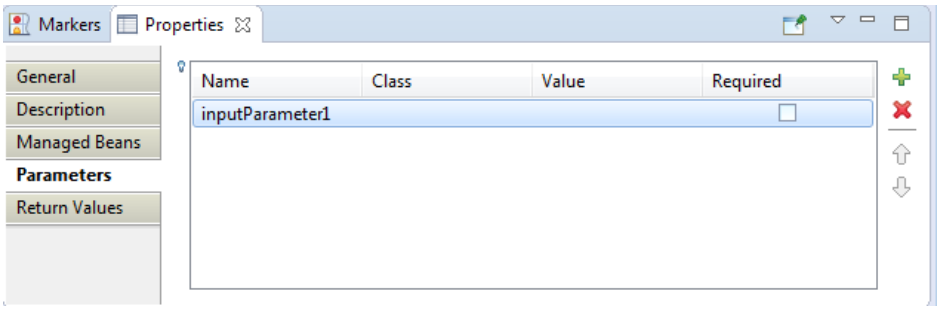
- On-demand loading of metadata: bounded task flow metadata is loaded on demand when entering a bounded task flow.

11.2.12.3 Using Parameters in Task Flows

A task flow´s ability to accept input parameters and return parameter values allows you to manipulate data in task flows and share data between task flows. Using these abilities, you can optimize the reuse of task flows in your MAF AMX application feature.

Figure 11–12 shows a task flow that specifies an input parameter definition to hold information about a user in a pageFlow scope.

Figure 11–12 Input Parameters in Task Flow



You can specify parameter values using standard EL expressions if you call a bounded task flow using a task flow call activity. For example, you can specify parameters using the following syntax for EL expressions:

```
#{bindings.bindingId.inputValue}
#{CustomerBean.zipCode}
```

Appending `inputValue` to the EL expression ensures that you assign to the parameter the value of the binding rather than the actual binding object.

11.2.12.3.1 Passing Parameters to a Bounded Task Flow

A called bounded task flow can accept input parameters from the task flow that calls it or from a task flow binding.

To pass an input parameter to a bounded task flow, you specify one or more of the following:

- Input parameters on the task flow call activity in the calling task flow: input parameters specify where the calling task flow stores parameter values.

- Input parameter definitions on the called bounded task flow: input parameter definitions specify where the called bounded task flow can retrieve parameter values at runtime.

Specify the same name for the input parameter that you define on the task flow call activity in the calling task flow and the input parameter definition on the called bounded task flow. Do this so you can map input parameter values to the called bounded task flow.

If you do not specify an EL expression to reference the value of the input parameter, the EL expression for value defaults to the following at runtime:

```
#{pageFlowScope.parmName}
```

where *parmName* is the value you entered for the input parameter name.

In an input parameter definition for a called bounded task flow, you can specify an input parameter as required. If the input parameter does not receive a value at runtime or design time, the task flow raises a warning in a log file of the MAF application that contains the task flow. An input parameter that you do not specify as required can be ignored during task flow call activity creation.

Task flow call activity input parameters can be passed by reference or passed by value when calling a task flow using a task flow call activity (see Section 11.2.5.4.2, "Specifying Input Parameters on a Task Flow Call Activity"). By default, primitive types (for example, `int`, `long`, or `boolean`) are passed by value (`pass-by-value`).

A called task flow can return values to the task flow that called it when it exits. For more information, see Section 11.2.12.3.2, "Configuring a Return Value from a Bounded Task Flow."

When passing an input parameter to a bounded task flow, you define values on both the calling task flow and the called task flow.

Before you begin:

- Create a calling and called task flow: the calling task flow can be bounded or unbounded. The called task flow must be bounded. For more information about creating task flows, see Section 11.2.1, "How to Create a Task Flow."
- Add a task flow call activity to the calling task flow.

Figure 11–13 shows an example where the view activity passes control to the task flow call activity.

Figure 11–13 Calling Task Flow

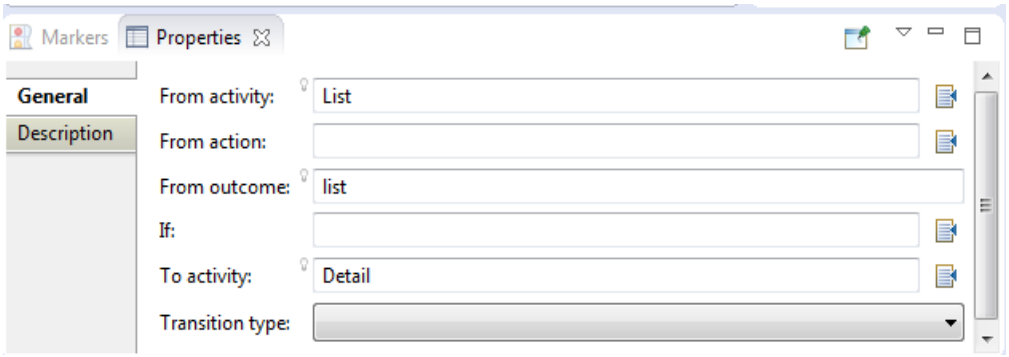
To pass an input parameter to a bounded task flow:

1. Open an MAF AMX page that contains an input component where the end user enters a value that is passed to a bounded task flow as a parameter at runtime. Note that the MAF AMX page that you open should be referenced by a view activity in the calling task flow.

2. Select an input text component on the MAF AMX page where the end user enters a value at runtime.
3. In the Properties window, expand the Common section and enter a value for the input text component in the **Value** field.

   You can specify the value as an EL expression (for example, `#{pageFlowScope.inputValue}`), either manually or using the Expression Builder.
4. Open the task flow that is to be called by double-clicking it in the Project Explorer, then switch the view to the Overview tab and select the **Parameters** navigation tab.
5. In the **Input Parameter Definition** section, click Add (+) to specify a new entry (see Figure 11–12):
   - In the **Name** field, enter a name for the parameter (for example, `inputParm1`).
   - In the **Value** field, enter an EL expression where the parameter value is stored and referenced (for example, `#{pageFlowScope.inputValue}`), either manually or using the Expression Builder.
6. In the Project Explorer, double-click the calling task flow that contains the task flow call activity to invoke the called bounded task flow.
7. In the Project Explorer, drag the called bounded task flow and drop it on top of the task flow call activity that is located in the diagram of the calling task flow. This automatically creates a task flow reference to the bounded task flow. As shown in Figure 11–14, the task flow reference contains the following:
   - The bounded task flow ID (`id`): an attribute of the bounded task flow's `task-flow-definition` element.
   - The document name that points to the source file for the task flow that contains the ID.

Figure 11–14 Task Flow Reference



8. In the Properties window for the task flow call activity, expand the **Parameters** section to view the **Input Parameters** section.
   - Enter a name that identifies the input parameter: since you dropped the bounded task flow on a task flow call activity having defined input parameters, the name should already be specified. You must keep the same input parameter name.
   - Enter a parameter value (for example, `#{pageFlowScope.param1}`): the value on the task flow call activity input parameter specifies where the calling task flow stores parameter values. The value on the input parameter definition for

the called task flow specifies the location from which the value is to be retrieved for use within the called bounded task flow once it is passed.

At runtime, the called task flow can to use the input parameter. If you specified `pageFlowScope` as the value in the input parameter definition for the called task flow, you can use the parameter value anywhere in the called bounded task flow. For example, you can pass it to a view activity on the called bounded task flow.

Upon completion, OEPE writes entries to the source files for the calling task flow and called task flow based on the values that you select.

Example 11–7 shows an input parameter definition specified on a a bounded task flow.

Example 11–7 Input Parameter Definition

```
<task-flow-definition id="sourceTaskflow">
...
   <input-parameter-definition>
      <name>inputParameter1</name>
      <value>#{pageFlowScope.parmValue1}</value>
      <class>java.lang.String</class>
   </input-parameter-definition>
...
</task-flow-definition>
```

Example 11–8 shows the input parameter metadata for the task flow call activity that calls the bounded task flow shown in Example 11–7. At runtime, the task flow call activity calls the bounded task flow and passes it the value specified by its value element.

Example 11–8 Input Parameter on Task Flow Call Activity

```
<task-flow-call id="taskFlowCall1">
...
   <input-parameter>
      <name>inputParameter1</name>
      <value>#{pageFlowScope.newCustomer}</value>
      <pass-by-value/>
   </input-parameter>
...
</task-flow-call>
```

11.2.12.3.2 Configuring a Return Value from a Bounded Task Flow

You configure a return value definition on the called task flow and add a parameter to the task flow call activity in the calling task flow that retrieves the return value at runtime.

Before you begin:

Create a bounded or unbounded task flow (calling task flow) and a bounded task flow (called task flow). For more information, see Section 11.2.1, "How to Create a Task Flow."

To configure a return value from a called bounded task flow:

1. Open the task flow that is to be called by double-clicking it in the Project Explorer, then switch the view to the Overview tab and select the **Parameters** navigation tab.
2. In the **Return Value Definitions** section, click Add (+) to define a return value (see Figure 11–12):

- In the **Name** field, enter a name to identify the return value (for example, `returnValue1`).
- In the **Class** field, enter a Java class that defines the data type of the return value. The default value is `java.lang.String`.
- In the **Value** field, enter an EL expression that specifies from where to read the return value (for example, `#{pageFlowScope.ReturnValueDefinition}`), either manually or using the Expression Builder.

3. In the Project Explorer, double-click the calling task flow.
4. With the task flow page open in the Diagram view, select **Components > Activities** from the Palette, and then drag and drop a task flow call activity onto the diagram.
5. In the Properties window for the task flow call activity, expand the **Parameters** section, click Add (+) for the Return Values entry, and then add values as follows to define a return value:
   - A name to identify the return value (for example, returnValue1). It must match the value you entered for the Name field when you defined the return value definition in step 2.
   - A value as an EL expression that specifies where to store the return value (for example, `#{pageFlowScope.ReturnValueDefinition}`). It must match the value you entered for the Value field when you defined the return value definition in step 2.

Upon completion, OEPE writes entries to the source files for the calling task flows that you configured.

Example 11–9 shows an example entry that OEPE writes to the source file for the calling task flow.

Example 11–9 Metadata in the Calling Task Flow to Configure a Return Value

```
<task-flow-call id="taskFlowCall1">
   <return-value id="__3">
      <name id="__4">returnValue1</name>
      <value id="__2">#{pageFlowScope.ReturnValueDefinition}</value>
   </return-value>
</task-flow-call>
```

Example 11–10 shows an example entry that OEPE writes to the source file for the called task flow.

Example 11–10 Metadata in the Called Task Flow to Configure a Return Value

```
<return-value-definition id="__2">
   <name id="__3">returnValue1</name>
   <value>#{pageFlowScope.ReturnValueDefinition}/</value>
   <class>java.lang.String</class>
</return-value-definition>
```

At runtime, the called task flow returns a value. If configured to do so, the task flow call activity in the calling task flow retrieves this value.

11.3 Creating Views

When you create a MAF application, OEPE automatically creates a view project to go with the assembly project and application project. You can also create additional view projects. The following sections discuss these important aspects of creating and working with views:

- Getting familiar with the MAF AMX page structure (see Section 11.3.1.1, "Interpreting the MAF AMX Page Structure")
- Editing and previewing an MAF AMX page (see Section 11.3.1.4, "Using UI Editors")
- Dragging and dropping components onto an MAF AMX page (see Section 11.3.2.1, "Adding UI Components")
- Adding data controls to a view (see Section 11.3.2.3, "Adding Data Controls to the View")

11.3.1 How to Work with MAF AMX Pages

An MAF AMX page is represented by an XML file.

11.3.1.1 Interpreting the MAF AMX Page Structure

The following is a basic structure of the MAF AMX file:

```
<amx:view>
   <amx:panelPage id="pp1">
      <amx:facet name="header">
         <amx:outputText id="ot1" value="Welcome"/>
      …
      </amx:facet>
   </amx:panelPage>
</amx:view>
```

With the exception of data visualization components (see Section 12.5, "Providing Data Visualization"), UI elements are declared under the `<amx>` namespace.

For more information, see Section 11.3.1.3, "What Happens When You Create an MAF AMX Page."

11.3.1.2 Creating MAF AMX Pages

MAF AMX files are contained in the View project of the MAF application. You create these files using the New MAF Page dialog.

MAF offers two alternative ways of creating an MAF AMX page:

- From the New menu (**File > New > MAF Page**)
- By right-clicking the ViewContent folder of the View project and selecting **New > MAF Page**

Before you begin:

To create an MAF AMX page, the MAF application must include a View project folder (see Chapter 2, "Getting Started with MAF Application Development"). Files created outside the ViewContent folder of the View project will not be included in the deployed application.

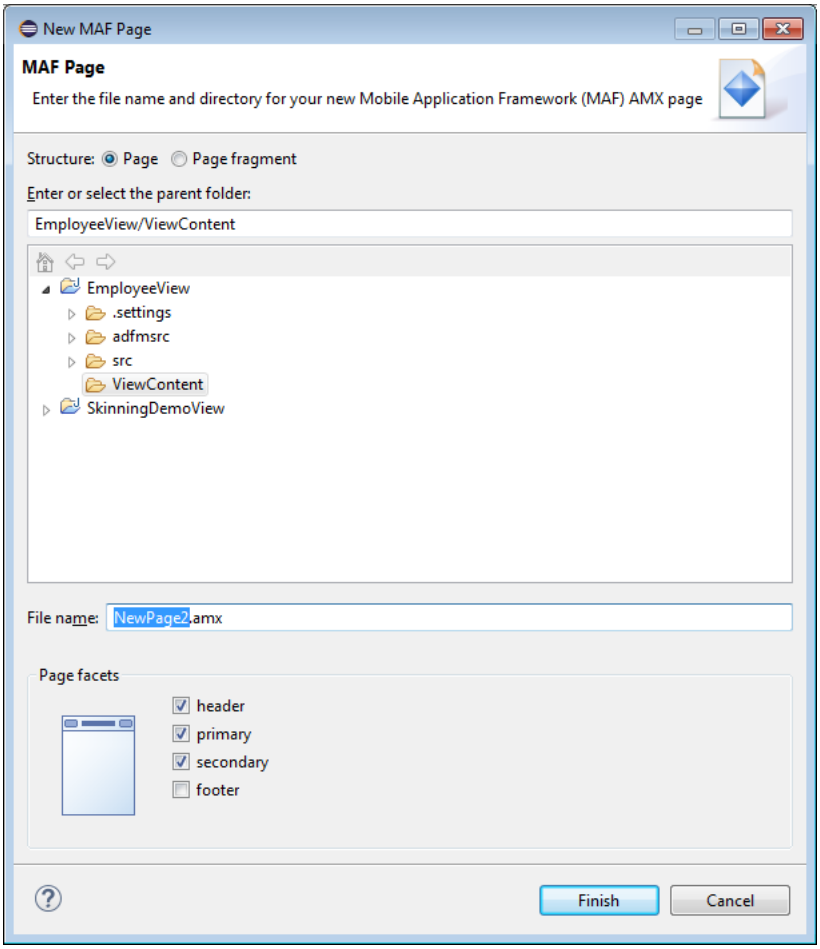
To create an MAF AMX page from the file menu:

1. From the top-level menu in OEPE, click **File**, and then select **New > MAF Page**.
2. In the **New MAF Page** dialog, browse to the View folder for the MAF application for which you are creating the AMX page. (Note that you may have more than one View project in an application.)
3. Enter a name (or accept the default) and, if needed, a location for your new file.
4. Optionally, you may select which facets your new MAF AMX page will include as a part of the page layout:
   - Header
   - Primary
   - Secondary
   - Footer

   For more information, see Section 11.3.1.3, "What Happens When You Create an MAF AMX Page" and Section 12.2.7, "How to Use a Facet Component."

   Note that when you select or deselect a facet, the image representing the page changes dynamically to reflect the changing appearance of the page.

Figure 11–15 Create New MAF Page Dialog



Note: MAF retains your page facet selection and applies it to each subsequent invocation of the New MAF AMX Page dialog.

5. Click **Finish** on the New MAF AMX Page dialog.

To create an MAF AMX page from a View component of the task flow:

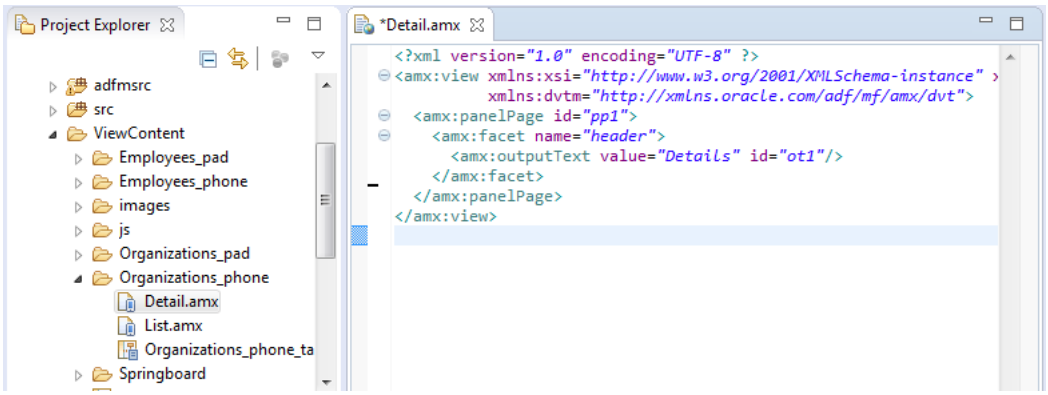
1. Open a task flow file in the diagrammer (see Section 11.2.1, "How to Create a Task Flow" and Section 11.2.4, "What You May Need to Know About the MAF Task Flow Diagrammer")
2. Double-click a View component of the task flow to open the Create MAF AMX Page dialog that Figure 11–15 shows, and then enter a name and, if needed, a location for your new file. Click **OK**.

11.3.1.3 What Happens When You Create an MAF AMX Page

When you use the Create MAF AMX Page dialog to create an MAF AMX page, OEPE creates the physical file and adds it to the `ViewContent` directory of the View Controller project.

In the Project Explorer that Figure 11–16 shows, the `ViewContent` node contains a newly created MAF AMX file called `department.amx`.

Figure 11–16 MAF AMX File in Project Explorer



OEPE also adds the code necessary to import the component libraries and display a page. This code is illustrated in the Source editor shown in Figure 11–16.

If you create a page with all the facets selected, note the following:

- The header is created with an Output Text component because this component is typically used for the page title.
- The primary and secondary actions are created with Button components because it is a typical pattern.
- Since there is no single dominant pattern for the footer, it is created with an Output Text component by default because that component is used in some patterns and it prevents OEPE from generating the initial code with audit violation.
- Adding either the primary or secondary action without adding the header facet still causes the header section to appear in the Page Facets section of Create MAF AMX Page dialog.

Figure 11–17 shows the Page Facet section of the Create MAF AMX Page dialog without any facets selected and Figure 11–18 shows the Preview pane with the generated MAF AMX code.

Figure 11–17 Creating MAF AMX Page Without Selected Facets

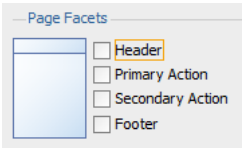
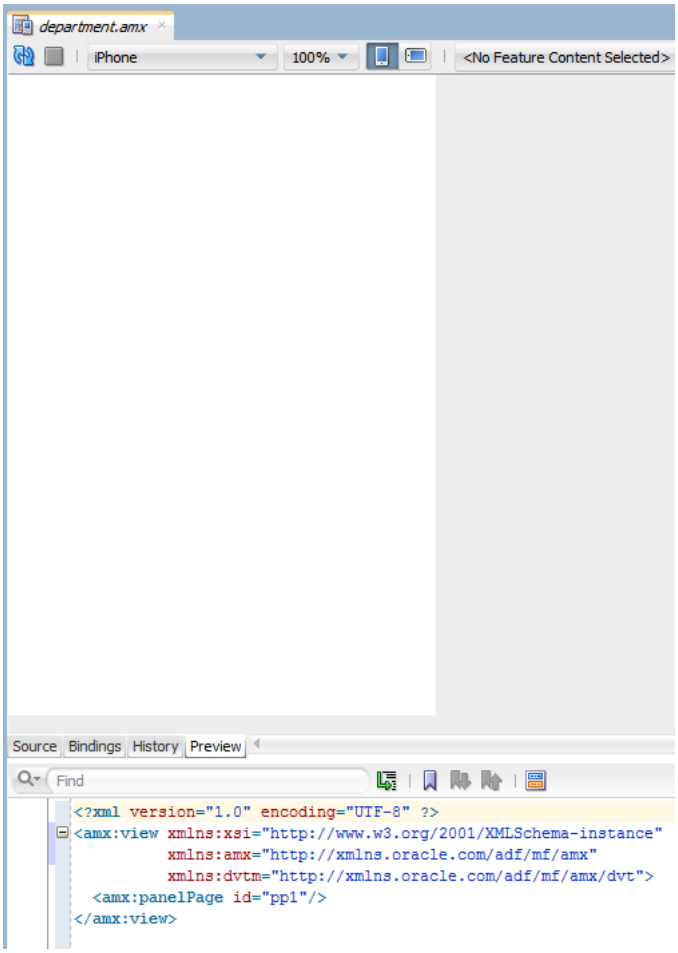


Figure 11–18 MAF AMX Page Without Facets



11.3.1.4 Using UI Editors

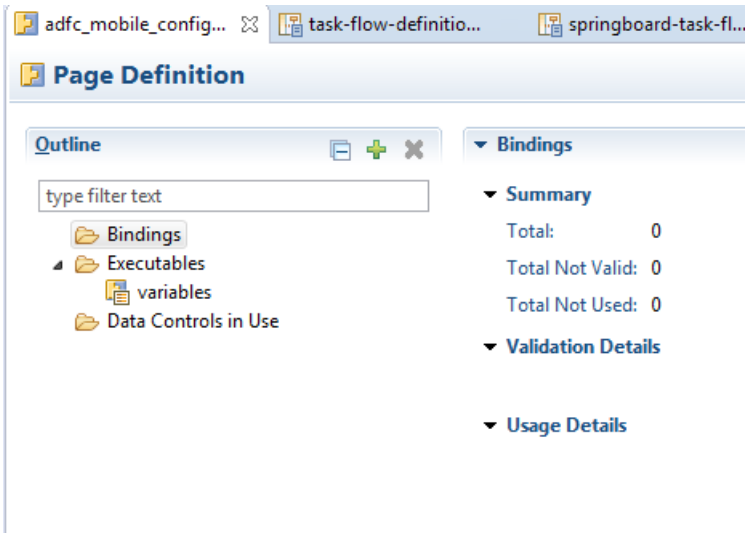
> **Note:** This feature is only available in the OEPE runtime and must be performed manually. It is not available as a design-time feature.

11.3.1.5 Accessing the Page Definition File

MAF AMX supports OEPE's Go to Page Definition functionality that enables you to navigate to the MAF AMX page definition (see Figure 11–19 and Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts") by using a

context menu that allows you to locate and edit the binding information quickly.

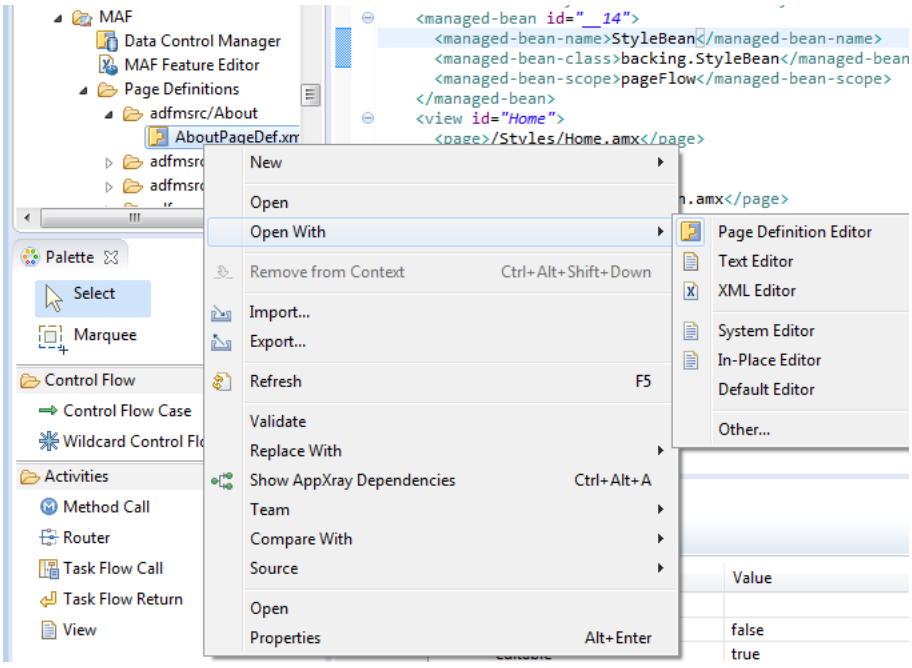
Figure 11–19 Page Definition editor



You can invoke the context menu that contains the Go to Page Definition option from the following:

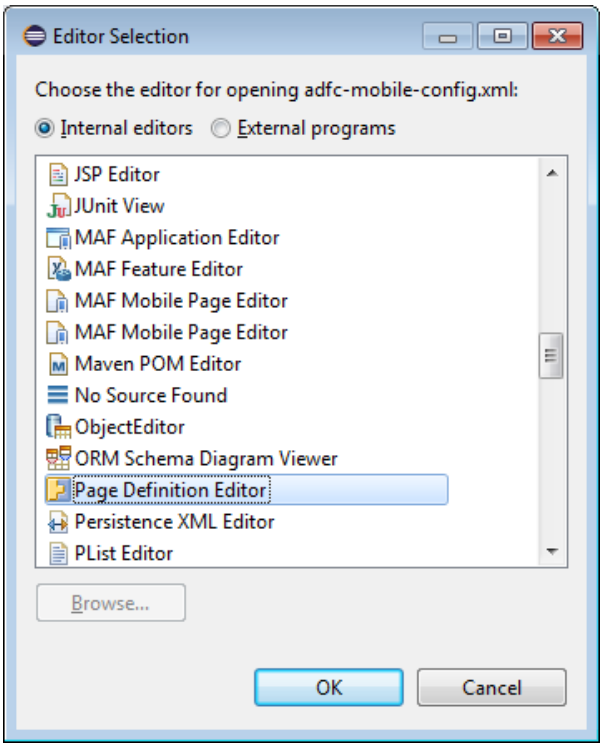
- Source editor, as Figure 11–20 shows.

Figure 11–20 Go to Page Definition from Source Editor



- Application window, by right-clicking the page, selecting **Open With > Other**, and then selecting **Page Definition Editor**, as Figure 11–21 shows.

Figure 11–21 Go to Page Definition from Application Window



11.3.1.6 Sharing the Page Contents

You can enable sharing of contents of MAF AMX pages. Fragment (`fragment`) is a dynamic declarative component that allows for reusable parts of an MAF AMX page elements, including attributes and facets, to be inserted into the content represented by a template. This enables you to standardize the look and feel of your application by reusing the Fragment template across various pages within the application.

You can drag and drop an MAF AMX fragment file (`.amxf`) onto an MAF AMX page or another fragment file to create a reference to the fragment and to define its attributes. The fragment file resides inside your project and you can drop it from the Project Explorer window into a target source file that is currently open in the Source editor.

Before you begin:

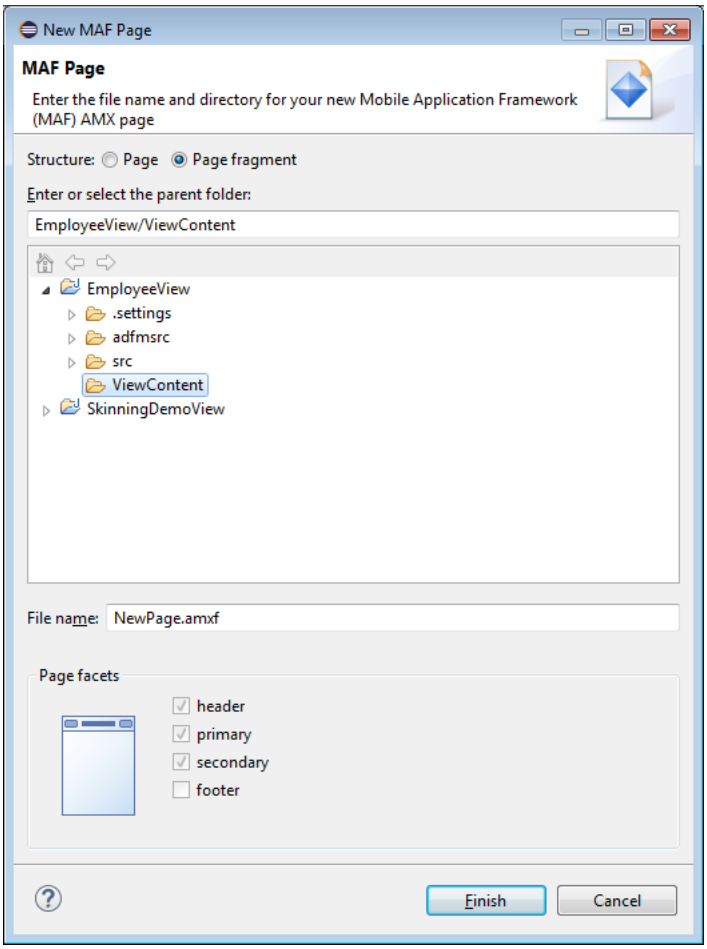
Ensure that the MAF application includes a View project.

If the View project does not contain an MAF AMX page or MAF AMX page task flow from which to create a page, you can invoke the Create MAF AMX Page dialog by double-clicking a view icon in a task flow diagram (see Section 11.3.1.2, "Creating MAF AMX Pages").

To create a Fragment from the File menu:

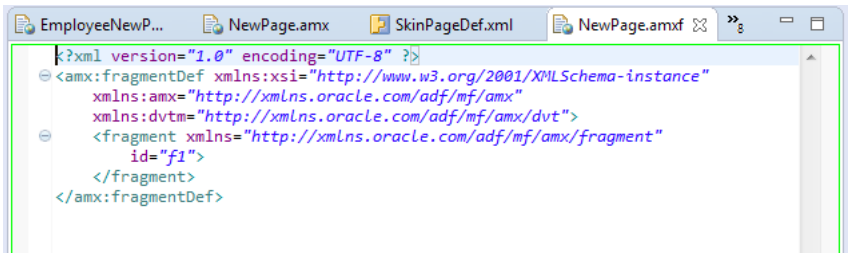
1. From the top-level menu in OEPE, click **File**, and then select **New > MAF Page**.
2. In the **New MAF Page** dialog, click to select the **Page Fragment** button. Open the **View Content** node, enter a descriptive file name if you prefer, and then click **Finish** (see Figure 11–22). Note that the facets are not available for selection in a page fragment.

Figure 11–22 Creating New Fragment



3. Upon completion of the dialog, a newly created file opens in the Source editor of OEPE (see Figure 11–23).

Figure 11–23 Fragment File



Example 11–11 shows an MAF AMX fragment file called `fragment1.amxf`.

Example 11–11 Fragment Definition

```
<amx:fragmentDef
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
   xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <fragment xmlns="http://xmlns.oracle.com/adf/mf/amx/fragment" id="f1">
      <description id="d1">Description of the fragment</description>
```

```
      <facet id="f2">
         <description id="d4">Description of the facet</description>
         <facet-name id="f3">facet1</facet-name>
      </facet>
      <attribute id="a1">
         <description id="d2">Description of an attribute</description>
         <attribute-name id="a2">text</attribute-name>
         <attribute-type id="at1">String</attribute-type>
         <default-value id="d3">defaultValue</default-value>
      </attribute>
   </fragment>
   <amx:panelGroupLayout id="pgl1">
      <amx:facetRef facetName="facet1" id="fr1"/>
      <amx:outputText value="#{text}" id="ot1"/>
   </amx:panelGroupLayout>
</amx:fragmentDef>
```

To include the contents of the fragment in the MAF AMX page, you create a Fragment component (see Section 12.2.16, "How to Use the Fragment Component") and set its `src` attribute to the fragment file of your choice. Example 11–12 shows a `fragment` element added to an MAF AMX page. This element points to the `fragment1.amxf` as its page contents. At the same time, the `facetRef` element, which corresponds to the Facet Definition MAF AMX component, points to `facet1` as its `facet` (MAF AMX Facet component). The `facetRef` element can only be specified in the `.amxf` file within the `fragmentDef`. You can pass attributes to the `facetRef` by specifying the MAF AMX `attribute` element as its child, which allows you to pass an EL variable from the Fragment to a Facet through the `attribute`'s value.

Example 11–12 Fragment in MAF AMX Page

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:panelPage id="pp1">
      <amx:panelGroupLayout layout="vertical"
                            id="itemPgl"
                            styleClass="amx-style-groupbox">
         <amx:fragment id="f1"
                       src="/simpleFragment.amxf"
            <amx:attribute id="a1"
                           name="text"
                           value="defaultValue" />
            <amx:facet name="facet">
               <amx:outputText id="ot5" value="Fragment"/>
            </amx:facet>
         </amx:fragment>
      </amx:panelGroupLayout>
   </amx:panelPage>
</amx:view>
```

The Fragment supports the following:

- Embedded popups (see Section 12.2.8, "How to Use a Popup Component").
- Reusable user interface that can be placed on one or more other parent pages or fragments. This allows you to create a component that is composed of other components without bindings.

- Definition of its own facets. This allows you to create a component such as a layout component that defines a header facet, summary facet, and detail facet, with each facet having its own style class as well as look and feel.
- Data model with both attributes and collections.

MAF sample applications called FragmentDemo and CompGallery demonstrate how to create and use the fragments. These sample applications are available from **File** > **New** > **MAF Examples**.

11.3.2 How to Add UI Components and Data Controls to an MAF AMX Page

After you create an MAF AMX page, you can start adding MAF AMX UI components and data controls to your page.

11.3.2.1 Adding UI Components

You can use the Palette to drag and drop MAF AMX components and MAF AMX data visualization components onto the page. OEPE then adds the necessary declarative page code and sets certain values for component attributes.

The Palette displays MAF AMX components by categories:

- Control Flow
- Activities

For information on adding and using specific components, see Section 12.3, "Creating and Using UI Components."

The Palette also displays MAF AMX data visualization components by categories:

- Common, with the following subcategories:
  - Chart
  - Gauge
  - Map
  - Miscellaneous
- Shared Child Tags
- Other Type-Specific Child Tags, with the following subcategories:
  - Chart
  - Gauge
  - NBox
  - Thematic Map
  - Timeline
  - Sunburst and Treemap

Before you begin:

The MAF application must include a View project, which may or may not contain an MAF AMX page or MAF AMX page task flow from which to create a page.

As described in Section 11.3.1.2, "Creating MAF AMX Pages," you can invoke the Create MAF AMX Page dialog by double-clicking a view icon in a task flow diagram or by selecting **New > MAF Page**.

To add UI components to a page:

1. Open an MAF AMX page in the Source editor (default).
2. In the Tags panel of the Palette, select **MAF AMX**.

   Tip: If the Palette is not displayed (because it has been closed), choose **Window > New Window** from the main OEPE menu. This creates a new instance of OEPE that displays the Palette. By default, the Palette is displayed in the lower left-hand corner of OEPE.

3. Select the component you wish to use, and then drag and drop it onto the Source editor or Structure window. You cannot drop components onto the Preview pane.

   Note: When building an MAF AMX page, you can only drop UI components into UI containers such as, for example, a Panel Group Layout.

   OEPE redraws the page in the Preview pane with the newly added component.

11.3.2.2 Using the Preview

Note: This feature is only available in the OEPE runtime and must be performed manually. It is not available as a design-time feature.

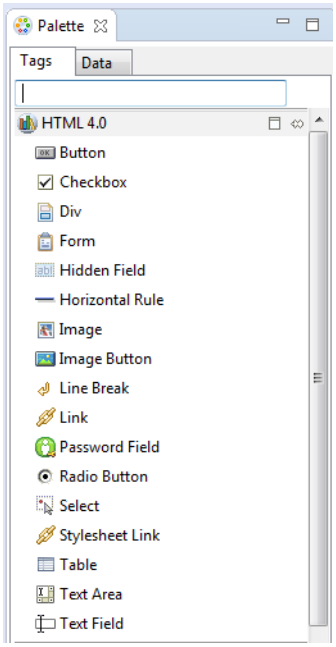
11.3.2.3 Adding Data Controls to the View

You can create databound UI components in an MAF AMX view by dragging data control elements from the Data Controls window and dropping them into either the Structure window or the Source editor. When you drag an item from the Data Controls window to either of these places, OEPE invokes a context menu of default UI components available for the item that you dropped. When you select the desired UI component, OEPE inserts it into an MAF AMX page. In addition, OEPE creates the binding information in the associated page definition file. If such file does not exist, then OEPE creates one. MAF provides a visual indicator for dropping data controls to inform you of the location of the new data control

Note: A data control can only be dropped at a location allowed by the underlying XML schema.

Depending on the approach you take, you can insert different types of data controls into the Structure window of an MAF AMX page. The data controls available from the Tags pane of the Palette are shown in Figure 11–24.

Figure 11–24 HTML Control Tags from the Palette



Dropping an attribute of a collection lets you create various input and output components. You can also create Button and Link components by dropping a data control operation on a page.

The respective action listener is added in the MAF AMX Button for each of these operations.

The data control attributes and operations can be dropped as one or more of the following MAF AMX UI components (see Section 12.3, "Creating and Using UI Components"):

- Button
- Link
- Input Date
- Input Date with Label
- Input Text
- Input Text with Label
- Output Text
- Output Text with Label
- Iterator
- List Item
- List View
- Select Boolean Checkbox
- Select Boolean Switch
- Select One Button
- Select One Choice

- Select One Radio
- Select Many Checkbox
- Select Many Choice
- Convert Date Time
- Convert Number
- Form
- Read Only Form
- Parameter Form

The following Date and Number types are supported:

- `java.util.Date`
- `java.sql.Timestamp`
- `java.sql.Date`
- `java.sql.Time`
- `java.lang.Number`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Float`
- `java.lang.Double`

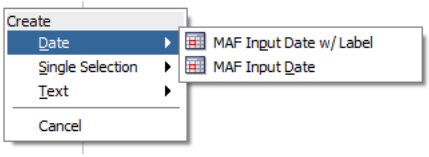
For information on how to use the Data Controls window in OEPE, see Section 13.6, "Creating Databound UI Components from the Data Controls Palette."

11.3.2.3.1 Dragging and Dropping Attributes If your MAF AMX page already contains a Panel Form Layout component or does not require to have all the fields added, you can drop individual attributes from a data control. Depending on the attributes types, different data binding menu options are provided as follows:

Date

This category provides options for creating MAF Input Date and MAF Input Date with Label controls. Figure 11–25 shows the context menu for adding date controls that appears when you drag an attribute from the Data Controls window onto the Source editor or Structure window of an MAF AMX page.

Figure 11–25 Context Menu for Date Controls



Single Selection

This category provides options for creating the following controls:

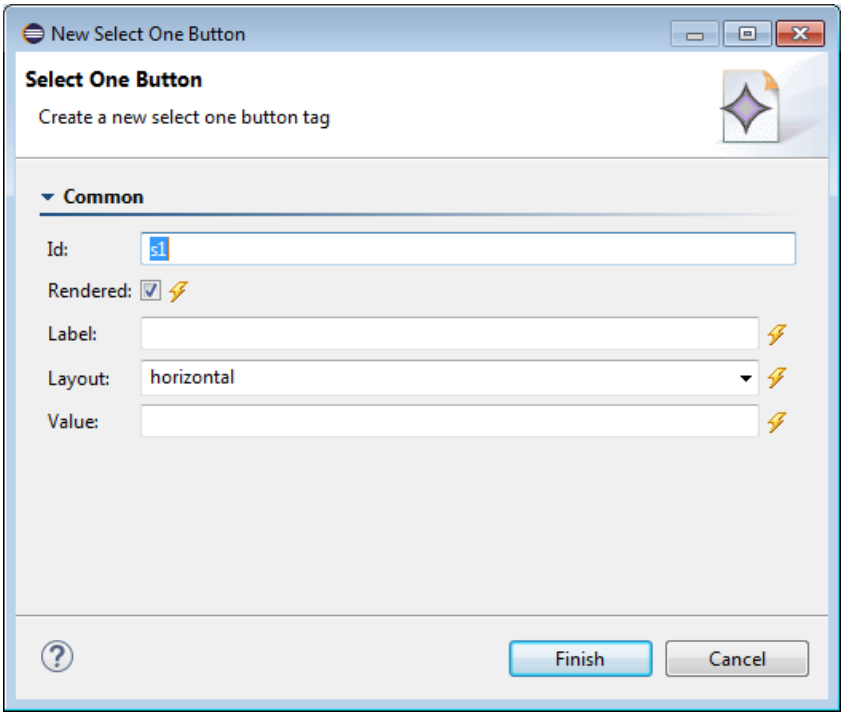
- MAF Select One Button
- MAF Select One Choice

- MAF Select One Radio
- MAF Select Boolean Checkbox
- MAF Select Boolean Switch

If you are working with an existing MAF AMX page and you select **MAF Select One Button** or **MAF Select One Choice** option, an appropriate version of the Edit List Binding dialog is displayed (see Figure 11–26). If you drop a control onto a completely new MAF AMX page, the Edit Action Binding dialog opens instead. After you click OK, the Edit List Binding dialog opens.

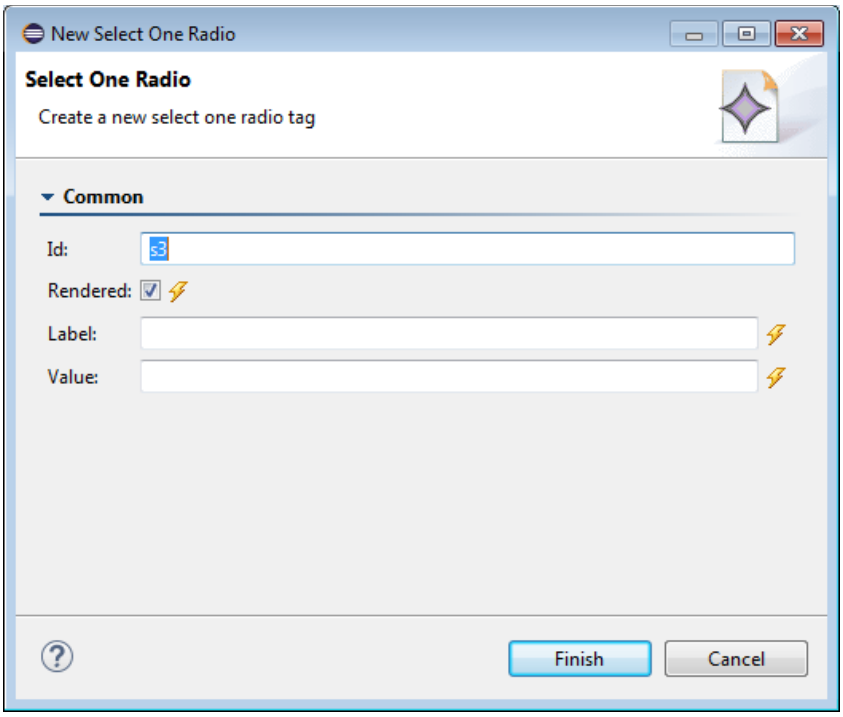
Note: The Edit List Binding or Edit Boolean Binding dialog appears every time you drop any data control attributes as any of the single selection or boolean selection components, respectively.

Figure 11–26 Edit List Binding Dialog for Select One Button and Choice Controls



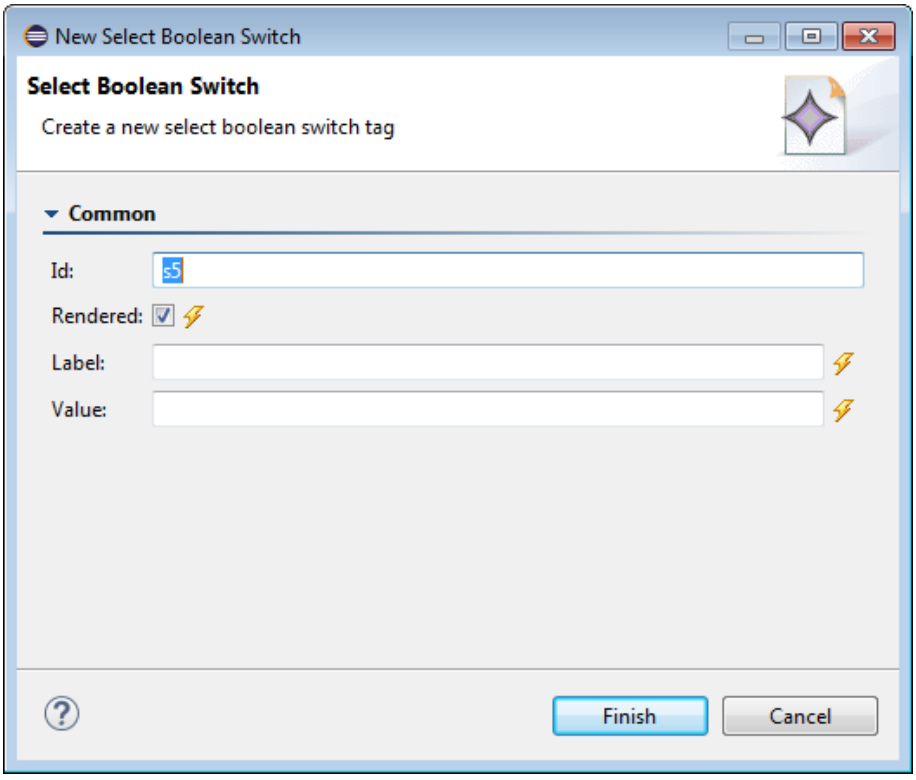
If you select **MAF Select One Radio** option, another version of the Edit List Binding dialog is displayed, as shown in Figure 11–27.

Figure 11–27 Edit List Binding Dialog for Select One Radio Control



If you select **MAF Select Boolean Checkbox** or **MAF Select Boolean Switch** option, another version of the Edit List Binding dialog is displayed, as shown in Figure 11–28.

Figure 11–28 Edit List Binding Dialog for Select Boolean Checkbox and Switch Controls



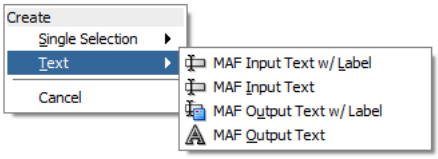
Text

This category provides options for creating the following controls:

- MAF Input Text
- MAF Input Text with Label
- MAF Output Text
- MAF Output Text with Label

Figure 11–29 shows the context menu for adding text controls that appears when you drag an attribute from the Data Controls window onto the Source editor or Structure window of an MAF AMX page.

Figure 11–29 Context Menu for Text Controls



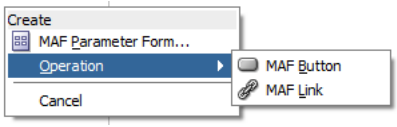
11.3.2.3.2 Dragging and Dropping Operations In addition to attributes, you can drag and drop operations and custom methods. Depending on the type of operation or method, different data binding menu options are provided, as follows:

Operation

This category is for data control operations. It provides the following options (see Figure 11–30):

- MAF Button
- MAF Link
- MAF Parameter Form

Figure 11–30 Context Menu for Operations



Note: If you drop an operation or a method as a child of the List View control, the context menu does not appear and the List Item is created automatically because no other valid control can be dropped as a direct child of the List View control. OEPE creates a binding similar to the following for the generated List Item:

```
<amx:listItem actionListener="#{bindings.getLocation.execute}"/>
```

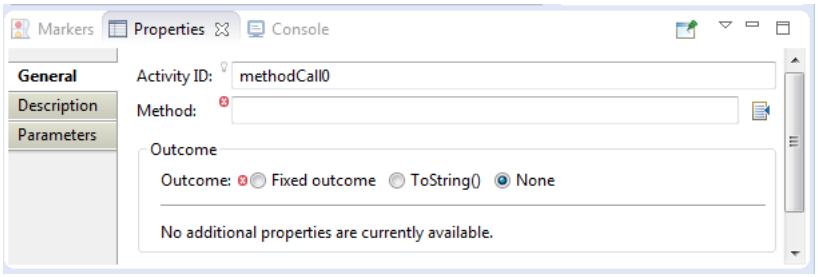
Method

This category is for custom methods. It provides the following options (see Figure 11–31):

- MAF Button

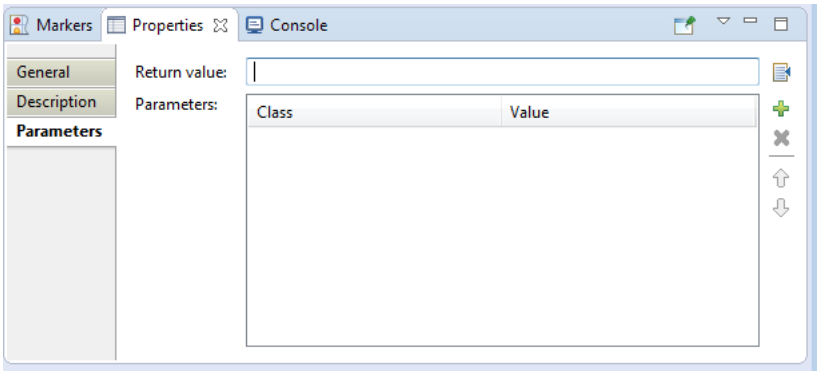
- MAF Link
- MAF Parameter Form

Figure 11–31 Editing a Method call with the General Tab of the Properties pane



The MAF Parameter Form option allows you to choose the method or operation arguments to be inserted in the form, as well as the respective controls for each of the arguments (see Figure 11–32).

Figure 11–32 Edit Form Fields Dialog



The following data bindings are generated after you select the appropriate options in the Edit Form Fields dialog:

```
<amx:panelFormLayout id="pfl1">
   <amx:inputText id="it1"
                  value="#{bindings.empId.inputValue}"
                  label="#{bindings.empId.hints.label}" />
</amx:panelFormLayout>
<amx:commandButton id="cb1"
                   actionListener="#{bindings.ExecuteWithParams1.execute}"
                   text="ExecuteWithParams1"
                   disabled="#{!bindings.ExecuteWithParams1.enabled}"/>
```

For more information about generated bindings, see Section 11.3.2.3.4, "What You May Need to Know About Generated Bindings."

The following are supported control types for the MAF Parameter Form:

- MAF Input Date
- MAF Input Date with Label
- MAF Input Text
- MAF Input Text with Label

- MAF Output Text with Label

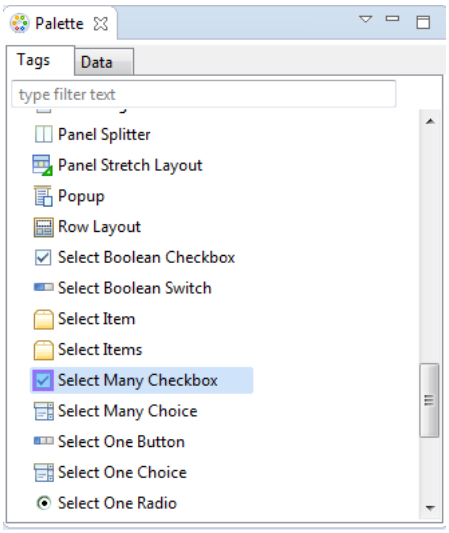
11.3.2.3.3 Dragging and Dropping Collections You can drag and drop collections. Depending on the type of collection, different data binding menu options are provided, as follows:

Multiple Selection

This category provides options for creating multiple selection controls. The following controls can be created under this category (see Figure 11–33):

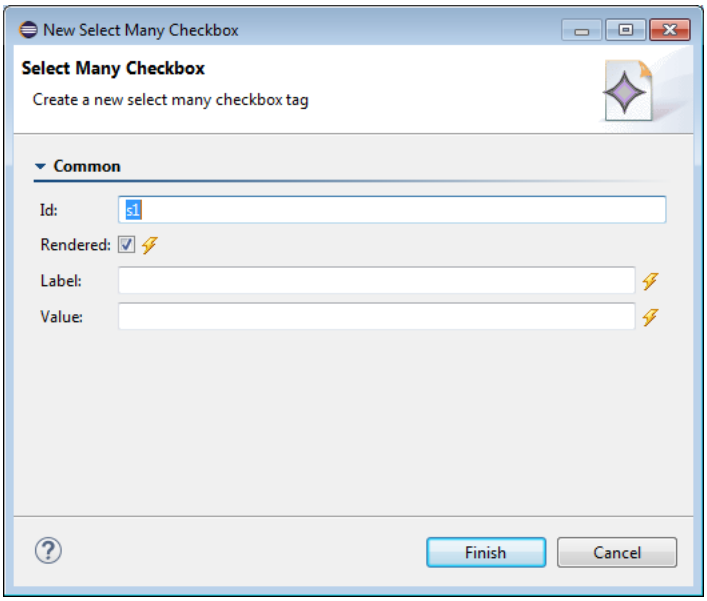
- MAF Select Many Checkbox
- MAF Select Many Choice

Figure 11–33 Context Menu for Multiple Selection Controls



If you select either **Select Many Choice** or **Select Many Checkbox** as the type of the control you want to create, the Edit List Binding dialog is displayed (see Figure 11–34).

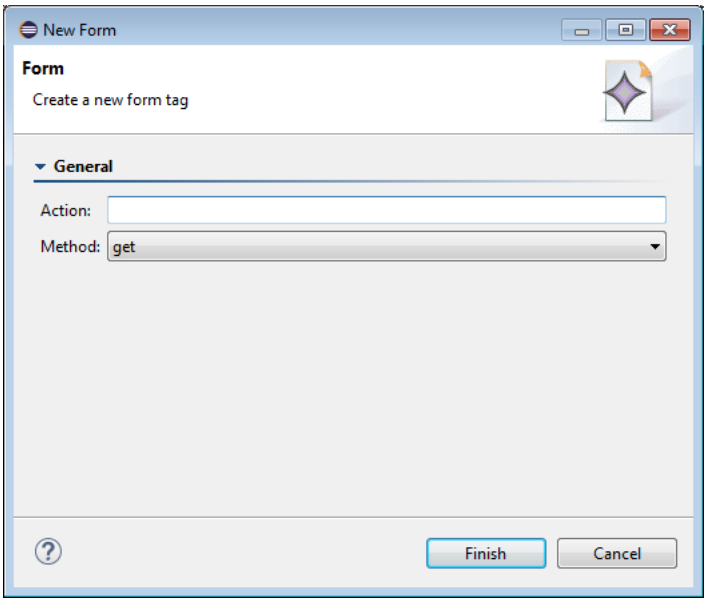
Figure 11–34 Edit List Binding Dialog for Multiple Selection Controls



Form

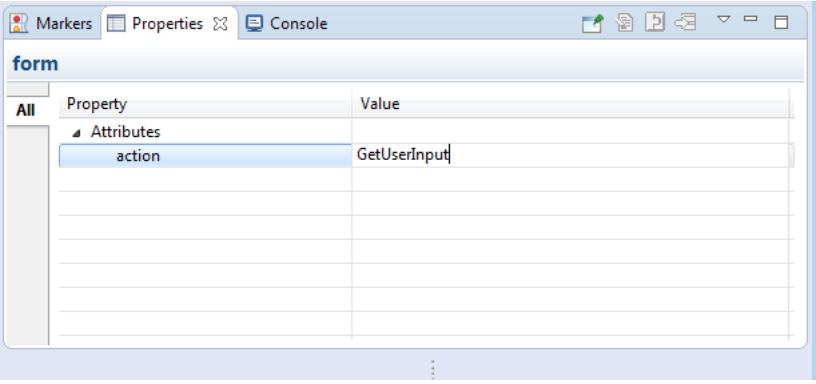
This category is used to create the MAF AMX Panel Form controls. When you select **Form** from the HTML pane of the Palette, OEPE displays the **New Form** dialog (see Figure 11–35):

Figure 11–35 New Form Dialog



Select the method (`get` or `post`) and enter the desired action. OEPE adds a basic HTML form tag to the source; you can edit the Value fields from the Properties pane (see Figure 11–36).

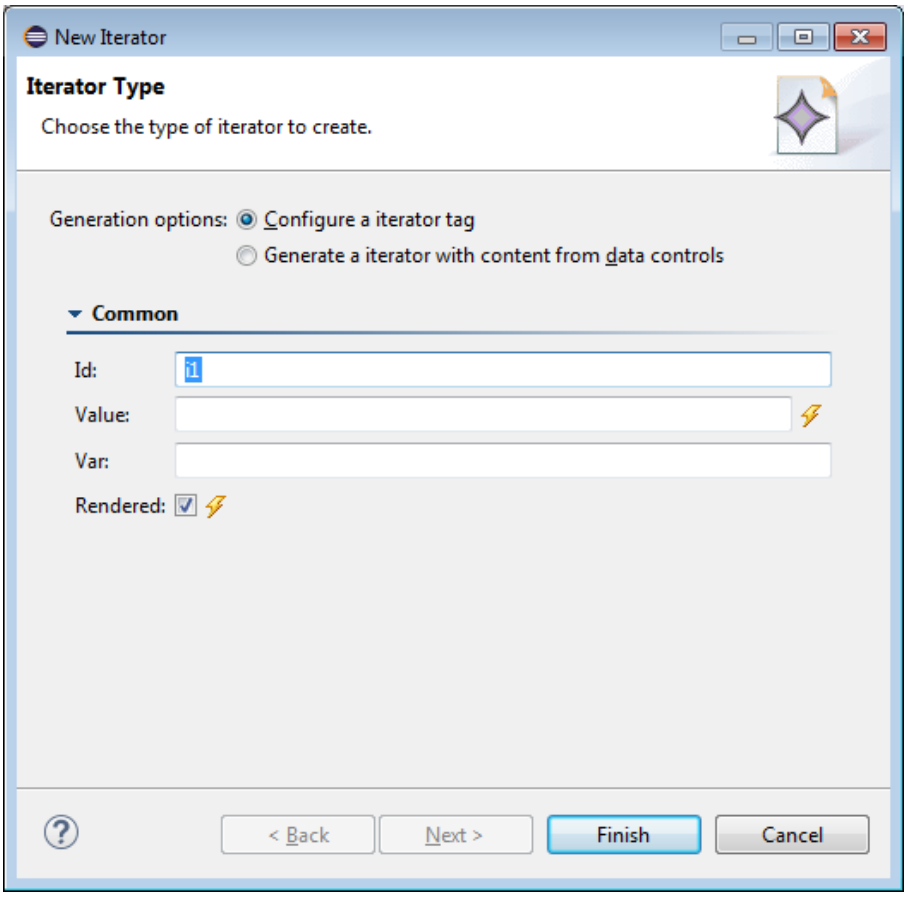
Figure 11–36 Edit Form Fields Dialog for MAF Form



Iterator

This provides an option for creating the MAF AMX Iterator with child components (see Figure 11–37).

Figure 11–37 Context Menu for Iterator Control



If you select **Generate an Iterator with content from data controls**, OEPE displays a Browse link from which you can select the file that contains the content.

The following data bindings are generated after you select the appropriate options in the Edit Iterator dialog:

```
<amx:iterator id="i1"
              var="row"
              value="#{bindings.jobs.collectionModel}">
   <amx:panelLabelAndMessage id="plam6"
                             label="#{bindings.jobs.hints.jobId.label}">
      <amx:outputText id="ot6" value="#{row.jobId}" />
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plam5"
                             label="#{bindings.jobs.hints.title.label}">
      <amx:outputText id="ot5" value="#{row.title}" />
   </amx:panelLabelAndMessage>
   <amx:inputText id="it1"
                  value="#{row.bindings.minSalary.inputValue}"
                  label="#{bindings.jobs.hints.minSalary.label}" />
   <amx:inputText id="it2"
                  value="#{row.bindings.maxSalary.inputValue}"
                  label="#{bindings.jobs.hints.maxSalary.label}" />
</amx:iterator>
```

For more information about generated bindings, see Section 11.3.2.3.4, "What You May Need to Know About Generated Bindings."

The following are supported controls for MAF Iterator:

- MAF Input Text
- MAF Input Text with Label
- MAF Output Text
- MAF Output Text with Label

List View

This provides an option for creating the MAF AMX List View with child components (see Figure 11–38).

Figure 11–38 Context Menu for List View Control

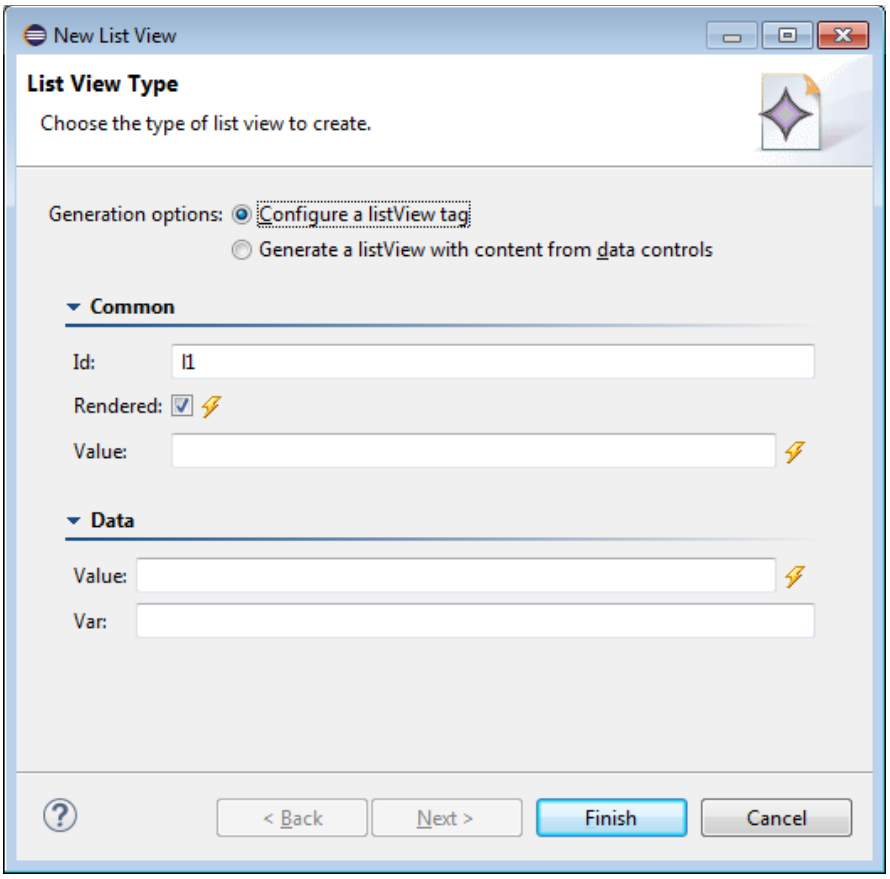


Table 11–4 lists the supported **List Formats** that are displayed in the ListView Gallery.

Table 11–4 List Formats

Format	Element Row Values
Simple	▪ Text
Main-Sub Text	▪ Main Text ▪ Subordinate Text
Start-End	▪ Start Text ▪ End Text
Quadrant	▪ Upper Start Text ▪ Upper End Text ▪ Lower Start Text ▪ Lower End Text

11.3.2.3.4 What You May Need to Know About Generated Bindings Table 11–5 shows sample bindings that are added to an MAF AMX page when components are dropped.

Table 11–5 Sample Data Bindings

Component	Data Bindings
Button	`<amx:commandButton id="commandButton1" actionListener="#{bindings.FindContacts.execute}" text="FindContacts" disabled="#{!bindings.FindContacts.enabled}"> </amx:commandButton>`
Link	`<amx:commandLink id="commandLink1" actionListener="#{bindings.FindContacts.execute}" text="FindContacts" disabled="#{!bindings.FindContacts.enabled}"> </amx:commandLink>`
Input Date with Label	`<amx:inputDate id="inputDate1" value="#{bindings.timeStamp.inputValue}" label="#{bindings.timeStamp.hints.label}"> </amx:inputDate>`
Input Date	`<amx:inputDate id="inputDate1" value="#{bindings.timeStamp.inputValue}"> </amx:inputDate>`
Input Text with Label	`<amx:inputText id="inputText1" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.hints.label}"> </amx:inputText>`
Input Text	`<amx:inputText id="inputText1" value="#{bindings.contactData.inputValue}" simple="true"> </amx:inputText>`
Output Text	`<amx:outputText id="outputText1" value="#{bindings.contactData.inputValue}"> </amx:outputText>`
Output Text with Label	`<amx:panelLabelAndMessage id="panelLabelAndMessage1" label="#{bindings.contactData.hints.label}"> <amx:outputText id="outputText1" value="#{bindings.contactData.inputValue}"/> </amx:panelLabelAndMessage>`
Select Boolean Checkbox	`<amx:selectBooleanCheckbox id="selectBooleanCheckbox1" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.label}"> </amx:selectBooleanCheckbox>`
Select Boolean Switch	`<amx:selectBooleanSwitch id="selectBooleanSwitch" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.label}"> </amx:selectBooleanSwitch>`
Select One Button	`<amx:selectOneButton id="selectOneButton1" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.label}" required="#{bindings.contactData.hints.mandatory}"> <amx:selectItems value="#{bindings.contactData.items}"/> </amx:selectOneButton>`

Table 11–5 (Cont.) Sample Data Bindings

Component	Data Bindings
Select One Choice	`<amx:selectOneChoice id="selectOneChoice1" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.label}"> <amx:selectItems id="selectItems1" value="#{bindings.contactData.items}"/> </amx:selectOneChoice>`
Select Many Checkbox	`<amx:selectManyCheckbox id="selectManyCheckbox1" value="#{bindings.AssetView.inputValue}" label="#{bindings.AssetView.label}"> <amx:selectItems id="selectItems1" value="#{bindings.AssetView.items}"/> </amx:selectManyCheckbox>`
Select One Radio	`<amx:selectOneRadio id="selectOneRadio1" value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.label}" <amx:selectItems id="selectItems1" value="#{bindings.contactData.items}"/> </amx:selectOneRadio>`
Select Many Choice	`<amx:selectManyChoice id="selectManyChoice1" value="#{bindings.AssetView.inputValue}" label="#{bindings.AssetView.label}"> <amx:selectItems id="selectItems1" value="#{bindings.AssetView.items}"/> </amx:selectManyChoice>`

11.3.2.3.5 What You May Need to Know About Generated Drag and Drop Artifacts The first drag and drop event generates the following directories and files:

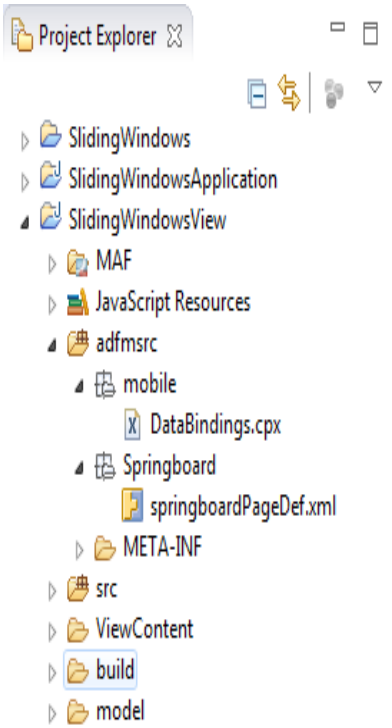


Figure 11–39 shows a sample `DataBindings.cpx` file generated upon drag and drop.

Figure 11–39 DataBindings.cpx File in Source View

The `DataBindings.cpx` files define the binding context for the entire MAF AMX application feature and provide the metadata from which the binding objects are created at runtime. An MAF AMX application feature may have more than one `DataBindings.cpx` file if a component was created outside of the project and then imported. These files map individual MAF AMX pages to page definition files and declare which data controls are being used by the MAF AMX application feature. At runtime, only the data controls listed in the `DataBindings.cpx` files are available to the current MAF AMX application feature.

OEPE automatically creates a `DataBindings.cpx` file in the default package of the ViewController project when you for the first time use the Data Controls window to add a component to a page or an operation to an activity. Once the `DataBindings.cpx` file is created, OEPE adds an entry for the first page or task flow activity. Each subsequent time you use the Data Controls window, OEPE adds an entry to the `DataBindings.cpx` for that page or activity, if one does not already exist.

Once OEPE creates a `DataBindings.cpx` file, you can open it in the Source view (see Figure 11–39) or the editor.

The Page Mappings (`pageMap`) section of the file maps each MAF AMX page or task flow activity to its corresponding page definition file using an ID. The Page Definition Usages (`pageDefinitionUsages`) section maps the page definition ID to the absolute path for page definition file in the MAF AMX application feature. The Data Control Usages (`dataControlUsages`) section identifies the data controls being used by the binding objects defined in the page definition files. These mappings allow the binding container to be initialized when the page is invoked.

You can use the editor to change the ID name for page definition files or data controls by double-clicking the current ID name and editing inline. Doing so will update all references in the MAF AMX application feature. Note, however, that OEPE updates only the ID name and not the file name. Ensure that you do not change a data control name to a reserved word.

You can also click the `DataBindings.cpx` file's element in the Structure window and then use the Properties window to change property values.

Figure 11–40 shows a sample `PageDef` file generated upon drag and drop.

Figure 11–40 PageDef File

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" version="12.1.1.60.73" id="view1PageDef"
                Package="mobile.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <methodIterator Binds="GetContacts.result" DataControl="DeviceDataControl" RangeSize="25"
                    BeanClass="oracle.adfmf.model.datacontrols.device.Contact" id="GetContactsIterator"/>
  </executables>
  <bindings>
    <methodAction id="GetContacts" RequiresUpdateModel="true" Action="invokeMethod" MethodName="GetContacts"
                  IsViewObjectMethod="false" DataControl="DeviceDataControl"
                  InstanceName="data.DeviceDataControl.dataProvider"
                  ReturnName="data.DeviceDataControl.methodResults.
                              GetContacts_DeviceDataControl_dataProvider_GetContacts_result"/>
    <attributeValues IterBinding="GetContactsIterator" id="contactData">
      <AttrNames>
        <Item Value="contactData"/>
      </AttrNames>
    </attributeValues>
  </bindings>
</pageDefinition>
```

Page definition files define the binding objects that populate the data in MAF AMX UI components at runtime. For every MAF AMX page that has bindings, there must be a corresponding page definition file that defines the binding objects used by that page. Page definition files provide design-time access to all the bindings. At runtime, the binding objects defined by a page definition file are instantiated in a binding container, which is the runtime instance of the page definition file.

The first time you use the Data Controls window to add a component to a page, OEPE automatically creates a page definition file for that page and adds definitions for each binding object referenced by the component. For each subsequent databound component you add to the page, OEPE automatically adds the necessary binding object definitions to the page definition file.

By default, the page definition files are located in the `mobile.PageDefs` package in the Application Sources node of the ViewController project. If the corresponding MAF AMX page is saved to a directory other than the default, or to a subdirectory of the default, then the page definition is also be saved to a package of the same name.

For information on how to open a page definition file, see Section 11.3.1.5, "Accessing the Page Definition File." When you open a page definition file in the editor, you can view and configure bindings, contextual events, and parameters for an MAF AMX page using the following tabs:

- Bindings and Executables: this tab shows three different types of objects: bindings, executables, and the associated data controls. Note that data controls do not display unless you select a binding or executable.

  By default, the model binding objects are named after the data control object that was used to create them. If a data control object is used more than once on a page, OEPE adds a number to the default binding object names to keep them unique.

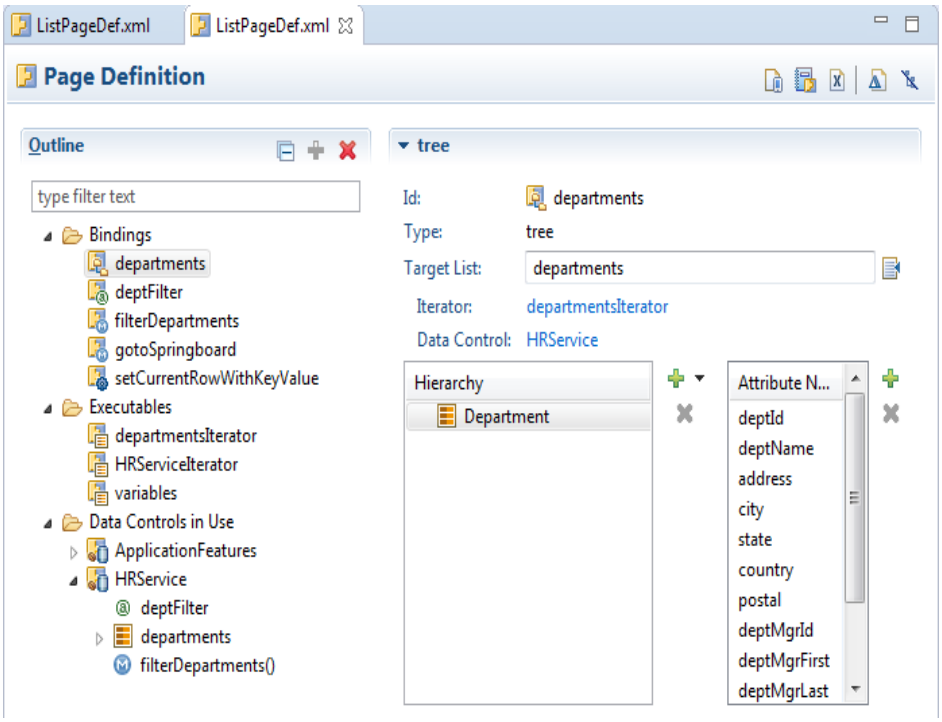
- Contextual Events: you can create contextual events to which artifacts in an MAF AMX application feature can subscribe.

- Parameters: parameter binding objects declare the parameters that the page evaluates at the beginning of a request. You can define the value of a parameter in the page definition file using static values or EL expressions that assign a static value.

When you click an item in the editor (or the associated node in the Structure window), you can use the Properties window to view and edit the attribute values for the item, or you can edit the XML source directly by clicking the Source tab.

11.3.2.3.6 Using the MAF AMX Editor Bindings Tab OEPE's Bindings tab (see Figure 11–41) is available in the MAF AMX Editor. It displays the data bindings defined for a specific MAF AMX page. If you select a binding, its relationship to the underlying Data Control are shown and the link to the `PageDef` file is provided.

Figure 11–41 Bindings Tab



11.3.2.3.7 What You May Need to Know About Removal of Unused Bindings When you delete or cut an MAF AMX component from the Structure window, unused bindings are automatically removed from your page.

> **Note:** Deleting a component from the Source editor does not trigger the removal of bindings.

Figure 11–42 demonstrates the deletion of a List View component that references bindings. Upon deletion, the related binding entry is automatically removed from the corresponding `PageDef.xml` file.

Figure 11–42 Deleting Bound Components from Page

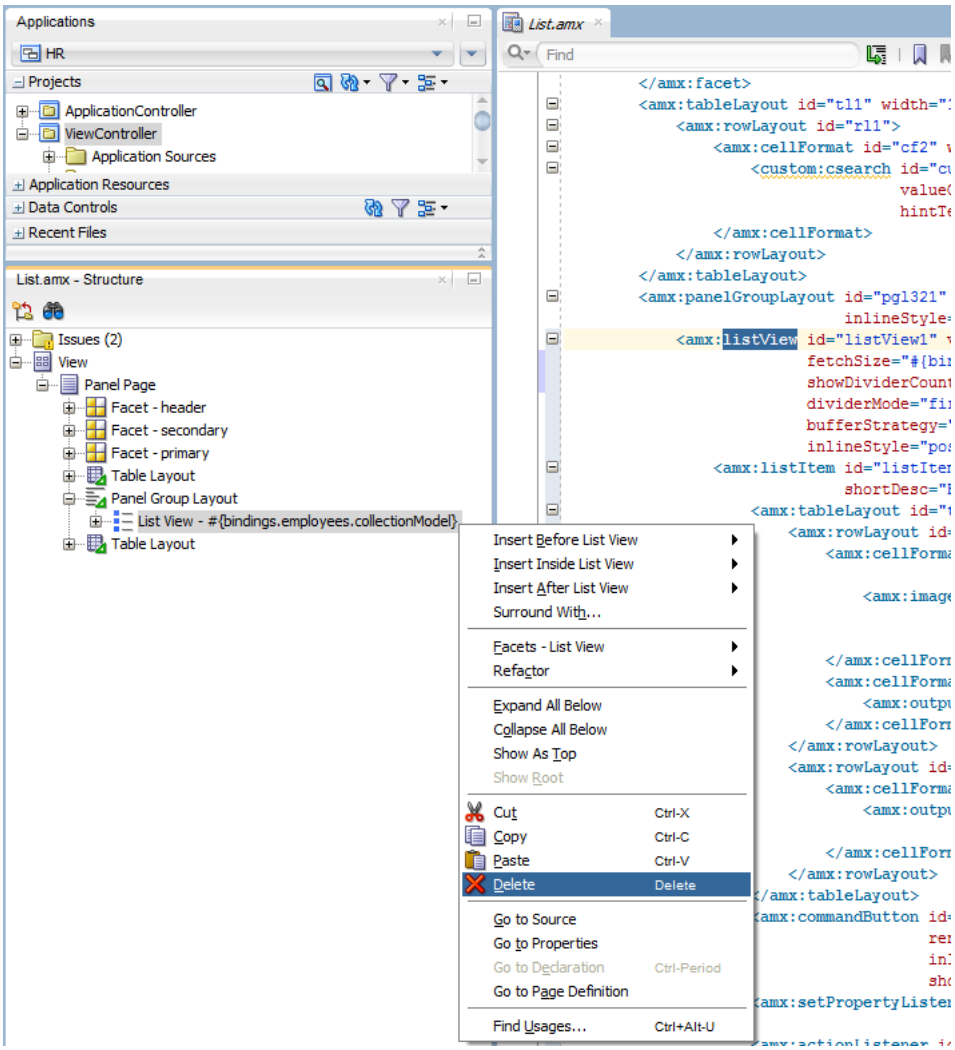
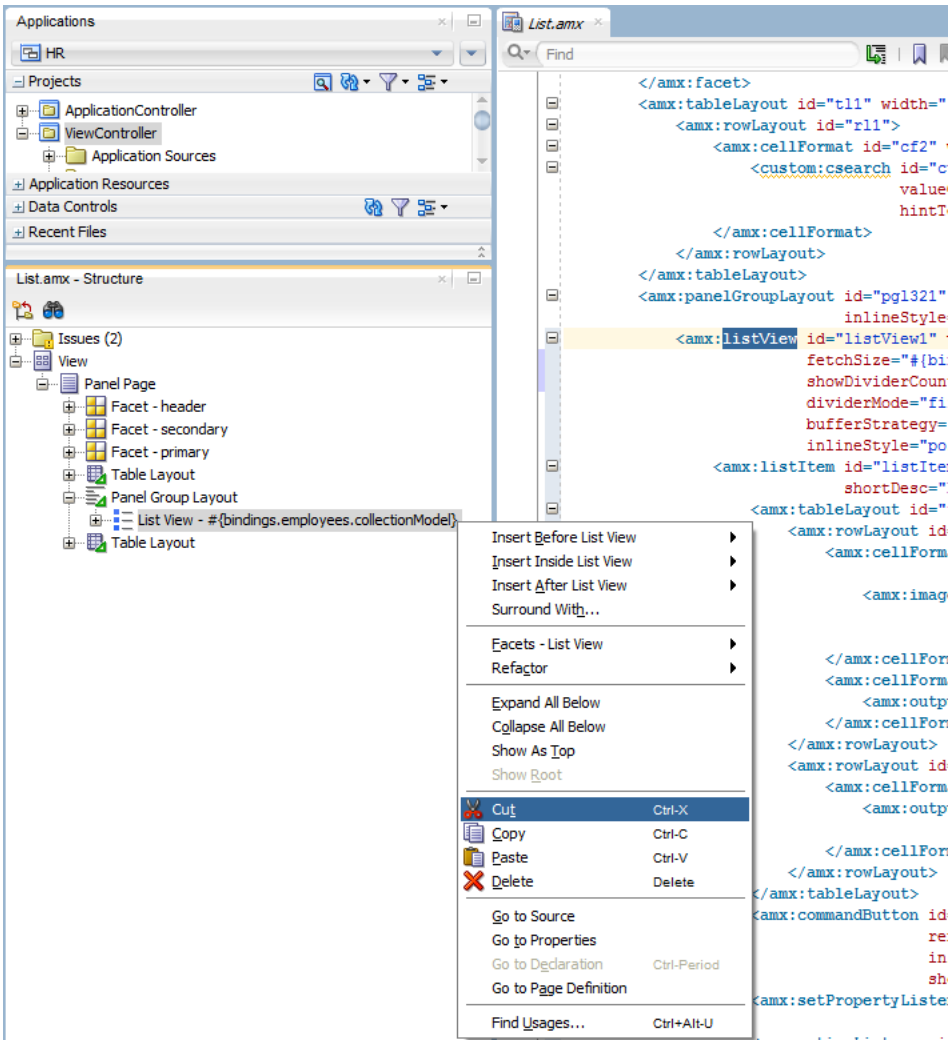


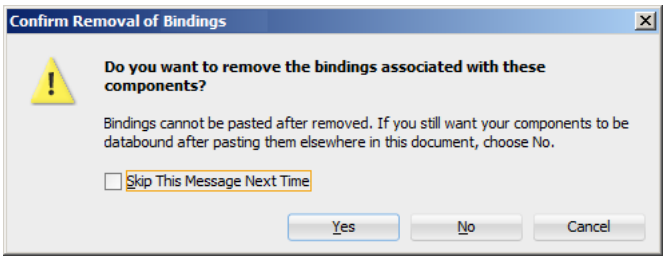
Figure 11–43 demonstrates the removal of the List View component by cutting it from the page.

Figure 11–43 Cutting Bound Components from Page



After clicking **Cut**, you are presented with the Confirm Removal of Bindings dialog that prompts you to choose whether or not to delete the corresponding bindings, as shown in Figure 11–44.

Figure 11–44 Confirm Removal of Bindings Dialog



11.3.2.4 What You May Need to Know About Element Identifiers and Their Audit

MAF generates a unique element identifier (`id`) and automatically inserts it into the MAF AMX page when an element is added by dropping a component from the Palette, or by dragging and dropping a data control. This results in a valid identifier in

the MAF AMX page that differentiates each component from others, possibly similar components within the same page.

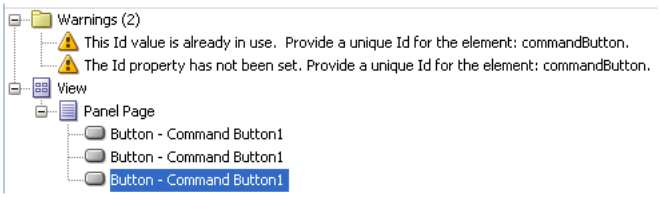
MAF provides an identifier audit utility that does the following:

- Checks the presence and uniqueness of identifiers in an MAF AMX page.
- If the identifier is not present or not unique, an error is reported for each required `id` attribute of an element.
- Provides an automatic fix to generate a unique `id` for the element when a problem with the identifier is reported.

Figure 11–45 and Figure 11–46 show the identifier error reporting in the Source editor and Structure pane respectively.

Figure 11–45 Element Identifier Audit in Source Editor

Figure 11–46 Element Identifier Audit in Structure Pane



In addition to the `id`, the audit utility checks the `popupId` and `alignId` attributes of the Show Popup Behavior operation (see Section 12.2.8, "How to Use a Popup Component").

Figure 11–47 and Figure 11–48 show the Show Popup Behavior's Popup Id and Align Id attributes error reporting in the Source Editor respectively.

Figure 11–47 Popup Id Attribute Audit in Source Editor

Figure 11–48 Align Id Attribute Audit in Source Editor

11.3.3 What You May Need to Know About the Server Communication

The security architecture used by MAF guarantees that the browser hosting an MAF AMX page does not have access to the security information needed to make connections to a secure server to obtain its resources. This has a direct impact on AJAX calls made from MAF AMX pages: these calls are not supported, which poses limitations on the use of Java Script from within MAF AMX UI components. Communication with the server must occur from the embedded Java code layer.

12

Creating the MAF AMX User Interface

This chapter describes how to create the user interface for MAF AMX pages.

This chapter includes the following sections:

- Section 12.1, "Introduction to Creating the User Interface for MAF AMX Pages"
- Section 12.2, "Designing the Page Layout"
- Section 12.3, "Creating and Using UI Components"
- Section 12.4, "Enabling Gestures"
- Section 12.5, "Providing Data Visualization"
- Section 12.6, "Styling UI Components"
- Section 12.7, "Localizing UI Components"
- Section 12.8, "Understanding MAF Support for Accessibility"
- Section 12.9, "Validating Input"
- Section 12.10, "Using Event Listeners"

12.1 Introduction to Creating the User Interface for MAF AMX Pages

MAF provides a set of UI components and operations that enable you to create MAF AMX pages which behave appropriately for both the iOS and Android user experience.

MAF AMX adheres to the typical OEPE development experience by allowing you to add UI components and operations in an editor window. In essence, MAF AMX UI components render HTML equivalents of the native components on the iOS and Android platforms, with their design-time behavior in OEPE being similar to components used by other technologies. In addition, the UI components are integrated with MAF's controller and model for declarative navigation and data binding.

> **Note:** When developing interfaces for mobile devices, always be aware of the fact that screen space is very limited. In addition, touchscreen support is not available on some mobile devices.

For more information, see the following:

- Chapter 11, "Creating MAF AMX Pages"
- Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX"
- Chapter 18, "Creating Custom MAF AMX UI Components"

12.2 Designing the Page Layout

MAF AMX provides layout components (listed in Table 12–1) that let you arrange UI components in a page. Usually, you begin building pages with these components, and then add other components that provide other functionality either inside these containers, or as child components to the layout components. Some of these components provide geometry management functionality, such as the capability to stretch when placed inside a component that stretches.

Table 12–1 MAF AMX Page Management, Layout, and Spacing Components

Component	Type	Description
View	Core Page Structure Component	Creates a `view` element in a MAF AMX file. Automatically inserted into the file when the file is created. For more information, see Section 12.2.1, "How to Use a View Component."
Panel Page	Core Page Structure Component	Creates a `panelPage` element in a MAF AMX file. Defines the central area in a page that scrolls vertically between the header and footer areas. For more information, see Section 12.2.2, "How to Use a Panel Page Component." For more information about MAF AMX files, see Section 11.3.1.2, "Creating MAF AMX Pages."
Facet	Core Page Structure Component	Creates a `facet` element in a MAF AMX file. Defines an arbitrarily named facet on the parent component. For more information, see Section 12.2.7, "How to Use a Facet Component."
Fragment	Core Page Structure Component	Creates a `fragment` element in a MAF AMX file. Enables sharing of the page contents. For more information, see Section 12.2.16, "How to Use the Fragment Component."
Facet Definition	Core Page Structure Component	Creates a `facetRef` element in a MAF AMX Fragment file. Used inside a page fragment definition (`fragmentDef`) to reference a facet defined in the page fragment usage. For more information, see Section 12.2.7, "How to Use a Facet Component."
Panel Group Layout	Page Layout Component	Creates a `panelGroupLayout` element in a MAF AMX file. Groups child components either vertically or horizontally. For more information, see Section 12.2.3, "How to Use a Panel Group Layout Component."
Panel Form Layout	Page Layout Component	Creates a `panelFormLayout` element in a MAF AMX file. Positions components, such as Input Text, so that their labels and fields line up horizontally or above each component. For more information, see Section 12.2.4, "How to Use a Panel Form Layout Component."
Panel Label And Message	Page Layout Component	Creates a `panelLabelAndMessage` element in a MAF AMX file. Lays out a label and its children. For more information, see Section 12.2.6, "How to Use a Panel Label And Message Component."
Panel Stretch Layout	Page Layout Component	Creates a `panelStretchLayout` element in a MAF AMX file. Allows placement of a panel on each side of another panel. For more information, see Section 12.2.5, "How to Use a Panel Stretch Layout Component."

Table 12–1 (Cont.) MAF AMX Page Management, Layout, and Spacing Components

Component	Type	Description
Popup	Secondary Window	Creates a `popup` element in a MAF AMX file. For more information, see Section 12.2.8, "How to Use a Popup Component."
Panel Splitter	Interactive Page Layout Container	Creates a `panelSplitter` element in a MAF AMX file. For more information, see Section 12.2.9, "How to Use a Panel Splitter Component."
Panel Item	Interactive Page Layout Component	Creates a `panelItem` element in a MAF AMX file. For more information, see Section 12.2.9, "How to Use a Panel Splitter Component."
Deck	Page Layout Component	Creates a `deck` element in a MAF AMX file. For more information, see Section 12.2.14, "How to Use a Deck Component."
Flex Layout	Page Layout Component	Creates a `flexLayout` element in a MAF AMX file. Provides flexible flow for components depending on the available space. For more information, see Section 12.2.15, "How to Use a Flex Layout Component."
Spacer	Page Layout Component	Creates an area of blank space represented by a `spacer` element in a MAF AMX file. For more information, see Section 12.2.10, "How to Use a Spacer Component."
Table Layout	Page Layout Component	Creates a `tableLayout` element in a MAF AMX file. Represents a table consisting of rows. For more information, see Section 12.2.11, "How to Use a Table Layout Component."
Row Layout	Page Layout Component	Creates a `rowLayout` element in a MAF AMX file. Represents a row consisting of cells in a Table Layout component. For more information, see Section 12.2.11, "How to Use a Table Layout Component."
Cell Format	Page Layout Component	Creates a `cellFormat` element in a MAF AMX file. Represents a cell in a Row Layout component. For more information, see Section 12.2.11, "How to Use a Table Layout Component."
Masonry Layout	Page Layout Container	Creates a `masonryLayout` element in a MAF AMX file. Presents its child components as tiles arranged in columns and rows. For more information, see Section 12.2.12, "How to Use a Masonry Layout Component."
Accessory Layout	Page Layout Component	Creates an `accessoryLayout` element in a MAF AMX file. Used within List View component to enable dragging of the content left or right to reveal optional content areas. For more information, see Section 12.2.13, "How to Use an Accessory Layout Component."

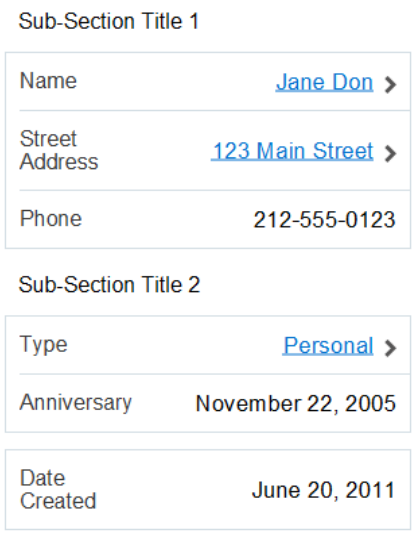
You add a layout component by dragging and dropping it onto a MAF AMX page from the Palette (see Section 11.3.2.1, "Adding UI Components"). Then you use the Properties window to set the component's attributes. For information on attributes of each particular component, see *Tag Reference for Oracle Mobile Application Framework*.

The following example demonstrates several page layout elements defined in a MAF AMX file.

Note: You declare the page layout elements under the <amx> namespace.

```
<amx:panelPage id="pp1">
   <amx:outputText id="outputText1"
                   value="Sub-Section Title 1"
                   styleClass="adfmf-text-sectiontitle"/>
   <amx:panelFormLayout id="panelFormLayout1" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage1" label="Name">
         <amx:commandLink id="commandLink1" text="Jane Don" action="editname" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage id="panelLabelAndMessage2" label="Street Address">
         <amx:commandLink id="commandLink2"
                          text="123 Main Street"
                          action="editaddr" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage id="panelLabelAndMessage3" label="Phone">
         <amx:outputText id="outputText2" value="212-555-0123" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
   <amx:outputText id="outputText3"
                   value="Sub-Section Title 2"
                   styleClass="adfmf-text-sectiontitle" />
   <amx:panelFormLayout id="panelFormLayout2" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage4" label="Type">
         <amx:commandLink id="commandLink3" text="Personal" action="edittype" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage label="Anniversary">
         <amx:outputText id="outputText4" value="November 22, 2005" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
   <amx:panelFormLayout id="panelFormLayout3" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage5" label="Date Created">
         <amx:outputText id="outputText5" value="June 20, 2011" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
</amx:panelPage>
```

Figure 12–1 Page Layout Components at Design Time



You use the standard Cascading Style Sheets (CSS) to manage visual presentation of your layout components. CSS are located in the `View Content/css` directory of your Application project, with default CSS provided by MAF. For more information, see Section 12.6.1, "How to Use Component Attributes to Define Style."

The user interface created for iOS platform using MAF AMX displays correctly in both the left-to-right and right-to-left language environments. In the latter case, the components originate on the right-hand side of the screen instead of on the left-hand side. Some of the MAF AMX layout components, such as the Popup (see Section 12.2.8, "How to Use a Popup Component"), Panel Item, and Panel Splitter (see Section 12.2.9, "How to Use a Panel Splitter Component") can be configured to enable specific right-to-left behavior. For more information about right-to-left configuration of MAF AMX pages, see Section 12.4, "Enabling Gestures" and Section 11.2.11, "How to Specify the Page Transition Style."

> **Note:** The right-to-left text direction is not supported on Android prior to version 4.2.

12.2.1 How to Use a View Component

A View (`view` element in a MAF AMX file) is a core page structure component that is automatically inserted into a MAF AMX file when the file is created. This component provides a hierarchical representation of the page and its structure and represents a single MAF AMX page.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.2 How to Use a Panel Page Component

A Panel Page (`panelPage` element in a MAF AMX file) is a component that allows you to define a scrollable area of the screen for laying out other components.

By default, when you create a MAF AMX page, OEPE automatically creates and inserts a Panel Page component into the page. When you add components to the page, they will be inserted inside the Panel Page component.

To prevent scrolling of certain areas (such as a header and footer of the page) and enable stretching when orientation changes, you can specify a Facet component for your Panel Page. The Panel Page's header Facet includes the title placed in the Navigation Bar of each page. For information about other types of Facet components that the Panel Page can contain, see Section 12.2.7, "How to Use a Facet Component."

The following example shows the `panelPage` element defined in a MAF AMX file. This Panel Page contains a header Facet.

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="ot1" value="Welcome"/>
   </amx:facet>
</amx:panelPage>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.3 How to Use a Panel Group Layout Component

The Panel Group Layout component is a basic layout component that lays out its children horizontally or vertically. In addition, there is a wrapping layout option that enables child components to flow across and down the page.

To create the Panel Group Layout component, use the Palette.

To add the Panel Group Layout component:

1. In the Palette, open the **MAF AMX** pane and drag and drop a Panel Group Layout to the MAF AMX page.
2. Insert the desired child components into the Panel Group Layout component.
3. To add spacing between adjacent child components, insert the Spacer (`spacer`) component, also available from the MAF AMX pane of the Palette.
4. Use the Properties window to set the component attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the `panelGroupLayout` element defined in a MAF AMX file.

```
<amx:panelGroupLayout styleClass="prod" id="pgl1">
   <amx:outputText styleClass="prod-label" value="Screen Size:" id="ot1"/>
</amx:panelGroupLayout>
```

12.2.3.1 Customizing the Scrolling Behavior

Scrolling behavior of the Panel Group Layout component is defined by its `scrollPolicy` attribute which can be set to `auto` (default), `none`, or `scroll`. By default, this behavior matches the one defined in the active skin.

To disable scrolling regardless of the behavior defined in the active skin, you set the `scrollPolicy` attribute to `none`. When the Panel Group Layout component is not scrollable, its content is not constrained.

To enable scrolling regardless of the behavior defined in the active skin, you set the `scrollPolicy` attribute to `scroll`. If the Panel Group Layout component is scrollable, the scrolling is provided when the component's dimensions are constrained.

Since scrolling consumes a lot of memory and may lead to the application crashing, you should minimize its use. In the `mobileAlta` skin (see Section 12.6.2, "What You May Need to Know About Skinning"), scrolling of the Panel Group Layout, Panel Form Layout (see Section 12.2.4, "How to Use a Panel Form Layout Component"), and Table Layout (see Section 12.2.11, "How to Use a Table Layout Component") is disabled. It is recommended that you use the `mobileAlta` skin for your application and limit instances of setting the `scrollPolicy` to `scroll` to when it is necessary. To simulate the scrolling behavior for the Panel Form Layout and Table Layout, you can enclose them within a scrollable Panel Group Layout component when scrolling is required.

12.2.4 How to Use a Panel Form Layout Component

The Panel Form Layout (`panelFormLayout`) component positions components so that their labels and fields align horizontally. In general, the main content of the Panel Form Layout component is comprised of input components (such as Input Text) and selection components (such as Choice). If a child component with a `label` attribute defined is placed inside the Panel Form Layout component, the child component's label and field are aligned and sized based on the Panel Form Layout definitions. Within the Panel Form Layout, the label area can either be displayed on the start side of the field area or on a separate line above the field area. Separate lines are used if the `labelPosition` attribute of the Panel Form Layout is set to `topStart`, `topCenter`, or `topEnd`. Otherwise the label area appears on the start side of the field area. Within the label area, the `labelPosition` attribute controls where the label text can be aligned:

- to the start side (`labelPosition="start"` or `labelPosition="topStart"`)
- to the center (`labelPosition="center"` or `labelPosition="topCenter"`)
- to the end side (`labelPosition="end"` or `labelPosition="topEnd"`)

Within the field area, the `fieldHalign` attribute controls where the field content can be aligned:

- to the start side (`fieldHalign="start"`)
- to the center (`fieldHalign="center"`)
- to the end side (`fieldHalign="end"`)

Within the Panel Form Layout, the child components can be placed in one or more columns using `maxColumns` and `rows` attributes. These attributes should be used in conjunction with `labelWidth`, `fieldWidth`, `labelPosition`, and `showHorizontalDividers` attributes to obtain the optimal multi-column layout.

> **Note:** To switch from a single-column to multi-column layout, the value of the `rows` attribute must be greater than 1, regardless of the value to which the `maxColumns` attribute is set. When the `rows` attribute is specified, the `maxColumns` attribute restricts the layout to that number of columns as a maximum; however, there are as many rows as are required to lay out the child components.

To add the Panel Form Layout component:

1. In the Palette, drag and drop a Panel Form Layout component to the MAF AMX page.
2. In the New Panel Form Layout dialog, set the component's attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the panelFormLayout element defined in a MAF AMX file.

```
<amx:panelFormLayout styleClass="prod" id="pfl1">
   <amx:panelLabelAndMessage label="Type" id="plm1">
      <amx:commandLink text="Personal" action="edittype" id="cl1"/>
   </amx:panelLabelAndMessage>
</amx:panelFormLayout>
```

12.2.5 How to Use a Panel Stretch Layout Component

The Panel Stretch Layout (panelStretchLayout) component manages three child Facet components: top, bottom, and center (see the next example). You can use any number and combination of these facets.

```
<amx:panelStretchLayout id="psl1">
   <amx:facet name="top">
   </amx:facet>
   <amx:facet name="center">
   </amx:facet>
   <amx:facet name="bottom">
   </amx:facet>
</amx:panelStretchLayout>
```

If an attempt is made to represent the Panel Stretch Layout component as a set of three rectangles stacked one on top of another, the following would apply:

- The height of the top rectangle is defined by the natural height of the top facet.
- The height of the bottom rectangle is defined by the natural height of the bottom facet.
- The rest of the vertical space is distributed to the rectangle in the middle. If the height of this rectangle is smaller than the value defined for Center.height and the scrollPolicy attribute of the panelStretchLayout is set to either scroll or auto, then scroll bars are added.

To add the Panel Stretch Layout component:

1. In the Palette, drag and drop a Panel Stretch Layout onto the MAF AMX page.
2. Review the created child Facet components and, if necessary, remove some of them.
3. Use the Properties window to set the component attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.6 How to Use a Panel Label And Message Component

Use the Panel Label And Message (panelLabelAndMessage) component to place a component which does not have a label attribute. These components usually include an Output Text, Button, or Link.

To add the Panel Label And Message component:

1. In the MAF AMX pane of the Palette, drag and drop a Panel Label And Message component into a Panel Group Layout component.
2. In the New Panel Label and Message dialog, set the component's attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The example below shows the panelLabelAndMessage element defined in a MAF AMX file. The label attribute is used for the child component.

```
<amx:panelLabelAndMessage label="Phone" id="plm1">
   <amx:outputText value="212-555-0123" id="ot1"/>
</amx:panelLabelAndMessage>
```

12.2.7 How to Use a Facet Component

You use the Facet (`facet`) component to define an arbitrarily named facet, such as a header or footer, on the parent layout component. The position and rendering of the Facet are determined by the parent component.

The MAF AMX page header is typically represented by the Panel Page component (see Section 12.2.2, "How to Use a Panel Page Component") in combination with the Header, Primary, and Secondary facets:

- Header facet: contains the page title.
- Primary Action facet: represents an area that appears in the left corner of the header bar and typically hosts Button or Link components, but can contain any component type.
- Secondary Action facet: represents an area that appears in the right corner of the header bar and typically hosts Button or Link components, but can contain any component type.

The MAF AMX page footer is represented by the Panel Page component (see Section 12.2.2, "How to Use a Panel Page Component") in combination with the footer facet:

- Footer facet: represents an area that appears below the content area and typically hosts Button or Link components, but can contain any component type.

The next example shows the `facet` element declared inside the Panel Page container. The type of the facet is always defined by its `name` attribute (see Table 12–2).

```
<amx:panelPage id="pp1">
   <amx:facet name="footer">
      <amx:commandButton id="cb2" icon="folder.png"
                         text="Move (#{myBean.mailcount})"
                         action"move"/>
   </amx:facet>
</amx:panelPage>
```

Table 12–2 lists predefined Facet types that you can use with specific parent components.

Table 12–2 Facet Types and Parent Components

Parent Component	Facet Type (name)
Panel Page (`panelPage`)	`header`, `footer`, `primary`, `secondary`
List View (`listView`)	`header`, `footer`
Carousel (`carousel`)	`nodeStamp`, `carouselItem`
Panel Splitter (`panelSplitter`)	`navigator`
Panel Stretch Layout (`panelStretchLayout`)	`top`, `center`, `bottom`
Data Visualization Components. For more information, see Section 12.5, "Providing Data Visualization."	`dataStamp`, `seriesStamp`, `overview`, `rows` (applicable to NBox), `columns` (applicable to NBox), `cells` (applicable to NBox), `icon` (applicable to NBoxNode), `indicator` (applicable to NBoxNode)

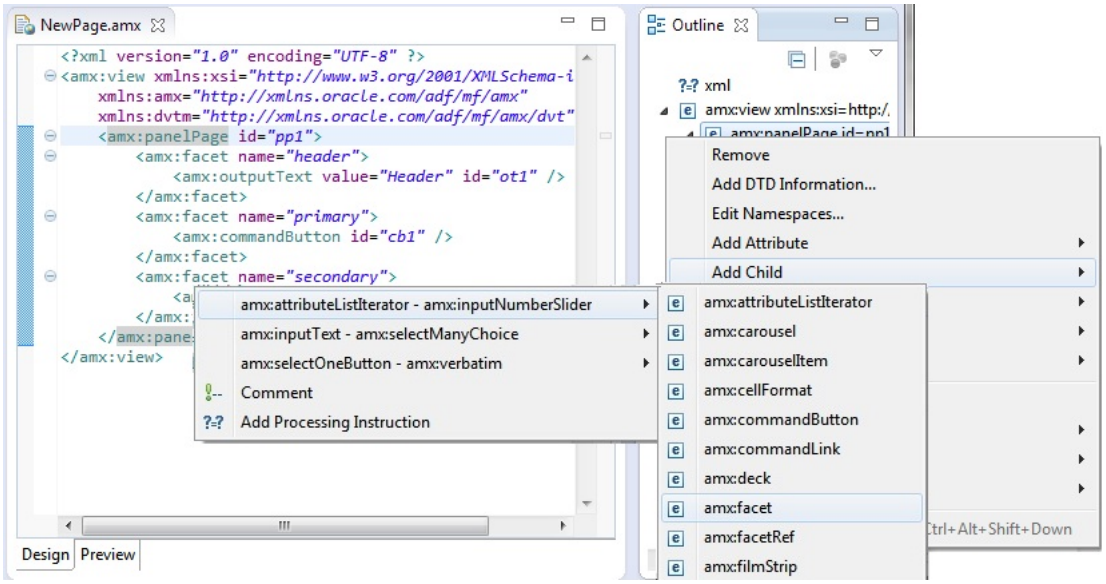
To add the Facet component:

You can use the context menu displayed on the Outline pane to add a Facet component as a child of another component. The context menu displays options that are valid for your selected parent component.

To add a Facet, first select the parent component in the Outline or Source editor, and complete one of the following steps:

- Right-clicking the `amx:PanelPage` entry in the Outline lets you enter a Facet, as Figure 12–2 shows.

Figure 12–2 Using Context Menu to Add Facet to Panel Page



Alternatively:

1. In the Palette, drag and drop a Facet component into another component listed in Table 12–2.
2. In the Properties window, set the component's attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.8 How to Use a Popup Component

Use the Popup (`popup`) component to display a popup window. You can declare this component as a child of the View component.

You can use the following operations in conjunction with the Popup component:

- Close Popup Behavior (`closePopupBehavior`) operation represents a declarative way to close the Popup in response to a client-triggered event.
- Show Popup Behavior (`showPopupBehavior`) operation represents a declarative way to show the Popup in response to a client-triggered event specified using the `type` attribute of the Show Popup Behavior.

  The `popupId` attribute of the Show Popup Behavior specifies the unique identifier of the Popup component relative to its parent component. The `alignId` attribute of the Show Popup Behavior specifies the unique identifier of the UI component relative to which the Popup is to be aligned. Since setting identifiers manually is tedious and can lead to invalid references, you set values for these two attributes

using an editor that is integrated with the standard Properties window. There is an Audit rule that is specifically defined to validate these identifiers (see Section 11.3.2.4, "What You May Need to Know About Element Identifiers and Their Audit").

The `decoration` attribute of the Show Popup Behavior allows you to configure the Popup to have an anchor pointing to the component that matches the specified `alignId`. You do so by setting the `decoration` attribute to `anchor` (the default value is `simple`).

Note: There is no need to define `decoration="anchor"` to use the `alignId` attribute. When using `decoration="anchor"`, if the `alignId` attribute is not specified or a match is not found for the `alignId`, the `decoration` defaults to `simple` resulting in minimal ornamentation of the Popup component.

Values you set for the `align` attribute of the Show Popup Behavior indicate where the alignment of the Popup component is to be positioned if there is enough space to satisfy that positioning. When there is not enough space, alternate positioning is chosen by MAF.

Tip: To center a Popup on the screen, you should set the `alignId` attribute of the Panel Page component, and then use the `align="center"`.

For more information on the Show Popup Behavior component's attributes and their values, see *Tag Reference for Oracle Mobile Application Framework*.

The example below shows `popup` and `showPopupBehavior` elements defined in a MAF AMX file.

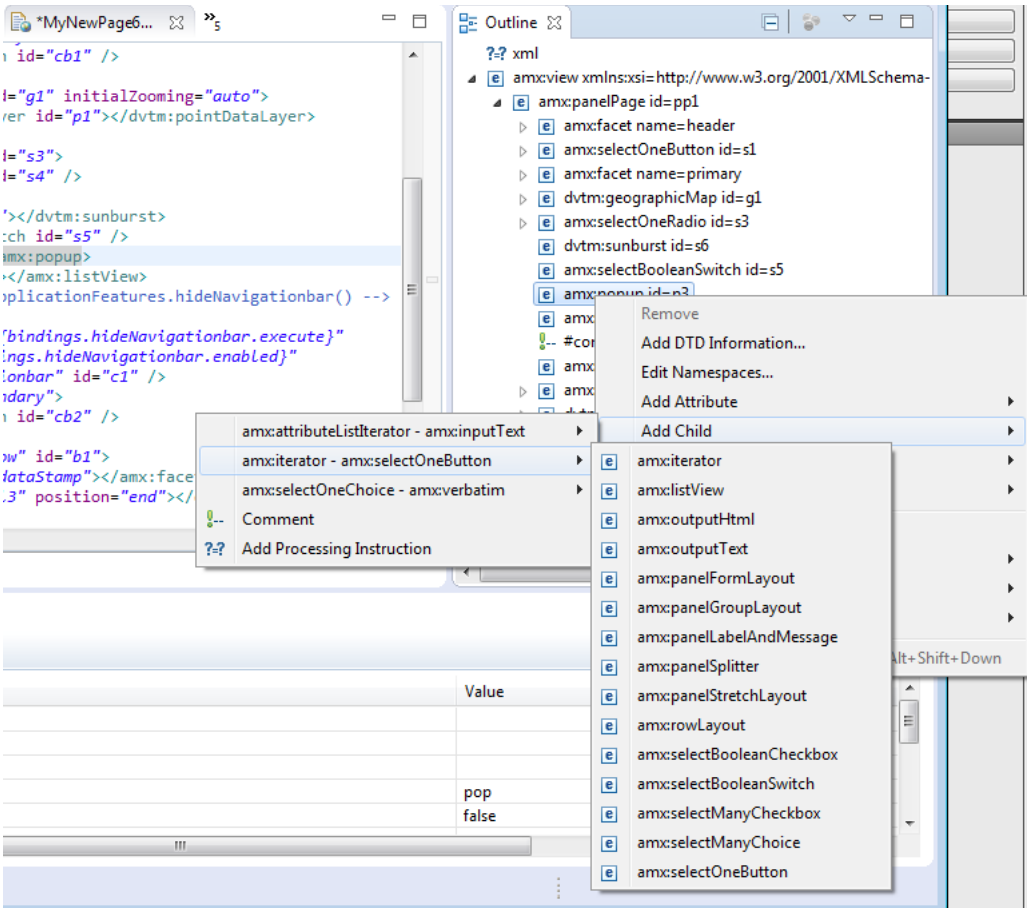
```
<amx:view>
   <amx:panelPage id="panelPage1">
      <amx:commandButton id="commandButton1" text="Show Popup">
         <amx:showPopupBehavior popupId="popup1" type="action"
                                align="topStart" alignId="panelPage1"
                                decoration="anchor"/>
      </amx:commandButton>
   </amx:panelPage>
   <amx:popup id="popup1"
              animation="slideUp"
              autoDismiss="true"
              backgroundDimming="off"/>
</amx:view>
```

Popup components can display validation messages when the user input errors occur. For more information, see Section 12.9, "Validating Input."

To set a Popup child:

1. Select the `popup` element in the Outline pane and click the right mouse button.
2. Select Add Child from a list of Popup components (see Figure 12–3).

Figure 12–3 Selecting Popup Child attribute from List



3. Select the Popup component to be displayed when this Show Popup Behavior is invoked.

A MAF sample application called UIDemo demonstrates how to use the Popup component and how to apply styles to adjust the page layout to a specific pattern. The UIDemo application is available from **File** > **New** > **MAF Examples**.

12.2.9 How to Use a Panel Splitter Component

Use the Panel Splitter (`panelSplitter`) component to display multiple content areas that may be controlled by a left-side navigation pane. Panel Splitter components are commonly used on tablet devices that have larger display size. These components are typically used with a list on the left and the content on the right side of the display area.

A Panel Splitter can contain a navigator Facet (see Section 12.2.7, "How to Use a Facet Component") which is generated automatically when you drag and drop the Panel Splitter onto a MAF AMX page, and a Panel Item component. The Panel Item (`panelItem`) component represents the content area of a Panel Splitter. Since each Panel Splitter component must have a least one Panel Item, the Panel Item is automatically added to the Panel Splitter when the Panel Splitter is created. Each Panel Item component can contain any component that a Panel Group Layout can contain (see Section 12.2.3, "How to Use a Panel Group Layout Component").

The left side of the Panel Splitter is represented by a navigator facet (`navigator`), which is optional in cases where only multiple content with animations is desired (for

example, drawing a multicontent area with a Select Button that requires animation when selecting different buttons to switch content). When in landscape mode, this facet is rendered; in portrait mode, a button is placed above the content area and when clicked, the content of the facet is launched in a popup.

When developing for iOS platform, you can configure the Panel Splitter and Panel Item to accommodate the right-to-left language environment by setting their `animation` attribute to either `slideStart`, `slideEnd`, `flipStart`, or `flipEnd`. The `animation` attribute of the Panel Item components overrides the Panel Splitter's `animation` attribute. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the `panelSplitter` element defined in a MAF AMX file, with the `navigator` facet used as a child component.

```
<amx:panelSplitter id="ps1"
               selectedItem="#{bindings.display.inputValue}"
               animation="flipEnd">
   <amx:facet name="navigator">
      <amx:listView id="lv1"
                    value="#{bindings.data.collectionModel}"
                    var="row"
                    showMoreStrategy="autoScroll"
                    bufferStrategy="viewport>
         ...
      </listView>
   </facet>
   <amx:panelItem id="x">
      <amx:panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
   <amx:panelItem id="y">
      <amx:panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
</panelSplitter>
```

For more examples, see the UIDemo application, available from **File** > **New** > **MAF Examples**.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.10 How to Use a Spacer Component

Use the Spacer (`spacer`) component to create an area of blank space with a purpose to separate components on a MAF AMX page. You can include vertical and horizontal spaces in a page using the `height` (for vertical spacing) and `width` (for horizontal spacing) attributes of the `spacer`:

To add the Spacer component:

1. In the **Components** window, drag and drop a **Spacer** onto the MAF AMX page.
2. Use the **Properties** window to set the attributes of the component. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the `spacer` element defined in a MAF AMX file.

```
<amx:outputText id="ot1" value="This is a long piece of text for this page..."/>
<amx:spacer id="s1" height="10"/>
```

```
<amx:outputText id="ot2" value="This is some more lengthy text..."/>
```

12.2.11 How to Use a Table Layout Component

Use the Table Layout (`tableLayout`) component to display data in a typical table format that consists of rows containing cells.

The Row Layout (`rowLayout`) component represents a single row in the Table Layout. The Table Layout component must contain either one or more Row Layout components or Iterator components that can produce Row Layout components.

The CellFormat (`cellFormat`) component represents a cell in the Row Layout. The Row Layout component must contain either one or more CellFormat components, Iterator components, Attribute List Iterator components, or Facet Definition components that can produce CellFormat components.

The Table Layout structure does not allow cell contents to use percentage heights nor can a height be assigned to the overall table structure as a whole. For details, see the description of the following attributes in the *Tag Reference for Oracle Mobile Application Framework*:

- `layout` and `width` attributes of the Table Layout component
- `width` and `height` attributes of the Row Layout component

To add the Table Layout component:

1. In the **Components** window, drag and drop a **Table Layout** onto the MAF AMX page.
2. Insert the desired number of Row Layout, Iterator, Attribute List Iterator, or Facet Definition child components into the Table Layout component.
3. Insert Cell Format, Iterator, Attribute List Iterator, or Facet Definition child components into each Row Layout component.
4. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The example below shows the `tableLayout` element and its children defined in a MAF AMX file.

```
<amx:tableLayout id="tableLayout1"
                 rendered="#{pageFlowScope.pRendered}"
                 styleClass="#{pageFlowScope.pStyleClass}"
                 inlineStyle="#{pageFlowScope.pInlineStyle}"
                 borderWidth="#{pageFlowScope.pBorderWidth}"
                 cellPadding="#{pageFlowScope.pCellPadding}"
                 cellSpacing="#{pageFlowScope.pCellSpacing}"
                 halign="#{pageFlowScope.pHalign}"
                 layout="#{pageFlowScope.pLayoutTL}"
                 shortDesc="#{pageFlowScope.pShortDesc}"
                 summary="#{pageFlowScope.pSummary}"
                 width="#{pageFlowScope.pWidth}">
   <amx:rowLayout id="rowLayout1">
      <amx:cellFormat id="cellFormatA" rowSpan="2" halign="center">
         <amx:outputText id="otA" value="Cell A"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatB" rowSpan="2" halign="center">
         <amx:outputText id="otB" value="Cell B (wide content)"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatC" rowSpan="2" halign="center">
         <amx:outputText id="otC" value="Cell C"/>
```

```
      </amx:cellFormat>
   </amx:rowLayout>
   <amx:rowLayout id="rowLayout2">
      <amx:cellFormat id="cellFormatD" halign="end">
         <amx:outputText id="otD" value="Cell D"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatE">
         <amx:outputText id="otE" value="Cell E"/>
      </amx:cellFormat>
   </amx:rowLayout>
</amx:tableLayout>
```

12.2.12 How to Use a Masonry Layout Component

The Masonry Layout (`masonryLayout`) is a container-type component that presents its child components as tiles arranged in columns and rows similar to a dashboard. The size of each column and row is fixed and defined in CSS. This size is independent of the size of the Masonry Layout component itself: the number of displayed columns may change depending on the Masonry Layout size, but the tile size does not change. In addition, the tile size is independent of its content.

A tile is represented by the Masonry Layout Item (`masonryLayoutItem`) component whose content is provided by various MAF AMX UI components. A tile can occupy more than one column and row (for example, a tile can occupy three columns and one row). MAF AMX provides the following predefined set of tile sizes available through the `dimension` attribute of the `masonryLayoutItem`:

- 1x1: one column and one row.
- 1x2: one column and two rows.
- 1x3: one column and three rows.
- 2x1: two columns and one row.
- 2x2: two columns and two rows.
- 2x3: two columns and three rows.
- 3x1: three columns and one row.
- 3x2: three columns and two rows

You can redefine the appearance of the Masonry Layout component by creating additional sizes in the `.amx-masonryLayoutItem` section of the CSS.

The space between rows and columns is also specified in the CSS.

The Masonry Layout component always attempts to make the best use of available space by positioning tiles where they fit via filling gaps left earlier in the layout. When the end user rotates the mobile device, the tiles rearrange themselves to fill the space optimally. This functionality is enabled via the `MasonryReorderEvent` that is fired by the Masonry Layout component every time the arrangement of the Masonry Layout Item changes.

To add the Masonry Layout component:

1. In the **Components** window, drag and drop a **Masonry Layout** onto the MAF AMX page.
2. Insert the desired number of Masonry Layout Item components and their child UI components into the Masonry Layout component.

3. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

he following example shows the masonryLayout element and masonryLayoutItem elements as well as their children defined in a MAF AMX file.

```
<amx:masonryLayout id="ml1"
                   initialOrder="#{pageFlowScope.componentProperties.order}">
   <amx:masonryLayoutItem id="mt1"
                    dimension="#{pageFlowScope.componentProperties.myTeamExpanded
? '3x1' : '1x1'}"
                    rendered="#{pageFlowScope.componentProperties.myTeam}">
      <amx:panelGroupLayout id="pgl9"
                            layout="vertical"
                            inlineStyle="margin: 6px; padding: 0px; border:
none">
         <amx:outputText value="My Team"
                         id="ot4"
                         inlineStyle="color: gray"/>
         <amx:panelGroupLayout id="pgl1"
                               layout="horizontal"
                               scrollPolicy="scroll"
                               inlineStyle="margin: 0px; padding: 2px; border:
none">
            <amx:panelGroupLayout id="pgl10"
                                  inlineStyle="margin: 0px; padding: 2px; border:
none">
               <amx:image id="i8"
                          source="/images/people/TerryLuca.png"
                          shortDesc="Terry Luca"/>
               <amx:outputText value="Terry Luca"
                               id="ot9"
                               inlineStyle="font-size: 12px; color: gray"/>
            </amx:panelGroupLayout>
            <amx:panelGroupLayout id="pgl11"
                                  inlineStyle="margin: 0px; padding: 2px; border:
none">
               <amx:image id="i9"
                          source="/images/people/SusanWong.png"
                          shortDesc="Susan Wong"/>
               <amx:outputText value="Susan Wong"
                               id="ot12"
                               inlineStyle="font-size: 12px; color: gray"/>
            </amx:panelGroupLayout>
            <amx:panelGroupLayout id="pgl12"
                                  inlineStyle="margin: 0px; padding: 2px; border:
none">
               <amx:image id="i10"
                          source="/images/people/RaviChouhan.png"
                          shortDesc="Ravi Chouhan"/>
               <amx:outputText value="Ravi Chouhan"
                               id="ot11"
                               inlineStyle="font-size: 12px; color: gray"/>
            </amx:panelGroupLayout>
            <amx:panelGroupLayout id="pgl13"
                                  inlineStyle="margin: 0px; padding: 2px; border:
none">
               <amx:image id="i11"
                          source="/images/people/KathyGreen.png"
                          shortDesc="Kathy Green"/>
               <amx:outputText value="Kathy Green"
```

```
                                id="ot10"
                                inlineStyle="font-size: 12px; color: gray"/>
            </amx:panelGroupLayout>
            <amx:panelGroupLayout id="pgl16"
                                  inlineStyle="margin: 0px; padding: 2px; border:
none">
               <amx:image id="i5"
                          source="/images/people/StellaBaumgardner.png"
                          shortDesc="Stella Baum"/>
               <amx:outputText value="Stella Baum"
                                id="ot3"
                                inlineStyle="font-size: 12px; color: gray"/>
            </amx:panelGroupLayout>
         </amx:panelGroupLayout>
      </amx:panelGroupLayout>
   </amx:masonryLayoutItem>
   <amx:masonryLayoutItem id="mt2"
                       dimension="#{pageFlowScope.componentProperties.socialExpanded
? '3x1' : '1x1'}"
                       rendered="#{pageFlowScope.componentProperties.social}">
      <amx:panelGroupLayout id="pgl2"
                            inlineStyle="margin: 6px; padding: 0px; border: none">
         <amx:panelGroupLayout id="pgl22"
                               layout="vertical"
                               inlineStyle="margin: 0px; padding: 0px; border:
none">
            <amx:outputText value="Social"
                            id="ot2"
                            inlineStyle="color: gray"/>
            <amx:spacer id="s5" height="6"/>
            <amx:outputText value="New Conversations"
                            id="ot14"
                            inlineStyle="color: gray; font-size: 15px"/>
            <amx:outputText value="6"
                            id="ot15"
                            inlineStyle="font-size:34px; color: #EE8A11"/>
            <amx:outputText value="New Followers"
                            id="ot17"
                            inlineStyle="color: gray; font-size: 15px"/>
            <amx:outputText value="5"
                            id="ot16"
                            inlineStyle="font-size:34px; color: #EE8A11"/>
         </amx:panelGroupLayout>
      </amx:panelGroupLayout>
   </amx:masonryLayoutItem>
   <amx:masonryLayoutItem id="mt3"
      ...
   </amx:masonryLayoutItem>
</amx:masonryLayout>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.13 How to Use an Accessory Layout Component

You use the Accessory Layout (`accessoryLayout`) component in a List View within its List Item child component to enable dragging of the content left or right to reveal optional content areas.

Typically, the Accessory Layout contains two child Facet components: start and end. If the drag gesture exceeds sufficiently beyond the facet's width, you may allow such a

gesture to trigger a tap on one of the child components in that content area. The revealed facet content is usually hidden when either another Accessory Layout's content is revealed or when focus moves to some other component. However, if the content was revealed using an accessibility trigger, it would not be hidden when the focus moves; otherwise, the end user would not be able to use links in that content area.

To add an Accessory Layout component:

1. In the **Components** window, drag and drop a **Deck** onto the MAF AMX page.
2. Insert the desired number of Transition operations and child UI components into the Deck component.
3. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the `accessoryLayout` element defined in a MAF AMX file, with a start and an end facet used as child components.

```
amx:listView id="lv1">
   <amx:listItem id="liSimple">
      <amx:showPopupBehavior popupId="itemPopup"
                             type="action"
                             alignId="pp1"
                             align="overlapMiddleCenter"/>
      <amx:accessoryLayout id="alSimple"
               rendered="#{pageFlowScope.componentProperties.rendered}"
               inlineStyle="#{pageFlowScope.componentProperties.inlineStyle}"
               styleClass="#{pageFlowScope.componentProperties.styleClass}"
               contentStyle="#{pageFlowScope.componentProperties.contentStyle}"
               contentClass="#{pageFlowScope.componentProperties.contentClass}"
               startDesc="#{pageFlowScope.componentProperties.startDesc}"
               startWidth="#{pageFlowScope.componentProperties.startWidth}"
               startStyle="#{pageFlowScope.componentProperties.startStyle}"
               startClass="#{pageFlowScope.componentProperties.startClass}"
               startFullTriggerSelector="#{pageFlowScope.componentProperties.
                                                   startFullTriggerSelector}"
               endDesc="#{pageFlowScope.componentProperties.endDesc}"
               endWidth="#{pageFlowScope.componentProperties.endWidth}"
               endStyle="#{pageFlowScope.componentProperties.endStyle}"
               endClass="#{pageFlowScope.componentProperties.endClass}"
               endFullTriggerSelector="#{pageFlowScope.componentProperties.
                                                 endFullTriggerSelector}">
         <amx:facet name="start">
            <amx:commandLink id="clStartSimple"
                             text="Start"
                             styleClass="full-trigger">
               <amx:showPopupBehavior popupId="startPopup"
                                      type="action"
                                      alignId="pp1"
                                      align="overlapMiddleCenter"/>
            </amx:commandLink>
         </amx:facet>
         <amx:facet name="end">
            <amx:commandLink id="clEndSimple"
                             text="End"
                             styleClass="full-trigger">
               <amx:showPopupBehavior popupId="endPopup"
                                      type="action"
                                      alignId="pp1"
                                      align="overlapMiddleCenter"/>
```

```
            </amx:commandLink>
          </amx:facet>
          <outputText id="otContentSimple" value="Simple example"/>
       </amx:accessoryLayout>
    </amx:listItem>
    ...
</amx:listView>
```

If your goal is to have some of the links hidden when in a full gesture, you can set the `styleClass` attribute of `commandLink` elements to `adfmf-accessoryLayout-hideWhenFull`.

For more examples, see the CompGallery sample applicationThe sample applications are available from **File** > **New** > **MAF Examples**.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.14 How to Use a Deck Component

The Deck (`deck`) component represents a container that shows one of its child components at a time. The transition from one displayed child component (defined by the `displayedChild` attribute) to another is enabled by the Transition (`transition`) operation, which can take a form of animation. Transition occurs by means of fading in, sliding and flipping from different directions, as well as covering and revealing child components.

The Deck can be navigated forward and backwards.

To add the Deck component:

1. In the **Components** window, drag and drop a **Deck** onto the MAF AMX page.
2. Insert the desired number of Transition operations and child UI components into the Deck component.
3. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The example below shows the `deck` element and its children defined in a MAF AMX file. The Deck component's `displayedChild` attribute to define which child component ID should be displayed. Typically, this is controlled by a component such as a Select One Button or other selection component.

```
<amx:deck id="deck1"
             rendered="#{pageFlowScope.pRendered}"
             styleClass="#{pageFlowScope.pStyleClass}"
             inlineStyle="width:95px;height:137px;overflow:hidden;
                                  #{pageFlowScope.pInlineStyle}"
             landmark="#{pageFlowScope.pLandmark}"
             shortDesc="#{pageFlowScope.pShortDesc}"
             displayedChild="#{pageFlowScope.pDisplayedChild}">

   <amx:transition triggerType="#{pageFlowScope.pTriggerType}"
                   transition="#{pageFlowScope.pTransition}"/>
   <amx:transition triggerType="#{pageFlowScope.pTriggerType2}"
                   transition="#{pageFlowScope.pTransition2}"/>
   <amx:commandLink id="linkCardBack1" text="Card Back">>
     <amx:setPropertyListener from="linkCardA"
                              to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
   <amx:commandLink id="linkCardA1" text="Card Front A">
   <amx:setPropertyListener id="setPL1"
```

```
                           from="linkCardB"
                           to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
   <amx:commandLink id="linkCardB1" text="Card Front B">
      <amx:setPropertyListener id="setPL2"
                               from="linkCardC"
                               to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
   <amx:commandLink id="linkCardC1" text="Card Front C">
      <amx:setPropertyListener id="setPL3"
                               from="linkCardD"
                               to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
   <amx:commandLink id="linkCardD1" text="Card Front D">
      <amx:setPropertyListener id="setPL4"
                               from="linkCardE"
                               to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
   <amx:commandLink id="linkCardE1" text="Card Front E">
      <amx:setPropertyListener id="setPL5"
                               from="linkCardBack"
                               to="#{pageFlowScope.pDisplayedChild}"/>
   </amx:commandLink>
</amx:deck>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.2.15 How to Use a Flex Layout Component

The Flex Layout (`flexLayout`) component provides either horizontal or vertical flexible flow for its child components that are allowed to grow, shrink, and wrap depending on the available space. You can create nested Flex Layout components.

This component is based on the Flexible Box Layout defined by CSS and supports a subset of its properties. For more information, see the W3C website at http://www.w3.org/TR/css-flexbox-1/.

To add the Flex Layout component:

1. In the **Components** window, drag and drop a **Flex Layout** onto the MAF AMX page.
2. Insert the desired number of child UI components, including other Flex Layout components, into the Flex Layout component.
3. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example shows the `flexLayout` element and its children defined in a MAF AMX file.

```
<amx:flexLayout id="fl1"
                itemFlexibility="equal"

orientation="#{pageFlowScope.componentProperties.layoutOrientation}"
                rendered="#{pageFlowScope.componentProperties.fl1Rendered}">

   <amx:panelStretchLayout inlineStyle="background-color: #ff0000; text-align:
center;"

rendered="#{pageFlowScope.componentProperties.p1Rendered}">
      <amx:facet name="center">
```

```
         <amx:outputText value="1" inlineStyle="font-size: 36px"/>
      </amx:facet>
   </amx:panelStretchLayout>

   <amx:panelStretchLayout inlineStyle="background-color: #00ff00; text-align:
center;"

rendered="#{pageFlowScope.componentProperties.p2Rendered}">
      <amx:facet name="center">
         <amx:outputText value="2" inlineStyle="font-size: 36px"/>
      </amx:facet>
   </amx:panelStretchLayout>

</amx:flexLayout>
```

12.2.16 How to Use the Fragment Component

The Fragment (`fragment`) component enables sharing of MAF AMX page contents. This component is used in conjunction with a MAF AMX fragment file. For more information, see Section 11.3.1.6, "Sharing the Page Contents."

To add the Fragment component:

1. In the **Components** window, drag and drop a **Fragment** to the MAF AMX page.
2. Use the **Insert Fragment** dialog to set the **Src** attribute of the Fragment to a fragment file (`.amxf`).
3. Optionally, use the **Structure** view to add child components, such as an **Attribute**, **Attribute List**, or **Facet**.
4. Use the **Properties** window to set the attributes of all added components. For more information, see *Tag Reference for Oracle Mobile Application Framework*.
5. Add the Facet Definition (`facetRef`) to the MAF AMX fragment file whose contents is to be included in the Fragment and set the `facetRef`'s `facetName` attribute to the name of a facet.

The following example shows a fragment element added to a MAF AMX page.

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:panelPage id="pp1">
      <amx:panelGroupLayout layout="vertical"
                            id="itemPgl"
                            styleClass="amx-style-groupbox">
         <amx:fragment id="f1"
                       src="/simpleFragment.amxf"
            <amx:attribute id="a1"
                           name="text"
                           value="defaultValue" />
            <amx:facet name="facet">
               <amx:outputText id="ot5" value="Fragment"/>
            </amx:facet>
         </amx:fragment>
      </amx:panelGroupLayout>
   </amx:panelPage>
</amx:view>
```

The following example shows the corresponding MAF AMX fragment file.

```
<amx:fragmentDef
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
   xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <fragment xmlns="http://xmlns.oracle.com/adf/mf/amx/fragment" id="f1">
      <description id="d1">Description of the fragment</description>
      <facet id="f2">
         <description id="d4">Description of the facet</description>
         <facet-name id="f3">facet1</facet-name>
      </facet>
      <attribute id="a1">
         <description id="d2">Description of an attribute</description>
         <attribute-name id="a2">text</attribute-name>
         <attribute-type id="at1">String</attribute-type>
         <default-value id="d3">defaultValue</default-value>
      </attribute>
   </fragment>
   <amx:panelGroupLayout id="pgl1">
      <amx:facetRef facetName="facet1" id="fr1"/>
      <amx:outputText value="#{text}" id="ot1"/>
   </amx:panelGroupLayout>
</amx:fragmentDef>
```

A MAF sample application called FragmentDemo demonstrates how to create and use the Fragment. This sample application is available from **File** > **New** > **MAF Examples**.

12.3 Creating and Using UI Components

You can use the following UI components when developing your MAF AMX application feature:

- Input Text (see Section 12.3.1, "How to Use the Input Text Component")
- Input Number Slider (see Section 12.3.2, "How to Use the Input Number Slider Component")
- Input Date (see Section 12.3.3, "How to Use the Input Date Component")
- Output Text (see Section 12.3.4, "How to Use the Output Text Component")
- Button (see Section 12.3.5, "How to Use Buttons")
- Link (see Section 12.3.6, "How to Use Links")
- Image (see Section 12.3.7, "How to Display Images")
- Checkbox (see Section 12.3.8, "How to Use the Checkbox Component")
- Select Many Checkbox (see Section 12.3.9, "How to Use the Select Many Checkbox Component")
- Select Many Choice (see Section 12.3.11, "How to Use the Select Many Choice Component")
- Boolean Switch (see Section 12.3.12, "How to Use the Boolean Switch Component")
- Choice (see Section 12.3.10, "How to Use the Choice Component")
- Select Button (see Section 12.3.13, "How to Use the Select Button Component")
- Radio Button (see Section 12.3.14, "How to Use the Radio Button Component")
- List View (see Section 12.3.15, "How to Use List View and List Item Components")
- Carousel (see Section 12.3.16, "How to Use the Carousel Component")

- Film Strip (see Section 12.3.17, "How to Use the Film Strip Component")
- Verbatim (see Section 12.3.18, "How to Use Verbatim Component")
- Output HTML (see Section 12.3.19, "How to Use Output HTML Component")
- Iterator (see Section 12.3.20, "How to Enable Iteration")
- Refresh Container (see Section 12.3.21, "How to Refresh Contents of UI Components"

You can also use the following miscellaneous components that include operations, listener-type components, and converters as children of the UI components when developing your MAF AMX application feature:

- Load Bundle (see Section 12.3.22, "How to Load a Resource Bundle")
- Action Listener (see Section 12.3.23, "How to Use the Action Listener")
- Set Property Listener (see Section 12.3.24, "How to Use the Set Property Listener")
- Client Listener (see
- Convert Date Time (see Section 12.3.26, "How to Convert Date and Time Values")
- Convert Number (see Section 12.3.27, "How to Convert Numerical Values")
- Navigation Drag Behavior (see Section 12.3.28, "How to Enable Drag Navigation")
- Loading Indicator Behavior (see Section 12.3.29, "How to Use the Loading Indicator")
- System Action Behavior (see

You add a UI component by dragging and dropping it onto a MAF AMX page from the Palette (see Section 11.3.2.1, "Adding UI Components"). Then you use the Properties window to set the component's attributes. For information on attributes of each particular component, see *Tag Reference for Oracle Mobile Application Framework*.

Note: On a MAF AMX page, you place UI components within layout components (see Section 12.2, "Designing the Page Layout"). UI elements are declared under the `<amx>` namespace, except data visualization components that are declared under the `<dvtm>` namespace.

You can add event listeners to some UI components. For more information, see Section 12.10, "Using Event Listeners." Event listeners are applicable to components for the MAF AMX runtime description on both iOS and Android-powered devices, but the listeners do not have any effect at design time.

For information on the UI components' support for accessibility, see Section 12.8, "Understanding MAF Support for Accessibility."

The user interface created for iOS platform using MAF AMX displays correctly in both the left-to-right and right-to-left language environments. In the latter case, the components originate on the right-hand side of the screen instead of on the left-hand side.

Note: MAF does not evaluate EL expressions at design time. If the value of a component's attribute is set to an expression, this value appears as such in OEPE's Preview and the component may look different at runtime.

A MAF sample application called CompGallery demonstrates how to create and configure MAF AMX UI components. Another sample application called UIDemo shows how to lay out components on a MAF AMX page. The sample applications are available from **File** > **New** > **MAF Examples**.

12.3.1 How to Use the Input Text Component

The Input Text (`inputText`) component represents an editable text field. The following types of Input Text components are available:

- Standard single-line Input Text, which is declared as an `inputText` element in a MAF AMX file:

```
<amx:inputText id="text1"
                  label="Text Input:"
                  value="#{myBean.text}" />
```

- Password Input Text:

```
<amx:inputText id="text1"
               label="Password Input:"
               value="#{myBean.text}"
               secret="true" />
```

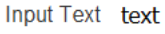
- Multiline Input Text (also known as text area):

```
<amx:inputText id="text1"
               label="Textarea:"
               value="#{myBean.text}"
               simple="true"
               rows="4" />
```

Figure 12–4 shows the Input Text component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:inputText id="inputText1"
               label="Input Text"
               value="text"/>
```

Figure 12–4 Input Text at Design Time



The `inputType` attribute lets you define how the component interprets the user input: as a text (default), email address, number, telephone number, or URL. These input types are based on the values allowed by HTML5.

To enable conversion of numbers, as well as date and time values that are entered in the Input Text component, you use the Convert Number (see Section 12.3.27, "How to Convert Numerical Values") and Convert Date Time (see Section 12.3.26, "How to Convert Date and Time Values") components.

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

On some mobile devices, when the end user taps an Input Text field, the keyboard is displayed (slides up). If an Input Text is the only component on a MAF AMX page, the input focus is on this field and the keyboard is displayed by default when the page loads.

A multiline Input Text may be displayed on a secondary page where it is the only component, in which case the multiline Input Text receives focus when the page loads and the keyboard becomes visible.

Input Text components render and behave differently on iOS and Android-powered devices: on iPhone and iPad, Input Text components may be displayed with or without a border.

When creating an Input Text component, consider the following:

- To input or edit content, the end user has to tap in the field, which triggers a blinking insertion cursor to be displayed at the point of the tap, allowing the end user to edit the content. If the field does not contain content, the insertion cursor is positioned at the start of the field.
- Fields represented by Input Text components may contain default text, typically used as a prompt. When the end user taps a key on the keyboard in such a field, the default text clears when Edit mode is entered. This behavior is enabled and configured through the Input Text's `hintText` attribute.
- Fields represented by Input Text components do not have a selected appearance. Selection is indicated by the blinking insertion cursor within the field.
- If the end user enters more text than fits in the field, the text content shifts left one character at a time as the typing continues.
- A multiline Input Text component is rendered as a rectangle of any height. This component supports scrolling when the content is too large to fit within the boundaries of the field: rows of text scroll up as the text area fills and new rows of text are added. The end user may flick up or down to scroll rows of text if there are more rows than can be displayed in the given display space. A scroll bar is displayed within the component to indicate the area is being scrolled.
- Password field briefly echoes each typed character, and then reverts the character to a dot to protect the password.
- The appearance and behavior of the Input Text component on iOS can be customized (see Section 12.3.1.1, "Customizing the Input Text Component").

12.3.1.1 Customizing the Input Text Component

MAF AMX provides support for the input capitalization and correction on iOS-powered devices, as well as the ability to override the return button located at the bottom right of the mobile devices's soft keypad (see Figure 12–5) such that this button would appear and act as the Go or Search button (see Figure 12–6) and trigger a `DataChangeEvent` for a single-line Input Text component.

Figure 12–5 Return Button on iOS-Powered Device at Runtime

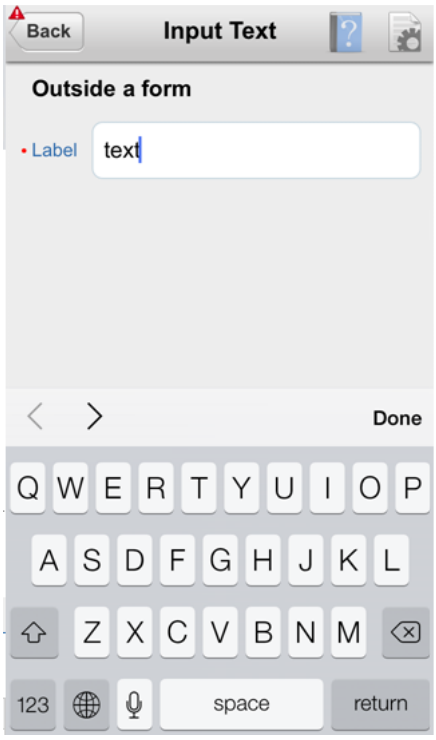


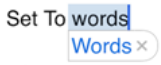
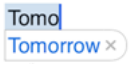
Figure 12–6 Go and Search Buttons on iOS at Runtime

Table 12–3 lists attributes of the Input Text component that allow you to customize the appearance and behavior of that component and the soft keypad that is used to enter values into fields represented by the Input Text.

Table 12–3 Input-Customizing Attributes of the Input Text Component

Attribute	Values	Description
`keyboardDismiss`	▪ `normal`: use the operating system's default. ▪ `go`: request the field to act like a trigger for behavior. ▪ `search`: request the field to act like a search field that triggers a lookup.	Indicates how the text field is to be used. If `go` or `search` is specified, dismissing the keypad causes the input to blur on both iOS and Android platforms. Even though support for Go and Search buttons on iOS is subject to change and might be terminated at any time, on some iOS-powered devices keypads are provided with the enhanced functionality. For example, instead of displaying a return button on a single-line text field, it might already replace it with a Go or Search button.

Table 12–3 (Cont.) Input-Customizing Attributes of the Input Text Component

Attribute	Values	Description
`autoCapitalize`	▪ `auto`: use the operating system's default. ▪ `sentences`: request that sentences comprising the input start with a capital letter. ▪ `none`: request that no capitalization be applied automatically to the input. ▪ `words`: request that words comprising the input start with capital letters. ▪ `characters`: request that each character typed as an input become capitalized.	Requests special treatment by iOS for capitalization of values while the field represented by the Input Text is being edited.  Note that setting this property has no impact on Android.
`autoCorrect`	▪ `auto`: use the operating system's default. ▪ `on`: request auto-correct support for the input. ▪ `off`: request auto-correct of the input be disabled.	Requests special treatment by iOS for correcting values while the field represented by the Input Text is being edited.  Note that setting this property has no impact on Android.

Since iOS provides limited support for auto-capitalization and auto-correction on its device simulator, you must test this functionality on an iOS device.

12.3.2 How to Use the Input Number Slider Component

The Input Number Slider (`inputNumberSlider`) component enables selection of numeric values from a range of values by using a slider instead of entering the value by using keys. The filled portion of the trough or track of the slider visually represents the current value.

The Input Number Slider may be used in conjunction with the Output or Input Text component to numerically show the value of the slider. The Input Text component also allows direct entry of a slider value: when the end user taps the Input Text field, the keyboard in numeric mode slides up; the keyboard can be dismissed by either using the slide-down button or by tapping away from the slider component.

The Input Number Slider component always shows the minimum and maximum values within the defined range of the component.

Note: The Input Number Slider component should not be used in cases where a precise numeric entry is required or where there is a wide range of values (for example, 0 to 1000).

The example below demonstrates the `inputNumberSlider` element defined in a MAF AMX file.

```
<amx:inputNumberSlider id="slider1" value="#{myBean.count}"/>
```

Figure 12–7 shows the Input Number Slider component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:inputNumberSlider id="inputNumberSlider1"
                       label="Input Number"
                       minimum="0"
                       maximum="20"
                       stepSize="1"
                       value="10"/>
```

Figure 12–7 Input Number Slider at Design Time

To enable conversion of numbers that are entered in the Input Number Slider component, you use the Convert Number component (see Section 12.3.27, "How to Convert Numerical Values").

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

Similar to other MAF AMX UI components, the Input Number Slider component has a normal and selected state. The component is in its selected state at any time it is touched. To change the slider value, the end user touches, and then interacts with the slider button.

The Input Number Slider component has optional `imageLeft` and `imageRight` attributes which point to images that can be displayed on either side of the slider to provide the end user with additional information.

12.3.3 How to Use the Input Date Component

The Input Date (`inputDate`) component presents a popup input field for entering dates. The default date format is the short date format appropriate for the current locale. For example, the default format in American English (ENU) is `mm/dd/yy`. The `inputType` attribute defines if the component accepts date, time, or date and time as an input. The time zone depends on the time zone configured for the mobile device, and, therefore, it is relative to the device. At runtime, the Input Date component has the device's native look and feel.

The example below demonstrates the `inputDate` element defined in a MAF AMX file. The `inputType` attribute of this component is set to the default value of `date`. If the `value` attribute is read-only, it can be set to either an EL expression or any other type of value; if `value` is not a read-only attribute, it can be specified only as an EL expression.

```
<amx:inputDate id="inputDate1" label="Input Date" value="#{myBean.date}"/>
```

For more information, see the following:

- *Tag Reference for Oracle Mobile Application Framework*

- HTML5 global dates and times defined by W3C
- CompGallery, available from **File** > **New** > **MAF Examples**.

12.3.4 How to Use the Output Text Component

MAF AMX provides the Output Text (`outputText`) component for you to use as a label to display text.

The example below demonstrates the `outputText` element defined in a MAF AMX file.

```
<amx:outputText id="ot1"
                value="output"
                styleClass="#{pageFlowScope.pStyleClass}"/>
```

Figure 12–8 shows the Output Text component displayed in the Preview pane.

Figure 12–8 Output Text at Design Time

You use the Convert Number (see Section 12.3.27, "How to Convert Numerical Values") and Convert Date Time (see Section 12.3.26, "How to Convert Date and Time Values") converters to facilitate the conversion of numerical and date-and-time-related data for the Output Text components.

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery and UIDemo, MAF sample applications available from **File** > **New** > **MAF Examples**.

12.3.5 How to Use Buttons

The Button (`commandButton`) component is used to trigger actions (for example, Save, Cancel, Send) and to enable navigation to other pages within the application (for example, Back: see Section 12.3.5.7, "Enabling the Back Button Navigation" for more information).

You may use the Button in one of the following ways:

- Button with a text label.
- Button with a text label and an image icon.

> **Note:** You may define the icon image and placement as left or right of the text label.

- Button with an image icon only (for example, the " + " and " - " buttons for adding or deleting records).

MAF supports one default Button type for the following three display areas:

1. Buttons that appear in the top header bar: in MAF AMX pages, the header is represented by the Panel Page component (see Section 12.2.2, "How to Use a Panel Page Component") in combination with the header, primary, and secondary facets, which is typical on iPhones:
   - Header Facet contains the page title.

- Primary Action Facet represents an area that appears in the left corner of the header bar and typically hosts Button or Link components, but can contain any component type.
- Secondary Action Facet represents an area that appears in the right corner of the header bar and typically hosts Button or Link components, but can contain any component type.

2. Buttons that appear in the content area of a page.
3. Buttons that appear in the footer bar of a page. In MAF AMX pages, the footer is represented by the Panel Page component (see Section 12.2.2, "How to Use a Panel Page Component") in combination with the footer facet:
   - Footer Facet represents an area that appears below the content area and typically hosts Button or Link components, but can contain any component type.

All Button components of any type have three states:

1. Normal.
2. Activated: represents appearance when the Button is tapped or touched by the end user. When a button is tapped (touch and release), the button action is performed. Upon touch, the activated appearance is displayed; upon release, the action is performed. If the end user touches the button and then drags their finger away from the button, the action is not performed. However, for the period of time the button is touched, the activated appearance is displayed.
3. Disabled.

The appearance of a Button component is defined by its `styleClass` attribute that you set to an `adfmf-commandButton-<style>`. You can apply any of the styles detailed in Table 12–4 to a Button placed in any valid location within the MAF AMX page.

Table 12–4 Main Button Styles

Button Style Name	Description
Default	The default style of a Button placed: ■ In any of the Panel Page facets (Primary, Secondary, Header, Footer). For more information, see Section 12.3.5.1, "Displaying Default Style Buttons." ■ Anywhere in the content area of a MAF AMX page. This style is used for buttons that are to perform specific actions within a page, typically based on their location or context within the page.
Back	The back style of a Button placed in any of the Panel Page facets (Primary, Secondary, Header, Footer). This style may be applied to the default Button to give the "back to page" appearance. This button style is typical for "Back to Springboard" or any "Back to Page" buttons. For more information, see Section 12.3.5.2, "Displaying Back Style Buttons."
Highlight	The highlight style of a Button placed in any of the Panel Page facets (Primary, Secondary, Header, Footer) or the content area of a MAF AMX page. This style may be added to a Button to provide the iPhone button appearance typical of Save (or Done) buttons. For more information, see Section 12.3.5.3, "Displaying Highlight Style Buttons."
Alert	The Alert style adds the delete appearance to a button. For more information, see Section 12.3.5.4, "Displaying Alert Style Buttons."

There is a Rounded style (`adfmf-commandButton-rounded`) that you can apply to a Button to decorate it with a thick rounded border (see Figure 12–9). You can define this style in combination with any other style.

Figure 12–9 Rounded Button at Design Time

MAF AMX provides a number of additional decorative styles (see Section 12.3.5.5, "Using Additional Button Styles").

There is a particular order in which MAF AMX processes the Button component's child operations and attributes. For more information, see Section 12.3.5.8, "What You May Need to Know About the Order of Processing Operations and Attributes."

12.3.5.1 Displaying Default Style Buttons

The following are various types of default style buttons that can be placed within Panel Page facets or content area:

- Normal, activated, or disabled Button with a text label only.
- Normal, activated, or disabled Button with an image icon only.

The examples below demonstrate the `commandButton` element declared in a MAF AMX file. The first example shows the definition of a default button with a text label.

```
<amx:panelPage id="pp1">
   <amx:facet name="primary">
      <amx:commandButton id="cb1"
                         text="Cancel"
                         action="cancel"
                         actionListener="#{myBean.rollback}"/>
   </amx:facet>
</amx:panelPage>
```

This example shows the definition of a default button with an image icon.

```
<amx:panelPage id="pp1">
   <amx:facet name="primary">
      <amx:commandButton id="cb1"
                         icon="plus.png"
                         action="add"
                         actionListener="#{myBean.AddItem}"/>
   </amx:facet>
</amx:panelPage>
```

The example below shows the `commandButton` element declared inside the Panel Page's footer facet.

```
<amx:panelPage id="pp1">
   <amx:facet name="footer">
      <amx:commandButton id="cb2"
                         icon="folder.png"
                         text="Move (#{myBean.mailcount})"
                         action="move"/>
   </amx:facet>
</amx:panelPage>
```

The following example demonstrates the `commandButton` element declared as a part of the Panel Page content area.

```
<amx:panelPage id="pp1">
   <amx:commandButton id="cb1"
                      text="Reply"
                      actionListener="#{myBean.share}"/>
</amx:panelPage>
```

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a sample application is available from **File** > **New** > **MAF Examples**

12.3.5.2 Displaying Back Style Buttons

The following are various types of back style buttons that are placed within Panel Page facets or content area:

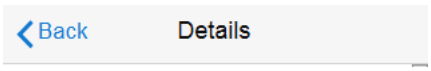
- Normal, activated, or disabled Button with a text label only.
- Normal, activated, or disabled Button with an image icon only:

The following example demonstrates the `commandButton` element declared in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText value="Details" id="ot1"/>
   </amx:facet>
   <amx:facet name="primary">
      <amx:commandButton id="cb1"
                         text="Back"
                         action="__back"/>
   </amx:facet>
   ...
</amx:panelPage>
```

Every time you place a Button component within the primary facet and set its `action` attribute to `__back`, MAF AMX automatically applies the back arrow styling to it, as Figure 12–10

Figure 12–10 Back Button at Design Time



For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.5.3 Displaying Highlight Style Buttons

Similar to other types of Buttons, highlight style buttons that are placed within Panel Page facets or content area can have their state as normal, activated, or disabled.

The following example demonstrates the `commandButton` element declared in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:facet name="secondary">
      <amx:commandButton id="cb2"
```

```
                        text="Save"
                        action="save"
                        styleClass="adfmf-commandButton-highlight"/>
   </amx:facet>
</amx:panelPage>
```

Figure 12–11 Highlight Button at Design Time

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

12.3.5.4 Displaying Alert Style Buttons

Alert style buttons placed within the Panel Page can have normal, activated, or disabled state.

The example below demonstrates the `commandButton` element declared in a MAF AMX file.

```
<amx:commandButton id="cb1"
                   text="Delete"
                   actionListener="#{myBean.delete}"
                   styleClass="adfmf-commandButton-alert" />
```

Figure 12–12 Alert Button at Design Time

For more information, illustrations, and examples, see the following:

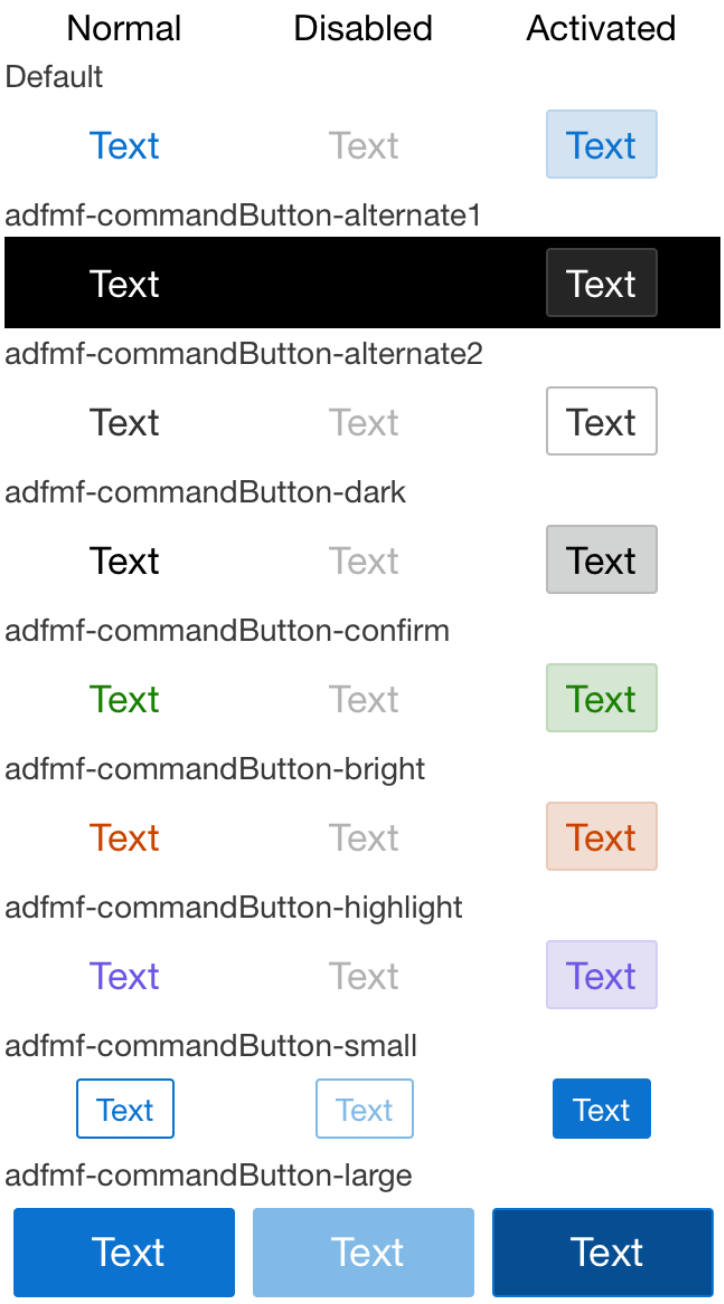
- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

12.3.5.5 Using Additional Button Styles

MAF AMX provides the following additional Button styles:

- Dark style
- Bright style
- Small style
- Large style
- Highlight style
- Confirm style
- Two varieties of the Alternate style

Figure 12–13 Additional Button Styles



12.3.5.6 Using Buttons Within the Application

In your MAF application, you can use the Button component within the following contexts:

- Navigation Bar
- The Content Area to perform specific actions
- Action Sheets
- Popup-style Alert Messages

Navigation Bar

MAF lets you create standard buttons for use on a navigation bar:

- Edit button allows the end user to enter an editing or content-manipulation mode.
- Cancel button allows the end user to exit the editing or content-manipulation mode without saving changes.
- Save button allows the end user to exit the editing or content-manipulation mode by saving changes.
- Done button allows the end user to exit the current mode and save changes, if any.
- Undo button allows the end user to undo the most recent action.
- Redo button allows the end user to redo the most recent undone action.
- Back button allows the end user to navigate back to the springboard.
- Back to Page button allows the end user to navigate back to the page identified by the button text label.
- Add button allows the end user to add or create a new object.

Content Area

Buttons that are positioned within the content area of a page perform a specific action given the location and context of the button within the page. These buttons may have a different visual appearance than buttons positioned with the navigation bar:

Action Sheets

An example of buttons placed within an action sheet is a group of Delete Note and Cancel buttons.

An action sheet button expands to the width of the display.

Alert Messages

An OK button can be placed within a validation message, such as a login validation after a failed password input.

12.3.5.7 Enabling the Back Button Navigation

MAF AMX supports navigation using the back button, with the default behavior of going back to the previously visited page. For more information, see Section 11.2.9, "How to Specify Action Outcomes Using UI Components."

If any Button component is added to the primary facet of a Panel Page that is equipped with the `__back` navigation, this Button is automatically given the back arrow visual styling (see Section 12.3.5.2, "Displaying Back Style Buttons"). To disable this, set the `styleClass` attribute to `amx-commandButton-normal`.

For more information, illustrations, and examples, see the following:

- Tag Reference for Oracle Mobile Application Framework
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.5.8 What You May Need to Know About the Order of Processing Operations and Attributes

The following is the order in which MAF AMX processes operations and attributes when such components as a Button, Link, and List Item are activated:

1. The following child operations are processed in the order they appear in the XML file:
   - Set Property Listener
   - Action Listener
   - Show Popup Behavior
   - Close Popup Behavior
2. The Action Listener (`actionListener`) attribute is processed and the associated Java method is invoked.
3. The Action (`action`) attribute is processed and any navigation case is followed.

12.3.6 How to Use Links

You use the Link (`commandLink`) component to trigger actions and enable navigation to other views.

The Link component can have any type of component defined as its child. By using such components as Set Property Listener (see Section 12.3.24, "How to Use the Set Property Listener"), Action Listener (see Section 12.3.23, "How to Use the Action Listener"), Show Popup Behavior, Close Popup Behavior see Section 12.2.8, "How to Use a Popup Component"), and Validation Behavior (see Section 12.9, "Validating Input") as children of the Link component, you can create an actionable area within which clicks and gestures can be performed.

By placing an Image component (see Section 12.3.7, "How to Display Images") inside a Link you can create a clickable image.

The example below demonstrates the `commandLink` element declared in a MAF AMX file.

```
<amx:commandLink id="cl1"
                 text="linked"
                 action="gotolink"
                 actionListener="#{myBean.doSomething}"/>
```

Figure 12–14 shows the basic Link component displayed in the Preview pane.

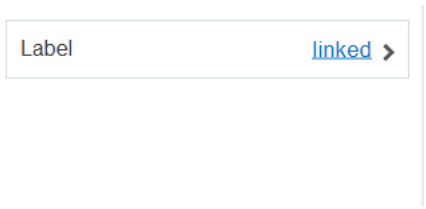
Figure 12–14 Link at Design Time

The following example demonstrates the `commandLink` element declared in a MAF AMX file. This component is placed within the `panelFormLayout` and `panelLabelAndMessage` components.

```
<amx:panelPage id="pp1">
   <amx:panelFormLayout id="form">
      <amx:panelLabelAndMessage id="panelLabelAndMessage1" label="Label">
         <amx:commandLink id="cl1"
                          text="linked"
                          action="gotolink"
                          actionListener="#{myBean.doSomething}"/>
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
</amx:panelPage>
```

Figure 12–15 shows the Link component placed within a form and displayed in the Preview pane.

Figure 12–15 Link Within Form at Design Time



There is a particular order in which MAF AMX processes the Link component's child operations and attributes. For more information, see Section 12.3.5.8, "What You May Need to Know About the Order of Processing Operations and Attributes."

MAF AMX provides another component which is similar to the Link, but which does not allow for navigation between pages: Link Go (`goLink`) component. You use this component to enable linking to external pages. Figure 12–16 shows the Link Go component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:goLink id="goLink1"
            text="Go Link"
            url="http://example.com"/>
```

Figure 12–16 Link Go at Design Time

Image is the only component that you can specify as a child of the Link Go component.

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application This sample application is available from **File** > **New** > **MAF Examples**

12.3.7 How to Display Images

MAF AMX enables the display of images on iOS and Android-powered devices using the Image (`image`) component represented by a bitmap.

In addition to placing an Image in a Button and List View, you can place it inside a Link component (see Section 12.3.6, "How to Use Links") to create a clickable image.

The example below demonstrates the `image` element definition in a MAF AMX file.

```
<amx:image id="i1"
           styleClass="prod-thumb"
           source="images/img-big-#{pageFlowScope.product.uid}.png" />
```

The following are supported formats on Android platform:

- GIF
- JPEG
- PNG

- BMP

The following are supported formats on iOS platform:

- PNG

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery and UIDemo, MAF sample applications available from **File** > **New** > **MAF Examples**.

12.3.8 How to Use the Checkbox Component

The Checkbox (`selectBooleanCheckbox`) component represents a check box that you create to enable single selection of `true` or `false` values, which allows toggling between selected and deselected states.

You can use the `label` attribute of the Checkbox component to place text to the left of the checkbox, and the `text` attribute places text on the right.

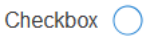
The example below demonstrates the `selectBooleanCheckbox` element declared in a MAF AMX file.

```
<amx:selectBooleanCheckbox id="check1"
                           label="Agree to the terms:"
                           value="#{myBean.bool1}"
                           valueChangeListener=
                              "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 12–17 shows the unchecked Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectBooleanCheckbox id="selectBooleanCheckbox1"
                           label="Checkbox"
                           value="false"
                           valueChangeListener=
                                "#{PropertyBean.ValueChangeHandler}"/>
```

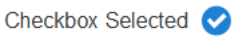
Figure 12–17 Unchecked Checkbox at Design Time




```
<amx:selectBooleanCheckbox id="selectBooleanCheckbox1"
                           label="Checkbox"
                           value="true"
                           valueChangeListener=
                                "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 12–18 shows the checked Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

Figure 12–18 Checked Checkbox Definition



For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application This sample application is available from **File** > **New** > **MAF Examples**

12.3.8.1 Support for Checkbox Components on iOS Platform

iOS does not support a native Checkbox component. The Boolean Switch is usually used in Properties pages to enable a boolean selection (see Section 12.3.12, "How to Use the Boolean Switch Component").

12.3.8.2 Support for Checkbox Components on Android Platform

Android provides support for a native Checkbox component. This component is used extensively on Settings pages to turn on or off individual setting values.

12.3.9 How to Use the Select Many Checkbox Component

The Select Many Checkbox (`selectManyCheckbox`) component represents a group of check boxes that you use to enable multiple selection of `true` or `false` values, which allows toggling between selected and deselected states of each check box in the group. The selection mechanism is provided by the Select Items or Select Item component (see Section 12.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Many Checkbox component.

> **Note:** The Select Many Checkbox component can contain more than one Select Item or Select Items components.

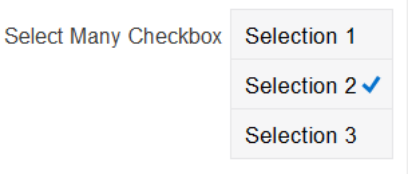
This example demonstrates the `selectManyCheckbox` element declared in a MAF AMX file.

```
<amx:selectManyCheckbox id="selectManyCheckbox1"
                        label="Select shipping options"
                        value="#{myBean.shipping}"
                        valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1"
                   label="Air"
                   value="#{myBean.shipping.air}"/>
   <amx:selectItem id="selectItem2"
                   label="Rail"
                   value="#{myBean.shipping.rail}"/>
   <amx:selectItem id="selectItem3"
                   label="Water"
                   value="#{myBean.shipping.water}"/>
</amx:selectManyCheckbox>
```

Figure 12–19 shows the Select Many Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectManyCheckbox id="selectManyCheckbox1"
                        label="Select Many Checkbox"
                        value="value2"
                        valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Selection 1" value="value1"/>
   <amx:selectItem id="selectItem2" label="Selection 2" value="value2"/>
   <amx:selectItem id="selectItem3" label="Selection 3" value="value3"/>
</amx:selectManyCheckbox>
```

Figure 12–19 Select Many Checkbox at Design Time



For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

12.3.9.1 What You May Need to Know About the User Interaction with Select Many Checkbox Component

MAF AMX provides two alternative ways for displaying the Select Many Checkbox component: pop-up style (default) and list style that is used when the number of available choices exceeds the device screen size.

The end user interaction with a pop-up style Select Many Checkbox component on both iPhone and iPad occurs as follows: when the end user taps the component, the list of choices is displayed in a popup. To make a choice, the end user taps one or more choices. To save the selections, the end user either taps outside the popup or closes the popup using the close (" x ") button.

Upon closing of the popup, the value displayed in the component is updated with the selected value.

When the number of choices exceed the dimensions of the device, a full-page popup containing a scrollable List View (see Section 12.3.15, "How to Use List View and List Item Components") is generated.

The end user interaction with a list-style Select Many Checkbox component on both iPhone and iPad occurs as follows: when the end user taps the component, the list of choices is displayed. To make a choice, the end user scrolls up or down to browse available choices, and then taps one or more choices. To save the selections, the end user taps the close (" x ") button.

Upon closing of the list, the value displayed in the component is updated with the selected value.

> **Note:** In both cases, there is no mechanism provided to cancel the selection.

12.3.10 How to Use the Choice Component

The Choice (`selectOneChoice`) component represents a combo box that is used to enable selection of a single value from a list. The selection mechanism is provided by the Select Items or Select Item component (see Section 12.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Choice component.

> **Note:** The Choice component can contain more than one Select Items or Select Item components.

This example demonstrates the `selectOneChoice` element definition with the `selectItem` subelement in a MAF AMX file.

```
<amx:selectOneChoice id="choice1"
                     label="Your state:"
                     value="#{myBean.myState}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Alaska" value="AK"/>
   <amx:selectItem id="selectItem2" label="Alabama" value="AL"/>
   <amx:selectItem id="selectItem3" label="California" value="CA"/>
   <amx:selectItem id="selectItem4" label="Connecticut" value="CT"/>
</amx:selectOneChoice>
```

This example demonstrates the `selectOneChoice` element definition with the `selectItems` subelement in a MAF AMX file.

```
<amx:selectOneChoice id="choice1"
                     label="Your state:"
                     value="#{myBean.myState}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItems id="selectItems1" value="myBean.allStates"/>
</amx:selectOneChoice>
```

Figure 12–20 shows the Choice component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneChoice id="selectOneChoice1"
                     label="Choice"
                     value="value1"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Value 1" value="value1"/>
   <amx:selectItem id="selectItem2" label="Value 2" value="value2"/>
   <amx:selectItem id="selectItem3" label="Value 3" value="value3"/>
</amx:selectOneChoice>
```

Figure 12–20 Choice at Design Time

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

12.3.10.1 What You May Need to Know About the User Interaction with Choice Component on iOS Platform

MAF AMX provides two alternative ways for displaying the Choice component: pop-up style and drop-down style.

On an iPhone, the end user interaction with a native Choice component occurs as follows: when the end user taps the component, the list of choices is displayed, with the first option selected by default. To make a choice, the end user scrolls up or down to browse available choices. To save the selection, the end user taps Done in the tool bar.

On an iPad, the user interaction is similar to the interaction on an iPhone, except the following:

- The list of choices is displayed in a popup dialog.

- iPad styling is implemented around the list of choices, with a notch used to indicate the source of the list.

To close the list of choices without selecting an item, the end user must tap outside the popup dialog.

> **Note:** The UI to display the list of choices and the tool bar are native to the browser and cannot be styled using CSS.

List values within the Choice component may be displayed as disabled.

When the number of choices exceeds the dimensions of the device display, a list page is generated that may be scrolled in a native way.

12.3.10.2 What You May Need to Know About the User Interaction with Choice Component on Android Platform

The end user interaction with a native Choice component on an Android-powered device occurs as follows: when the end user taps the component, the list of choices in the form of a popup dialog is displayed. A simple popup is displayed if the number of choices fits within the dimensions of the device, in which case:

- A single tap on an item from the selection list selects that item and closes the popup; the selection is reflected in the Choice component label.
- A single tap outside the popup or a click on the Back key closes the popup with no changes applied.

If the number of choices to be displayed does not fit within the device dimensions, the popup contains a scrollable list, in which case:

- A single tap on an item from the selection list selects that item and closes the popup; the selection is reflected in the Choice component label.
- A click on the Back key closes the popup with no changes applied.

12.3.10.3 What You May Need to Know About Differences Between Select Items and Select Item Components

The Select Items (`selectItems`) component provides a list of objects that can be selected in both multiple-selection and single-selection components.

The Select Item (`selectItem`) component represents a single selectable item of selection components.

12.3.11 How to Use the Select Many Choice Component

The Select Many Choice (`selectManyChoice`) component allows selection of multiple values from a list. The selection mechanism is provided by the Select Items or Select Item component (see Section 12.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Many Checkbox component.

> **Note:** The Select Many Checkbox component can contain more than one Select Items or Select Item components.

The following example demonstrates the `selectManyChoice` element declared in a MAF AMX file.

```
<amx:selectManyChoice id="check1"
                      label="Select Option:"
                      value="#{myBean.shipping}"
                      valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1"
                   label="Signature Required"
                   value="signature" />
   <amx:selectItem id="selectItem2"
                   label="Insurance"
                   value="insurance" />
   <amx:selectItem id="selectItem3"
                   label="Delivery Confirmation"
                   value="deliveryconfirm"/>
</amx:selectManyChoice>
```

Figure 12–21 Select Many Choice at Design Time

```
<amx:selectManyChoice id="check1"
                      label="Select Shipping Options:"
                      value="#{myBean.shipping}">
   <amx:selectItems id="selectItems1" value="#{myBean.shippingOptions}"/>
</amx:selectManyChoice>
```

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

The look and behavior of the Select Many Choice component on all supported devices is almost identical to the Select Many Checkbox component (see Section 12.3.9, "How to Use the Select Many Checkbox Component" for more information).

12.3.12 How to Use the Boolean Switch Component

The Boolean Switch (`selectBooleanSwitch`) component allows editing of boolean values as a switch metaphor instead of a checkbox.

Similar to other MAF AMX UI components, this component has a normal and selected state. To toggle the value, the end user taps (touches and releases) the switch once. Each tap toggles the switch.

The following example demonstrates a `selectBooleanSwitch` element defined in a MAF AMX file.

```
<amx:selectBooleanSwitch id="switch1"
                         label="Flip switch:"
                         onLabel="On"
                         offLabel="Off"
                         value="#{myBean.bool1}"
                         valueChangeListener=
                                "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 12–22 shows the Boolean Switch component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectBooleanSwitch id="selectBooleanSwitch1"
                         label="Switch"
                         value="value1"
                         valueChangeListener=
                                "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 12–22 Boolean Switch at Design Time

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.12.1 What You May Need to Know About Support for Boolean Switch Components on iOS Platform

On iOS, Boolean Switch components are often used on Settings pages to enable or disable an attribute value.

12.3.13 How to Use the Select Button Component

The Select Button (`selectOneButton`) component represents a button group that lists actions, with a single button active at any given time. The selection mechanism is provided by the Select Items or Select Item component (see Section 12.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Button component.

> **Note:** The Select Button component can contain more than one Select Items or Select Item components.

This example demonstrates the `selectOneButton` element defined in a MAF AMX file.

```
<amx:selectOneButton id="bg1"
                     value="#{myBean.myState}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Yes" value="yes"/>
   <amx:selectItem id="selectItem2" label="No" value="no"/>
   <amx:selectItem id="selectItem3" label="Maybe" value="maybe"/>
</amx:selectOneButton>
```

Figure 12–23 shows the Select Button component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneButton id="selectOneButton1"
                     label="Select Button"
                     value="value1"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Value 1" value="value1"/>
   <amx:selectItem id="selectItem2" label="Value 2" value="value2"/>
```

```
   <amx:selectItem id="selectItem3" label="Value 3" value="value3"/>
</amx:selectOneButton>
```

Figure 12–23 Select Button at Design Time

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.14 How to Use the Radio Button Component

The Radio Button (`selectOneRadio`) component represents a group of radio buttons that lists available choices. The selection mechanism is provided by the Select Items or Select Item component (see Section 12.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Radio Button component.

> **Note:** The Radio Button component can contain more than one Select Items or Select Item components.

The following examples demonstrates the `selectOneRadio` element definition in a MAF AMX file. The first example shows a radio button definition using a select item component.

```
<amx:selectOneRadio id="radio1"
                    label="Choose a pet:"
                    value="#{myBean.myPet}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Cat" value="cat"/>
   <amx:selectItem id="selectItem2" label="Dog" value="dog"/>
   <amx:selectItem id="selectItem3" label="Hamster" value="hamster"/>
   <amx:selectItem id="selectItem4" label="Lizard" value="lizard"/>
</amx:selectOneRadio>
```

This example shows a radio button definition using a select items component.

```
<amx:selectOneRadio id="radio1"
                    label="Choose a pet:"
                    value="#{myBean.myPet}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItems id="selectItems1" value="myBean.allPets"/>
</amx:selectOneRadio>
```

Figure 12–24 shows the Boolean Switch component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneRadio id="selectOneRadio1"
                     label="Radio Button"
                     value="value1"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1" label="Value 1" value="value1"/>
   <amx:selectItem id="selectItem2" label="Value 2" value="value2"/>
   <amx:selectItem id="selectItem3" label="Value 3" value="value3"/>
```

```
</amx:selectOneRadio>
```

Figure 12–24 Radio Button at Design Time

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.15 How to Use List View and List Item Components

Use the List View (`listView`) component to display data as a list of choices where the end user can select one or more options.

Typically, the List Item (listItem) component represents a single item in the List View component, where you place a List Item component inside the List View to lay out and style a list of data items. Each item can contain more than one List Item component, in which case List Item components fill the item (line) and excess List Item components wrap onto the subsequent lines. You configure this by setting the List View's layout attribute to cards (the default layout is rows and displays one List Item component per item within the list). For more information, see Section 12.3.15.3, "Configuring The List View Layout."

The List View allows you to define one of the following:

- A selectable item that is replicated based on the number of items in the list (collection).
- A static item that is produced by adding a child List Item component without specifying the List View's var and value attributes. You can add as many of these static items as necessary, which is useful when you know the contents of the list at design time. In this case, the list is not editable and behaves like a set of menu items.

You can create the following types of List View components:

- Basic List

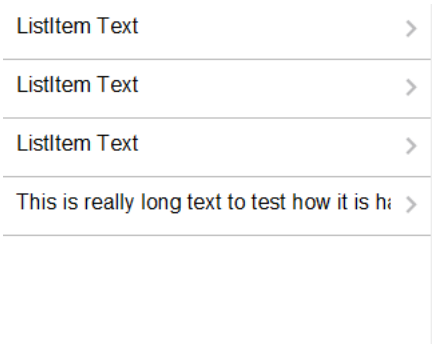
  This example shows the `listView` element defined in a MAF AMX file. This definition corresponds to the basic component.

```
<amx:listView id="listView1"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem id="listItem1">
      <amx:outputText id="outputText1" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem2">
      <amx:outputText id="outputText3" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem3">
      <amx:outputText id="outputText5" value="ListItem Text"/>
   </amx:listItem>
```

```
   <amx:listItem id="listItem4">
      <amx:outputText id="outputText7"
                      value="This is really long text to test how it is
handled"/>
   </amx:listItem>
</amx:listView>
```

Figure 12–25 demonstrates a basic List View component at design time.

Figure 12–25 Basic List View at Design Time



This example shows another definition of the `listView` element in a MAF AMX file. This definition also corresponds to the basic component; however, the value of this List View is provided by a collection.

```
<amx:listView id="list1"
              value="#{myBean.listCollection}"
              var="row"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem actionListener="#{myBean.selectRow}"
                 showLinkIcon="false"
                 id="listItem1">
      <amx:outputText value="#{row.name}" id="outputText1"/>
   </amx:listItem>
</amx:listView>
```

Note: Currently, when a text string in an Output Text inside a List Item is too long to fit on one line, the text does not wrap at the end of the line. You can prevent this by adding `"white-space: normal;"` to the `inlineStyle` attribute of the subject Output Text child component.

- List with icons

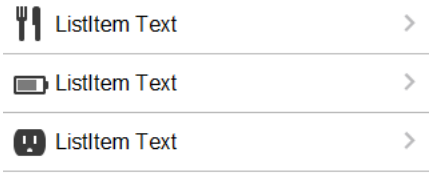
  This example shows the `listView` element defined in a MAF AMX file. This definition corresponds to the component with icons.

```
<amx:listView id="list1"
              value="#{myBean.listCollection}"
              var="row"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem id="listItem1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
```

```
            <amx:cellFormat id="cf11" width="40px" halign="center">
               <amx:image id="image1" source="#{row.image}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf12" width="100%" height="43px">
               <amx:outputText id="outputText1" value="#{row.desc}"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–26 demonstrates a List View component with icons and text at design time.

Figure 12–26 List View with Icons at Design Time



- List with search
- List with dividers. This type of list allows you to group data and show order. Attributes of the List View component define characteristics of each divider. For information about attributes, see *Tag Reference for Oracle Mobile Application Framework*.

  MAF AMX provides a list divider that can do the following:

  – Collapse its contents independently.

  – Show a count of items in each divider.

  – Collapse at the same time.

  This example shows the `listView` element defined in a MAF AMX file. This definition corresponds to the component with collapsible dividers and item counts.

```
<amx:listView id="list1"
          value="#{bindings.data.collectionModel}"
          var="row"
          collapsibleDividers="true"
          collapsedDividers="#{pageFlowScope.mylistDisclosedDividers}"
          dividerMode="all"
          dividerAttribute="type"
          showDividerCount="true"
          showMoreStrategy="autoScroll"
          bufferStrategy="viewport"
          fetchSize="10">
   <amx:listItem>
      <amx:outputText id="ot1" value="#{row.name}">
   </amx:listItem>
</amx:listView>
```

Note: Data in the list with dividers must be sorted by the `dividerAttribute` because this type of list does not sort the data; instead, it expects the data it receives to be already sorted.

Note: Dividers are not displayed when a List View component is in edit mode (that is, its `editMode` attribute is specified).

When dividers are visible, the end user can quickly navigate to a specific divider using the List View's localized alphabetical index utility, which is available for List View components whose `dividerMode` attribute is set to `firstLetter`. You can disable this utility by setting the `sectionIndex` attribute to `off`.

The index utility (indexer) consists of an index bar and index item and has the following characteristics:

- If the list contains unsorted data or duplicate dividers, the index item points to the first occurrence in the list.
- Only available letters are highlighted in the index, and only those highlighted become active. This is triggered by the change in the data model (for example, when the end user taps on More row item).
- The index is not case-sensitive.
- Unknown characters are hidden under the hash (#) sign.

The indexer letters can only be activated (tapped) on rows that have been loaded into the list. For example, if the List View, using its `fetchSize` attribute, has loaded rows up to the letter C, the indexer enables letters from A to C. Other letters appear on the indexer when more rows are loaded into it.

Table 12–5 describes styles that you can define for the index utility.

Table 12–5 The List View Index Styles

styleClass name	Description
`adfmf-listView-index`	Defines style of the index bar.
`adfmf-listView-indexItem`	Defines style of one item in the index bar.
`adfmf-listView-indexItem-active`	Defines style of the item in the index bar which has link to a related divider.
`adfmf-listView-indexCharacter`	Defines style of a character in the index bar.
`adfmf-listView-indexBullet`	Defines style of a bullet between two characters in index bar.
`adfmf-listView-indexOther`	Defines style of a character that represents all unknown characters in the index bar.

When the List View component with visible dividers functions as a container that provides scrolling and it becomes a subject to scrolling, the dividers are pinned at the top of the view. If this is the case, you have to explicitly set the height of the List View component. In all other cases, when the List View does not perform any scrolling itself but instead uses the scrolling of its parent container (such as the Panel Page), the List View does not have any height constraint set and its height is determined by its child components. This absence of the defined height constraint effectively disables the animated transition and pinning of dividers.

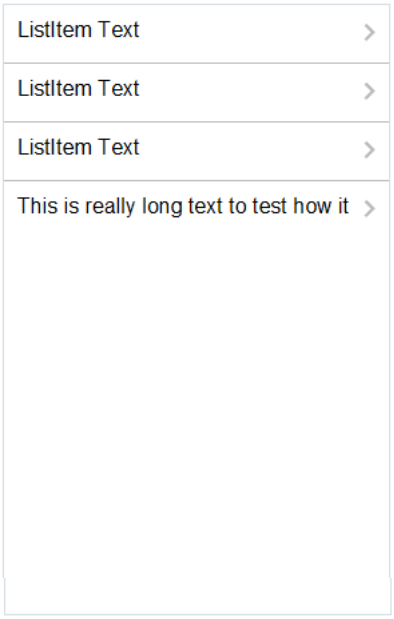
- Inset List

  This example shows the `listView` element defined in a MAF AMX file. The definition corresponds to the inset component.

```
<amx:listView id="listView1"
              styleClass="adfmf-listView-insetList"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem id="listItem1">
      <amx:outputText id="outputText1" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem2">
      <amx:outputText id="outputText3" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem3">
      <amx:outputText id="outputText5" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem4">
      <amx:outputText id="outputText7"
                      value="This is really long text to test how it is
handled"/>
   </amx:listItem>
</amx:listView>
```

  Figure 12–27 demonstrates an inset List View component at design time.

Figure 12–27 Inset List View at Design Time



  This example shows another definition of the `listView` element in a MAF AMX file. This definition also corresponds to the inset component, however, the value of this List View is provided by a collection.

```
<amx:listView id="list1"
              value="#{CarBean.carCollection}"
              var="row"
              styleClass="adfmf-listView-insetList"
              showMoreStrategy="autoScroll"
```

```
                bufferStrategy="viewport"
                fetchSize="10">
      <amx:listItem id="li1" action="go">
         <amx:outputText id="ot1" value="#{row.name}"/>
      </amx:listItem>
   </amx:listView>
```

There is a particular order in which MAF AMX processes the List Item component's child operations and attributes. For more information, see Section 12.3.5.8, "What You May Need to Know About the Order of Processing Operations and Attributes."

Unlike other MAF AMX components, when you drag and drop a List View onto a MAF AMX page and generate the listView with content from a data control, a dialog called the **Component Gallery** appears. This dialog allows you to choose a specific layout for the List View. Figure 12–28 shows the List View Component Gallery.

Figure 12–28 Component Gallery for List View formats

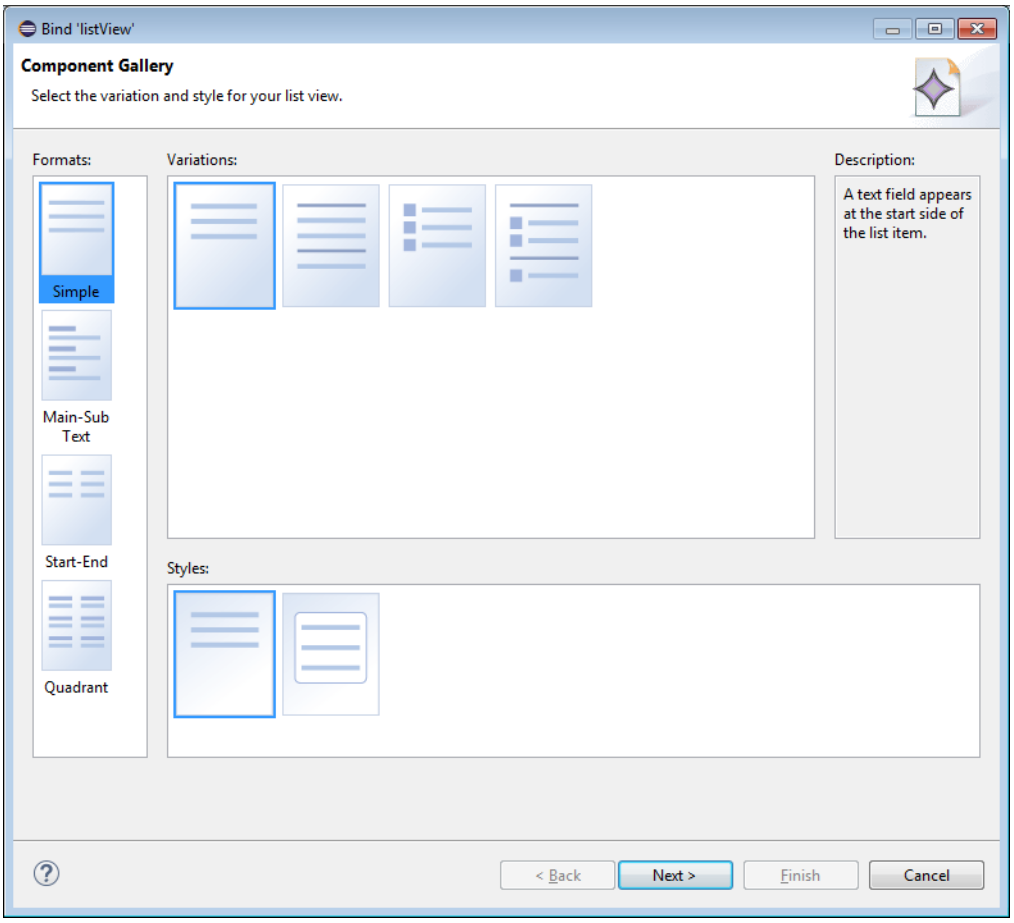


Table 12–6 lists the supported **List Formats** that are displayed in the ListView Component Gallery.

Table 12–6 List Formats

Format	Element Row Values
Simple	▪ Text

Table 12–6 (Cont.) List Formats

Format	Element Row Values
Main-Sub Text	■ Main Text ■ Subordinate Text
Start-End	■ Start Text ■ End Text
Quadrant	■ Upper Start Text ■ Upper End Text ■ Lower Start Text ■ Lower End Text

The **Variations** presented in the ListView Gallery for a selected list format consist of options to add either dividers, a leading image, or both:

- Selecting a variation with a leading image adds an Image row to the List Item Content table.
- Selecting a variation with a divider defaults the Divider Attribute field to the first attribute in its list rather than the default No Divider value, and populates the Divider Mode field with its default value of All.

The **Styles** options presented in the ListView Gallery allow you to suppress chevrons, use an inset style list, or both:

- The selections do not modify any state in the Edit List View dialog. They only affect the generated MAF AMX markup.
- Selecting a style with the inset list sets the `adfmf-listView-insetList` style class on the `listView` element in the generated MAF AMX markup.

The following is an example of the Simple format with the inset list:

```
<amx:listView var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              styleClass="adfmf-listView-insetList"
              id="listView2"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem id="li2">
      <amx:outputText value="#{row.employeename}" id="ot3"/>
   </amx:listItem>
</amx:listView>
```

The ListView Gallery's **Description** pane is updated based on the currently selected Variation. The format includes a brief description of the main style, followed by the details of the selected variation. Four main styles with four variations on each provide sixteen unique descriptions detailed in Table 12–7.

Table 12–7 List View Variations and Styles

List Format	Variation	Description
Simple	Basic	A text field appears at the start side of the list item.
Simple	Dividers	A text field appears at the start side of the list item, with items grouped by dividers.

Table 12–7 (Cont.) List View Variations and Styles

List Format	Variation	Description
Simple	Images	A text field appears at the start side of the list item, following a leading image.
Simple	Dividers and Images	A text field appears at the start side of the list item, following a leading image. The list items are grouped by dividers.
Main-Sub Text	Basic	A prominent main text field appears at the start side of the list item with subordinate text below.
Main-Sub Text	Dividers	A prominent main text field appears at the start side of the list item with subordinate text below. The list items are grouped by dividers.
Main-Sub Text	Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image.
Main-Sub Text	Dividers and Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image. The list items are grouped by dividers.
Start-End	Basic	Text fields appear on each side of the list item.
Start-End	Dividers	Text fields appear on each side of the list item, with the items grouped by dividers.
Start-End	Images	Text fields appear on each side of the list item, following a leading image.
Start-End	Dividers and Images	Text fields appear on each side of the list item, following a leading image. The list items are grouped by dividers.
Quadrant	Basic	Text fields appear in the four corners of the list item.
Quadrant	Dividers	Text fields appear in the four corners of the list item, with items grouped by dividers.
Quadrant	Images	Text fields appear in the four corners of the list item, following a leading image.
Quadrant	Dividers and Images	Text fields appear in the four corners of the list item, following a leading image. The list items are grouped by dividers.

After you have chosen a data binding from the **Palette > Data> Data Controls** pane and dropped it into the editor, OEPE displays the ListView Gallery (see Figure 12–29).

When completing the dialog, consider the following:

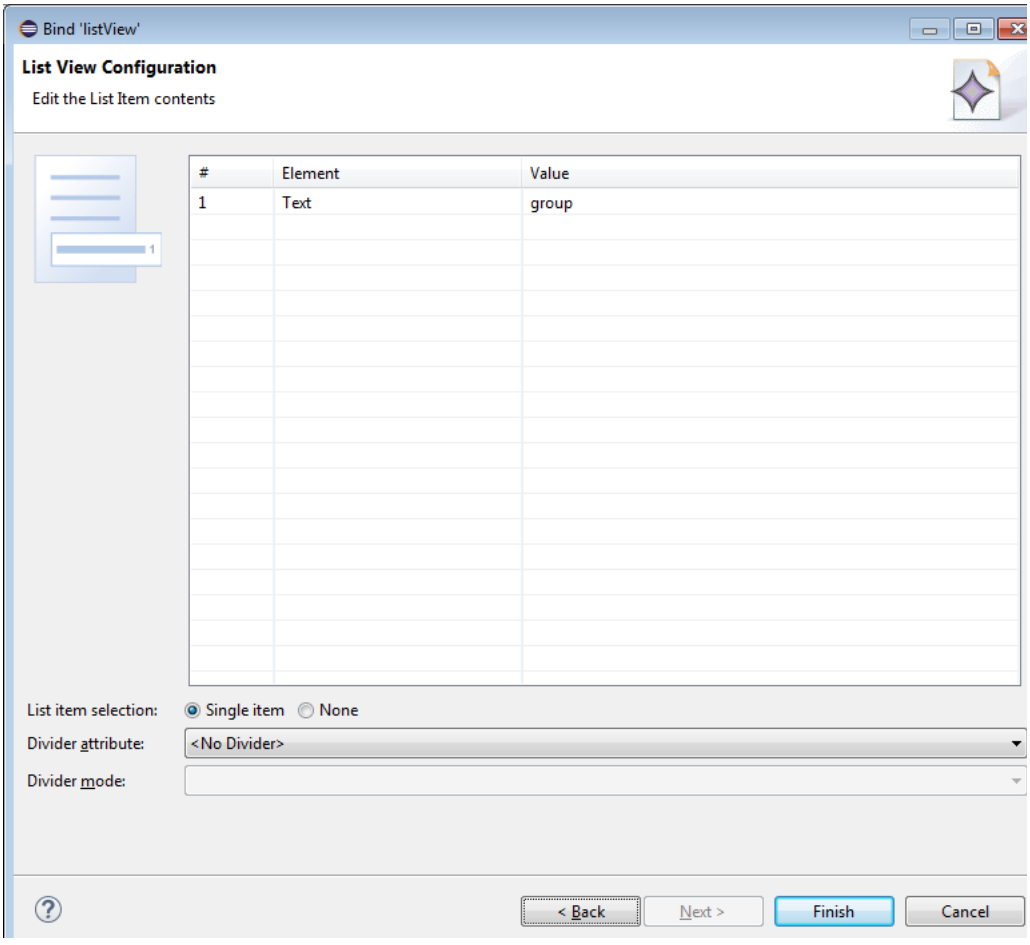
- The image on the left reflects the main content elements from the selected List View format
- The editable cells of the **Value** column are populated with static text strings (see Table 12–8).
- If the **List Item Content** cell contains an Image, the `Value` cell is defaulted to the `<add path to your image>` string. If this is the case, you have to replace it with the path to the image.
- The **List Item Selection** indicates the selection mode.
- Since you cannot set the divider attribute when the List View contains List Item child components, rather than being data bound, both the Divider Attribute and the Divider Mode fields are disabled.

Table 12–8 Static Text Strings for List View

List Format	Element Row Values	Values for the Output Text
Simple	▪ Text	▪ 'ListItem Text'
Main-Sub Text	▪ Main Text ▪ Subordinate Text	▪ 'Main Text' ▪ 'This is the subordinate text.'
Start-End	▪ Start Text ▪ End Text	▪ 'Start Text' ▪ 'End Text'
Quadrant	▪ Upper Start Text ▪ Upper End Text ▪ Lower Start Text ▪ Lower End Text	▪ 'Upper Start Text' ▪ 'Upper End Text' ▪ 'Lower Start Text' ▪ 'Lower End Text'

Figure 12–29 shows the List View Configuration dialog on which you can specify the layout of your list elements.

Figure 12–29 The List View Configuration dialog



The following are special cases to consider when creating a bound List View:

- If a JavaBean method returns a list without generics, you should use the Element Type field to create the List Item content, as Figure 12–29 shows.

- If the list data collection value provided is not collection-based, a Value column replaces the Value Bindings column with blank values.

For more information, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- Section 12.3.15.3, "Configuring The List View Layout"
- Section 11.3.2.3.3, "Dragging and Dropping Collections"
- UIDemo, a MAF sample application available from **File** > **New** > **MAF Examples**. This sample demonstrates how to use various types of the List View component and how to apply styles to adjust the page layout to a specific pattern.

12.3.15.1 Configuring Paging and Dynamic Scrolling

You can configure the List View component to display data in a list that is arbitrarily long. This is done by appending data to the bottom of the list.

The `fetchSize` attribute determines how many rows the List View component should initially load. If there are more rows available than defined by the `fetchSize`, a clickable area is displayed after the last fetched row. Clicking within this area loads another batch of rows that equals the `fetchSize`. Once there are no more rows to display, the clickable area disappears.

The `fetchSize` attribute does not represent the number of loaded rows. Instead, it represents the value by which the number of rows is incremented. When the List View component is created, the `fetchSize` attribute is by default set to use an EL expression that points to the `rangeSize` of the `PageDef` iterator. For information on the `PageDef` file, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts" and Figure 11–40, "PageDef File". Setting the `fetchSize` to the same value as the `rangeSize` improves the application performance.

This example shows the `listView` element that was created from a collection called `testResults` of a data control (see Section 11.3.2.3, "Adding Data Controls to the View").

```
<amx:listView var="row"
              value="#{bindings.testResults.collectionModel}"
              fetchSize="#{bindings.testResults.rangeSize}">
```

In the preceding example, the `fetchSize` attribute points to the `rangeSize` on `bindings.testResults`. The next example shows a line from the `PageDef` file for this MAF AMX page. This `PageDef` file contains an `accessorIterator` element called `testResultsIterator` to which the `testResults` is bound.

```
<accessorIterator MasterBinding="Class1Iterator"
                  Binds="testResults"
                  RangeSize="25"
                  DataControl="Class1"
                  BeanClass="mobile.Test"
                  id="testResultsIterator"/>
```

If the `fetchSize` attribute is set to -1, all records are retrieved.

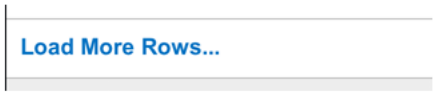
The following two attributes of the List View component enable its scrolling behavior:

- `showMoreStrategy`: defines the List View component's strategy for loading more rows when required.

  When a List View component provides its own scrolling (see "To force a List View to provide its own scrolling:" on page 12-56) as opposed to being a subject to

scrolling by one of its parent containers, and that List View is scrolled to the end, it automatically invokes the showMoreStrategy allowing itself to fetch the next set of records.

At runtime, this attribute expresses itself as a Load More Rows link by default:



For information on how to configure the List View's scrolling and row-displaying behavior by setting values of the `showMoreStrategy` attribute, see *Tag Reference for Oracle Mobile Application Framework*.

- `bufferStrategy`: defines how the user interface for the rows is retained

  When the List View's height is constrained allowing it to provide its own scrolling (see "To force a List View to provide its own scrolling:" on page 12-56) as opposed to being a subject to scrolling by one of its parent containers, it fetches and displays previously hidden rows and their contents. The List View achieves this in one of the following ways:

  – By continuously adding an increasing number of hidden rows to the list.

  – By limiting the number of added rows that are available within the rendering engine and only retaining the rows that are in the List View's visible area (viewport), therefore reducing the amount of memory the MAF application uses.

  You can configure this functionality through the `listView`'s `bufferStrategy` attribute by setting it to either `additive` (if you are not concerned with the memory consumption) or `viewport` (if you are trying to reduce the memory usage). In the latter case, there may be a delay while scrolling before the contents of the new rows become visible. To minimize the number of displayed blank rows, you can set the `listView`'s `bufferSize` attribute. This attribute determines the distance (in pixels) at which the row must be located from the viewport to become hidden.

To force a List View to provide its own scrolling:

- Make the List View the only non-Facet child of a Panel Page.
- Set a fixed height for the List View. For example:

  ```
  inlineStyle="height: 200px;"
  ```

The `rangeChangeListener` attribute (see Section 12.10, "Using Event Listeners") of the List View component allows you to bind a Java handler method for when the Load More Rows link is activated or when the List View is scrolled to the end. This method uses the `oracle.adfmf.amx.event.RangeChangeEvent` object as its parameter and is created when you invoke the **Edit Property: Range Change Listener** dialog from the Properties window.

When you click **OK** on the dialog, this setting is added to the `listView` element in the MAF AMX page:

```
<amx:listView id="listView1" rangeChangeListener="#{pageFlowScope.HRBean.goGet}" >
```

And the Java method that the example below shows is added to a sample `HRBean` class:

```
public void goGet(RangeChangeEvent rangeChangeEvent) {
   // Add event code here...
```

```
}
```

Note: The `rangeChangeListener` is called every time new data is being fetched by the List View. Using the `RangeChangeEvent`, you can define whether or not more data is available (see Section 12.10, "Using Event Listeners").

12.3.15.2 Rearranging List View Items

Items in a List View can be rearranged. This functionality is similar on iOS and Android platforms: both show a Rearrange icon aligned along the right margin for each list item. The Rearrange operation is initiated when the end user touches and drags a list item using the Rearrange affordance as a handle. The operation is completed when the end user lifts their finger from the display screen.

Note: If the end user touches and drags anywhere else on the list item, the list scrolls up or down.

The Rearrange Drag operation "undocks" the list item and allows the end user to move the list item up or down in the list.

For its items to be rearrangeable, the List View must not be static, must be in an edit mode, and must be able to listen to moves.

This example shows the `listView` element defined in a MAF AMX file. This definition presents a list with an Edit and Stop Editing buttons at the top that allow switching between editable and read-only list mode.

```
<amx:panelPage id="pp1">
   <amx:commandButton id="edit"
                      text="Edit"
                      rendered="#{!bindings.inEditMode.inputValue}">
      <amx:setPropertyListener id="spl1"
                               from="true"
                               to="#{bindings.inEditMode.inputValue}"
                               type="action"/>
   </amx:commandButton>
   <amx:commandButton id="stop"
                      text="Stop Editing"
                      rendered="#{bindings.inEditMode.inputValue}">
      <amx:setPropertyListener id="spl2"
                               from="false"
                               to="#{bindings.inEditMode.inputValue}"
                               type="action"/>
   </amx:commandButton>
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row"
                 editMode="#{bindings.inEditMode.inputValue}"
                 moveListener="#{MyBean.Reorder}">
      <amx:listItem id="li1">
         <amx:outputText id= "ot1" value="#{row.description}">
      </amx:listItem>
   </amx:listView>
</amx:panelPage>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.3.15.3 Configuring The List View Layout

The List View component can be laid out as either a set of rows, with each row containing one List Item component (default), or a set of cards, with each card containing one or more List Item components.

To define the layout, you use the List View's `layout` attribute and set it to either `rows` or `cards`. When using the cards layout, consider the following:

- Each List Item component is presented as a card in a group of horizontally arranged cards.
- If all available space is consumed, the next card wraps onto a new line.
- To control horizontal alignment of List Item components (cards) within the List View, set the `halign` attribute of the List View to either `start`, `center`, or `end`.
- To generally customize the appearance of the List View:
  - To override the card size defined by default in the skin, specify a new width using the List Item's `inlineStyle` attribute. For more information, see Section 12.6.1, "How to Use Component Attributes to Define Style"

    Note: You cannot set the value to auto or use percent units.

    Alternatively, you can use skinning to override the width from the `.amx-listView-cards .amx-listItem` selector (see Section 12.6.2, "What You May Need to Know About Skinning").
  - To override spacing around the cards defined by default in the skin, you can specify new `margin-top` and `margin-left` using the List Item's `inlineStyle` attribute (see Section 12.6.1, "How to Use Component Attributes to Define Style"), as well as new `padding-bottom` and `padding-right` using the List View's `contentStyle` attribute.

    Alternatively, you can use skinning to override the margin-top and margin-left from the `.amx-listView-cards .amx-listItem` selector, as well as `padding-bottom` and `padding-right` from the `.amx-listView-cards .amx-listView-content` selector (see Section 12.6.2, "What You May Need to Know About Skinning").

For the rows layout, you can use the `halign` attribute to change the alignment of trivial List Item content. However, the use of this attribute might not have a visual effect.

When the List View component with cards layout is in edit mode, its layout switches to `rows` mode.

To adjust the MAF AMX page layout to a specific pattern, you can combine the use of the various types of List View components and styles that are defined through the `styleClass` attribute (see Section 12.6, "Styling UI Components") that uses a set of predefined styles.

A MAF sample application called UIDemo demonstrates all the optional styles for each component and their associated rendering. The UIDemo application is available from **File** > **New** > **MAF Examples**.

The example below shows the `listView` element and its child elements defined in a MAF AMX file. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-startText` places the Output Text component to the start

(left side) of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-endText` places the Output Text component to the end (right side) of the row and applies a blue font to its text. Whether or not the arrow pointing to the right is visible is configured by the `showLinkIcon` attribute of the `listItem` element. Since the default value of this attribute is `true`, the example does not contain this setting.

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItem1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1s1" width="10px"/>
            <amx:cellFormat id="cf11" width="60%" height="43px">
               <amx:outputText id="outputText11" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px"/>
            <amx:cellFormat id="cf12" halign="end" width="40%">
               <amx:outputText id="outputText12" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–30 shows a List View component with differently styled text added to the start (left side) and end (right side) of each row. Besides the text, rows are equipped with a right-pointing arrow representing a link that expands each list item.

Figure 12–30 List View with Start and End Text at Design Time

The example below shows the `listView` element and its child elements defined in a MAF AMX file. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-startText` places the Output Text component to the start of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-endText` places the Output Text component to the end of the row and applies a blue font to its text. Whether or not the arrow pointing to the right is visible on each particular row is configured by the `showLinkIcon` attribute of the `listItem` element. Since in this example this attribute is set to `false` for every `listItem` element, arrows pointing to the right are not displayed.

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItem1" showLinkIcon="false">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1s1" width="10px"/>
            <amx:cellFormat id="cf11" width="60%" height="43px">
```

```
            <amx:outputText id="outputText11" value="#{row.name}"/>
         </amx:cellFormat>
         <amx:cellFormat id="cf1s2" width="10px"/>
         <amx:cellFormat id="cf12" halign="end" width="40%">
            <amx:outputText id="outputText12" value="#{row.status}"
                            styleClass="adfmf-listItem-highlightText"/>
         </amx:cellFormat>
      </amx:rowLayout>
   </amx:tableLayout>
</amx:listItem>
</amx:listView>
```

Figure 12–31 shows a List View component with differently styled text added to the start and end of each row. The rows do not contain right-pointing arrows representing links.

Figure 12–31 List View with Start and End Text Without Expansion Links at Design Time

The example below shows the `listView` element and its child elements defined in a MAF AMX file. In addition to the text displayed at the start and end of each row, this List View contains subtext placed under the end text. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component to the left side of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component to the right side of the row and applies a smaller grey font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-upperStartText` and applies a smaller grey font to its text.

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItem1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl11">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf11" width="60%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px"/>
            <amx:cellFormat id="cf12" halign="end" width="40%">
               <amx:outputText id="outputTexta2" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rl12">
            <amx:cellFormat id="cf13" columnSpan="3" width="100%" height="12px">
               <amx:outputText id="outputTexta3"
                               value="#{row.desc}"
```

```
                                styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–32 shows a List View component with differently styled text added to the start and end of each row, and with a subtext added below the end text on the left.

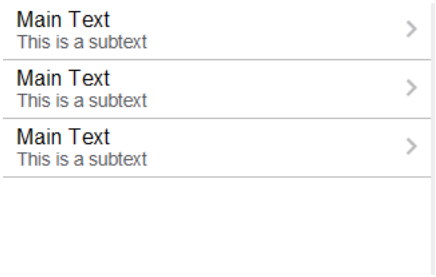
Figure 12–32 List View with Start and End Text and Subtext at Design Time

The example below shows the `listView` element and its child elements defined in a MAF AMX file. This List View is populated with rows containing a main text and subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-mainText` places the Output Text component to the start of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-mainText` and applies a smaller grey font to its text.

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItem1">
      <amx:tableLayout id="tla1" width="100%">
         <amx:rowLayout id="rla1">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa1" width="100%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rla2">
            <amx:cellFormat id="cfa2" width="100%" height="12px" >
               <amx:outputText id="outputTexta2" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–33 shows a List View component with differently styled text added as a main text and subtext to each row.

Figure 12–33 List View with Main Text and Subtext at Design Time

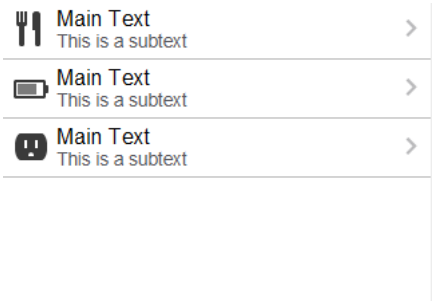


The example below shows the `listView` element and its child elements defined in a MAF AMX file. This List View is populated with cells containing an icon, main text, and subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-mainText` places the Output Text component to the left side of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-mainText` and applies a smaller grey font to its text. The position of the `image` element is defined by its enclosing `cellFormat`.

```
<amx:listView id="lv1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="li1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1" rowSpan="2" width="40px" halign="center">
               <amx:image id="i1" source="#{row.image}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf2" width="100%" height="28px">
               <amx:outputText id="ot1" value="#{row.name}"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rl2">
            <amx:cellFormat id="cf3" width="100%" height="12px">
               <amx:outputText id="ot2" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–34 shows a List View component with icons and differently styled text added as a main text and subtext to each row.

Figure 12–34 List View with Icons, Main Text and Subtext at Design Time

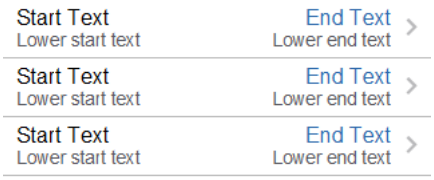


The example below shows the `listView` element and its child elements defined in a MAF AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component at the top left corner of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component at the top right corner of the row and applies a large grey font to its text; setting this attribute to `adfmf-listItem-lowerStartText` places the Output Text component at the bottom left corner of the row and applies a smaller grey font to its text; setting this attribute to `adfmf-listItem-lowerEndText` places the Output Text component at the bottom right corner of the row and applies a smaller grey font to its text. Whether or not the arrow pointing to the right is visible is configured by the `showLinkIcon` attribute of the `listItem` element. Since the default value of this attribute is `true`, the example does not contain this setting.

```
<amx:listView id="lv1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="li1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf2" width="60%" height="28px">
               <amx:outputText id="ot1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf3" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf4" width="40%" halign="end">
               <amx:outputText id="ot2" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rla2">
            <amx:cellFormat id="cf5" width="60%" height="12px">
               <amx:outputText id="ot3" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf6" width="40%" halign="end">
               <amx:outputText id="ot4" value="#{row.priority}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–35 shows a List View component with two types of differently styled text added to the start and end of each row. Besides the text, rows are equipped with a right-pointing arrow representing a link that expands each list item.

Figure 12–35 List View with Four Types of Text at Design Time



The example below shows the `listView` element and its child elements defined in a MAF AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component at the top left corner of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component at the top right corner of the row and applies a large grey font to its text; setting this attribute to `adfmf-listItem-lowerStartTextNoChevron` places the Output Text component at the bottom left corner of the row and applies a smaller grey font to its text; setting this attribute to `adfmf-listItem-lowerEndTextNoChevron` places the Output Text component at the bottom right corner of the row and applies a smaller grey font to its text. Whether or not the arrow pointing to the right is visible on each particular row is configured by the `showLinkIcon` attribute of the `listItem` element. Since in this example this attribute is set to `false` for every `listItem` element, arrows pointing to the right are not displayed.

```
<amx:listView id="lv1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="li1" showLinkIcon="false">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf2" width="60%" height="28px">
               <amx:outputText id="ot1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf3" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf4" width="40%" halign="end">
               <amx:outputText id="ot2" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rl2">
            <amx:cellFormat id="cf5" width="60%" height="12px">
               <amx:outputText id="ot3" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf6" width="40%" halign="end">
               <amx:outputText id="ot4" value="#{row.priority}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
```

```
        </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 12–36 shows a List View component with two types of differently styled text added to the start and end of each row.

Figure 12–36 List View with Four Types of Text and Without Expansion Links at Design Time

12.3.15.4 What You May Need to Know About Using a Static List View

If you create a List View component that is not populated from the model but by hardcoded values, this List View becomes static resulting in some of its properties that you set at design time (for example, `dividerAttribute`, `dividerMode`, `fetchSize`, `moveListener`) ignored at run time.

MAF AMX treats a List View component as static if its `value` attribute is not set. Such lists cannot be editable (that is, you cannot specify its `editMode` attribute).

12.3.16 How to Use the Carousel Component

You use the Carousel (`carousel`) component to display other components, such as images, in a revolving carousel. The end user can change the active item by using either the slider or by dragging another image to the front.

The Carousel component contains a Carousel Item (`carouselItem`) component, whose text represented by the `text` attribute is displayed when it is the active item of the Carousel. Although typically the Carousel Item contains an Image component, other components may be used. For example, you can use a Link as a child that surrounds an image. Instead of creating a Carousel Item component for each object to be displayed and then binding these components to the individual object, you bind the Carousel component to a complete collection. The component then repeatedly renders one Carousel Item component by stamping the value for each item. As each item is stamped, the data for the current item is copied into a property that can be addressed using an EL expression using the Carousel component's `var` attribute. Once the Carousel has completed rendering, this property is removed or reverted back to its previous value. Carousel components contain a Facet named `nodeStamp`, which is both a holder for the Carousel Item used to display the text and short description for each item, and also the parent component to the Image displayed for each item.

The Carousel Item stretches its sole child component. If you place a single Image component inside of the Carousel Item, the Image stretches to fit within the square allocated for the item (as the end user spins the carousel, these dimensions shrink or grow).

> **Tip:** To minimize any negative effect on performance of your application, you should avoid using heavy-weight components as children: a complex structure creates a multiplied effect because several Carousel Items stamps are displayed simultaneously.

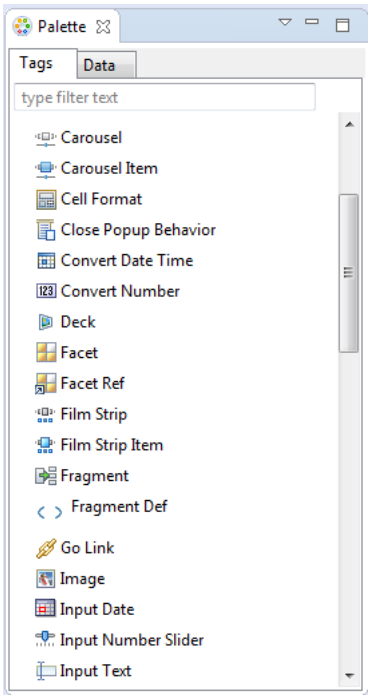
By default, the Carousel displays horizontally. The objects within the horizontal orientation of the Carousel are vertically-aligned to the middle and the Carousel itself is horizontally-aligned to the center of its container. You can configure the Carousel so that it can be displayed vertically, as you might want for a reference rolodex. By default, the objects within the vertical orientation of the Carousel are horizontally-aligned to the center and the Carousel itself is vertically aligned middle. You can change the alignments using the Carousel's `orientation` attribute.

Instead of partially displaying the previous and next images, you can configure your Carousel to display images in a filmstrip or circular design using the `displayItems` attribute.

By default, if the Carousel is configured to display in the circular mode, when the end user hovers over an auxiliary item (that is, an item that is not the current item at the center), the item is outlined to show that it can be selected. Using the `auxiliaryPopOut` attribute, you can configure the Carousel so that instead the item pops out and displays at full size.

In OEPE, the Carousel is located under MAF AMX in the Palette (see Figure 12–37).

Figure 12–37 Carousel in Palette



To create a Carousel component, you must first create the data model that contains images to display, then bind the Carousel to that model and insert a Carousel Item component into the `nodeStamp` facet of the Carousel. Lastly, you insert an Image component (or other components that contain an Image component) as a child to the Carousel Item component.

The example below demonstrates the `carousel` element definition in a MAF AMX file. When defining the `carousel` element, you must place the `carouselItem` element inside of a `carousel` elements's `nodeStamp` facet.

```
<amx:carousel id="carousel1"
              value="#{bindings.products.collectionModel}"
              var="item"
              auxiliaryOffset="0.9"
              auxiliaryPopOut="hover"
              auxiliaryScale="0.8"
              controlArea="full"
              displayItems="circular"
              halign="center"
              valign="middle"
              disabled="false"
              shortDesc="spin"
              orientation="horizontal"
              styleClass="AMXStretchWidth"
              inlineStyle="height:250px;background-color:#EFEFEF;">
   <amx:facet name="nodeStamp">
      <amx:carouselItem id="item1" text="#{item.name}"
                        shortDesc="Product: #{item.name}">
         <amx:commandLink id="link1" action="goto-productDetails"
                          actionListener="#{someMethod()}">
            <amx:image id="image1" styleClass="prod-thumb"
                       source="images/img-big-#{item.uid}.png"/>
            <amx:setPropertyListener id="spl1"
                                     from="#{item}"
                                     to="#{pageFlowScope.product}"
                                     type="action"/>
         </amx:commandLink>
      </amx:carouselItem>
   </amx:facet>
</amx:carousel>
```

The Carousel component uses the `CollectionModel` class to access the data in the underlying collection. You may also use other model classes, such as `java.util.List` or `array`, in which case the Carousel automatically converts the instance into a `CollectionModel` class, but without any additional functionality.

A slider allows the end user to navigate through the Carousel collection. Typically, the thumb on the slider displays the current object number out of the total number of objects. When the total number of objects is too great to calculate, the thumb on the slider shows only the current object number.

For more information and examples, see the following:

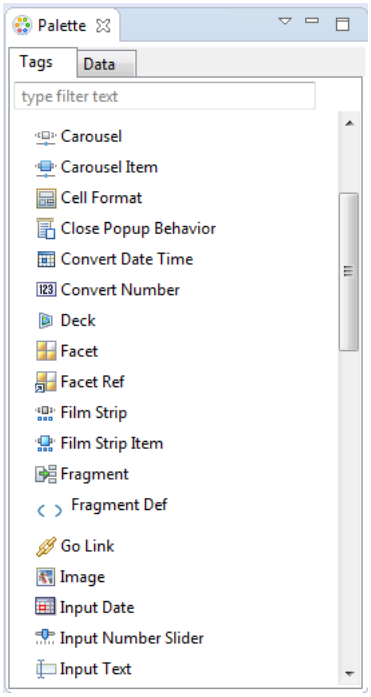
- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**.

12.3.17 How to Use the Film Strip Component

A Film Strip (`filmStrip`) is a container component that visualizes data distributed among a set of groups (pages) in a form of a vertical or horizontal strip. The UI components that represent display items (`filmStripItem`) included in a group must be of the same size and type, and only one group visible at a time. The end user can navigate pages of the Film Strip, select an item and generate actions by tapping it.

In OEPE, the Film Strip is located under MAF AMX in the Palette (see Figure 12–38).

Figure 12–38 Film Strip in Palette



The example below demonstrates the `filmStrip` element definition in a MAF AMX file.

```
<amx:filmStrip id="fs1"
               var="item"
               value="#{bindings.contacts.collectionModel}"
               rendered="#{pageFlowScope.pRendered}"
               styleClass="#{pageFlowScope.pStyleClass}"
               inlineStyle="#{pageFlowScope.pInlineStyle}"
               shortDesc="#{pageFlowScope.pShortDesc}"
               halign="#{pageFlowScope.pFsHalign}"
               valign="#{pageFlowScope.pFsValign}"
               itemsPerPage="#{pageFlowScope.pMaxItems}"
               orientation="#{pageFlowScope.pOrientation}">
   <amx:filmStripItem id="fsi1"
                      inlineStyle="text-align:center; width:200px;">
      <amx:commandLink id="ciLink"
                       action="details"
                       shortDesc="Navigate to details">
         <amx:image id="ciImage" source="images/people/#{item.first}.png"/>
         <amx:setPropertyListener id="spl1"
                                  from="#{item.rowKey}"
                                  to="#{pageFlowScope.currentContact}"
                                  type="action"/>
         <amx:setPropertyListener id="spl2"
                                  from="#{item.first}"
                                  to="#{pageFlowScope.currentContactFirst}"
                                  type="action"/>
         <amx:setPropertyListener id="spl3"
                                  from="#{item.last}"
                                  to="#{pageFlowScope.currentContactLast}"
                                  type="action"/>
      </amx:commandLink>
   </amx:filmStripItem>
```

```
</amx:filmStrip>
```

For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.17.1 What You May Need to Know About the Film Strip Layout

In a vertically laid out Film Strip, the display items are placed in a top-down manner. Depending on the text direction, the page control is located as follows:

- For the left-to-right text direction, the page control is on the right side;
- For the right-to-left text direction, the page control is on the left side.

In a horizontally laid out Film Strip should reflect the text direction mode: in the left-to-ride mode, the first item is located on the left; in the right-to-left mode, the first item is located on the right. In both cases, the page status component is at the bottom.

12.3.17.2 What You May Need to Know About the Film Strip Navigation

The navigation of the Film Strip component is similar to the Deck (see Section 12.2.14, "How to Use a Deck Component") and Panel Splitter component (see Section 12.2.9, "How to Use a Panel Splitter Component"): one page at the time is displayed and the page change is triggered by selection of the page ID or number.

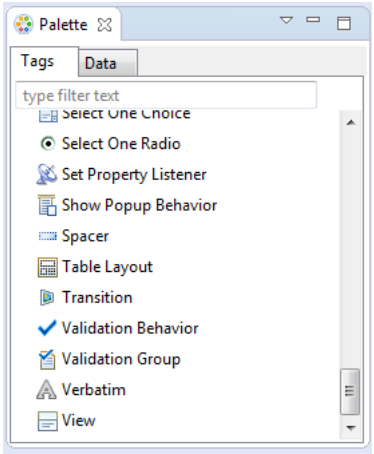
Since the child Film Strip Item component is not meant to be used for navigation to other MAF AMX pages, it does not support the `action` attribute. When required, you can enable navigation through the nested Command Link component.

12.3.18 How to Use Verbatim Component

You use the Verbatim (`verbatim`) operation to insert your own HTML into a page in cases where such a component does not exist or you prefer laying it out yourself using HTML.

In OEPE, Verbatim is located under MAF AMX in the Palette (see Figure 12–39).

Figure 12–39 Verbatim in Palette



For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.18.1 What You May Need to Know About Using JavaScript and AJAX with Verbatim Component

Inserting JavaScript directly into the verbatim content (within the `amx:verbatim` element) is not a recommended practice as it may not execute properly on future versions of the currently supported platforms or on other platforms that MAF might support in the future. Instead, JavaScript and CSS inclusions should be done through the existing `adfmf:include` elements in the `maf-feature.xml` file, which ensures injection of the script into the page at the startup of the MAF AMX application feature.

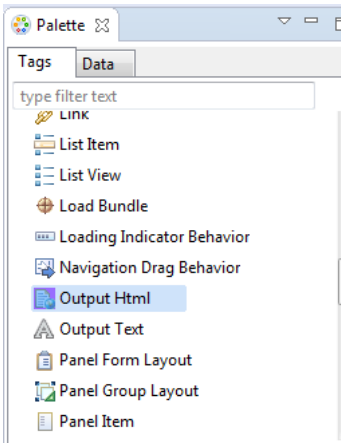
In addition, the use of JavaScript with the Verbatim component is affected by the fact that AJAX calls from an AMX page to a server are not supported. This is due to the security architecture that guarantees that the browser hosting the MAF AMX page does not have access to the security information needed to make connections to a secure server to obtain its resources. Instead, communication with the server must occur from the embedded Java code layer.

12.3.19 How to Use Output HTML Component

The Output HTML (`outputHtml`) component allows you to dynamically and securely obtain HTML content from an EL-bound property or method with the purpose of displaying it on a MAF AMX page. The Output HTML component behaves similarly to the Verbatim component (see Section 12.3.18, "How to Use Verbatim Component") and the same restrictions with regards to JavaScript and AJAX usage apply to it (see Section 12.3.18.1, "What You May Need to Know About Using JavaScript and AJAX with Verbatim Component").

In OEPE, Output HTML is located under **MAF AMX** in the Palette (see Figure 12–40).

Figure 12–40 Output HTML in Components Window



The example below demonstrates the `outputHtml` element definition in a MAF AMX file. The `value` attribute provides an EL binding to a String property that is used to obtain the HTML content to be displayed as the `outputHTML` in the final rendering of the MAF AMX page.

```
<amx:tableLayout id="t1" width="100%">
   <amx:rowLayout id="r1">
```

```
      <amx:cellFormat id="cf1" width="100%">
         <amx:outputHtml id="ReceiptView"
                         value="#{pageFlowScope.purchaseBean.htmlReceiptView}"/>
      </amx:cellFormat>
   </amx:rowLayout>
   <amx:rowLayout id="r2">
      <amx:cellFormat id="cf2" width="100%">
         <amx:outputHtml id="CheckView"
                         value="#{pageFlowScope.purchaseBean.htmlCheckView}"/>
      </amx:cellFormat>
   </amx:rowLayout>
</amx:tableLayout>
```

The example below shows the code from the JavaBean called `MyPurchaseBean` that provides HTML for the Output HTML component shown in the example above.

```
// The property accessor that gets the receipt view HTML
public String getHtmlReceiptView() {
   // Perform some URL remote call to get the HTML to be
   // inserted as a view of the Receipt and return that value
   return obtainReceiptHTMLFromServer();
}
// The property accessor that gets the check view HTML
public String getHtmlCheckView() {
   // Perform some URL remote call to get the HTML to be
   // inserted as a view of the Check and return that value
   return obtainCheckHTMLFromServer();
}
```

Since the Output HTML component obtains its HTML content from a JavaBean property as opposed to downloading it directly from a remote source, consider using the existing MAF security features when coding the retrieval or generation of the dynamically provided HTML. For more information, see Chapter 29, "Securing MAF Applications." In addition, ensure that the HTML being provided comes from a trusted source and is free of threats.

For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.20 How to Enable Iteration

You use the Iterator (`iterator`) operation to stamp an arbitrary number of items with the same kind of data, which allows you to iterate through the data and produce UI for each element.

In OEPE, the Iterator is located under MAF AMX in the Palette.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.3.21 How to Refresh Contents of UI Components

The Refresh Container (`refreshContainer`) component houses contents that can be refreshed by the end user through a pull down gesture resulting in display of a status indicator.

Upon release of a pull down gesture or reaching threshold, the contents' update begins and the status indicator changes until the contents, such as List Item instances, are refreshed.

In other words, the Refresh Container component allows you to expose refresh as a gesture thus eliminating the need of adding a refresh button to a MAF AMX page. Note that Refresh Container should not be placed within a scrollable parent component as it would result in an undesirable scrolling experience for the end user. Instead, you may place a scrollable component, such as a List View, inside of the Refresh Container. When the end user performs a pull down gesture anywhere within the Refresh Container, MAF AMX determines if any UI component between the Refresh Container and the finger of the end user is not scrolled to its top: if any of these components are not scrolled to their tops, this Refresh Container ignores the gesture so the end user can scroll the contents as usual; if all of the components are scrolled to their tops, then the Refresh Container allows the end user to pull down the content to reveal a previously-hidden informational pocket at the top of the Refresh Container; if the finger is lifted prior to reaching a required threshold (a height specified in the skin), the pocket becomes hidden again; when the end user drags the finger down past that threshold, an action event is fired allowing the application to perform operations that would result in data being refreshed. The pocket remains open until the processing completes or, if specified, until a data change event is triggered on the optional `refreshCompleteExpression` attribute of the Refresh Container.

The following example demonstrates the `refreshContainer` element definition in a MAF AMX file. This component refreshes the contents of a List View. The `pullText`, `busyText`, and `subText` attributes define text that appears at various stages of the activated Refresh Container.

In OEPE, the Refresh Container is located under MAF AMX in the Palette.

```
<amx:refreshContainer id="rc1"
       refreshDesc="#{pageFlowScope.componentProperties.refreshDesc}"
       pullText="#{pageFlowScope.componentProperties.pullText}"
       busyText="#{pageFlowScope.componentProperties.busyText}"
       subText="#{pageFlowScope.componentProperties.subText}"
       actionListener="#{PropertyBean.RefreshActionHandler}">
   <amx:setPropertyListener type="action"
       from="Last updated: Recently"
       to="#{pageFlowScope.componentProperties.subText}"/>
   <amx:listView id="listView1"
       var="row"
       value="#{bindings.contacts.collectionModel}"
       bufferStrategy="viewport"
       collapsibleDividers="true"
       dividerAttribute="last"
       dividerMode="firstLetter"
       rendered="#{pageFlowScope.componentProperties.rendered}"
       showDividerCount="true"
       showMoreStrategy="autoScroll">
      <amx:listItem id="listItem1" action="details">
         <amx:outputText id="outputText1"
                         value="#{row.first} #{row.last}"/>
         <amx:setPropertyListener from="#{row.rowKey}"
                                  to="#{pageFlowScope.currentContact}"
                                  type="action"/>
         <amx:setPropertyListener from="#{row.first}"
                                  to="#{pageFlowScope.currentContactFirst}"
                                  type="action"/>
         <amx:setPropertyListener from="#{row.last}"
```

```
                                to="#{pageFlowScope.currentContactLast}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.address}"
                                to="#{pageFlowScope.currentContactAddress}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.city}"
                                to="#{pageFlowScope.currentContactCity}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.state}"
                                to="#{pageFlowScope.currentContactState}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.zip}"
                                to="#{pageFlowScope.currentContactZip}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.phone}"
                                to="#{pageFlowScope.currentContactPhone}"
                                type="action"/>
            <amx:setPropertyListener from="#{row.mobile}"
                                to="#{pageFlowScope.currentContactMobile}"
                                type="action"/>
        </amx:listItem>
    </amx:listView>
</amx:refreshContainer>
```

For more information and examples, see the following:

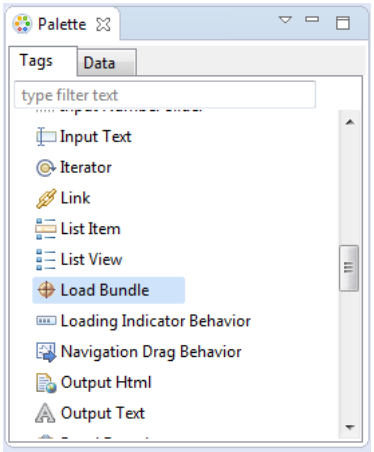
- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.22 How to Load a Resource Bundle

The Load Bundle (`loadBundle`) operation allows you to specify the resource bundle that provides localized text for the MAF AMX UI components on a page. For more information, see Section 12.7, "Localizing UI Components."

In OEPE, the Load Bundle is located under MAF AMX in the Palette (see Figure 12–41).

Figure 12–41 Load Bundle in Palette



In your MAF AMX file, you declare the `loadBundle` element as a child of the `view` element.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

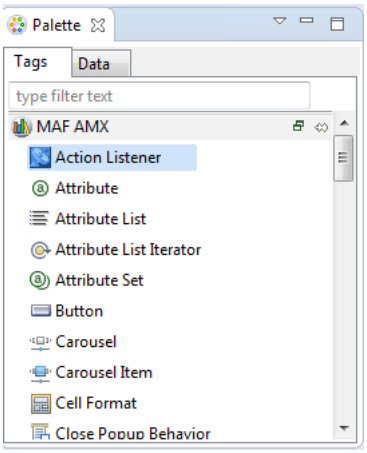
12.3.23 How to Use the Action Listener

The Action Listener (`actionListener`) component allows you to declaratively invoke commands through EL based on the type of the parent component's usage.

The predominant reason for using the Action Listener component is to add gesture-supported actions to its parent components, as well as ability to perform multiple operations for a single gesture, including tap. For more information, see Section 12.3.23.1, "What You May Need to Know About Differences Between the Action Listener Component and Attribute."

In OEPE, the Action Listener component is located under **MAF AMX** in the Palette (see Figure 12–42).

Figure 12–42 Action Listener in Palette



You can add zero or more Action Listener components as children of any command component (Button, Link, List Item, Film Strip Item, and a subset of data visualization components). The `type` attribute of the Action Listener component defines which gesture, such as `swipeLeft`, `swipeRight`, `tapHold`, and so on, causes the `ActionEvent` to fire.

For more information, see the following:

- Section 12.10, "Using Event Listeners"
- Section 12.3.24, "How to Use the Set Property Listener"
- Section 12.4, "Enabling Gestures"
- Section 12.3.23.1, "What You May Need to Know About Differences Between the Action Listener Component and Attribute"
- *Tag Reference for Oracle Mobile Application Framework*

12.3.23.1 What You May Need to Know About Differences Between the Action Listener Component and Attribute

Components such as the Button, Link, and List Item have an `actionListener` attribute, which by inference seems to make the Action Listener component redundant. However, unlike the Action Listener component, these components do not have the `type` attribute that supports gestures, which is the reason MAF provides the

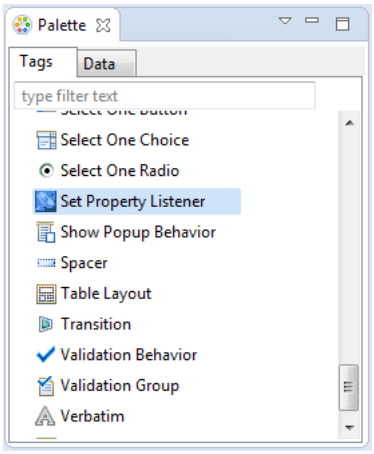
Action Listener component in addition to the `actionListener` attribute of the parent components.

12.3.24 How to Use the Set Property Listener

The Set Property Listener (`setPropertyListener`) component allows you to declaratively copy values from one location (defined by the component's `from` attribute) to another (defined by the component's `to` attribute) as a result of an end-user action on a component, thus freeing you from the need to write Java code to achieve the same result.

In OEPE, the Set Property Listener component is located under **MAF AMX** in the Palette (see Figure 12–43).

Figure 12–43 Set Property Listener in Palette



This example demonstrates the `setPropertyListener` element definition in a MAF AMX file.

```
<amx:listView value="#{bindings.products.collectionModel}" var="row" id="lv1">
   <amx:listItem id="li1" action="details">
      <amx:outputText value="#{row.name}" id="ot1" />
      <amx:setPropertyListener id="spl1"
                               from="#{row}"
                               to="#{pageFlowScope.product}"
                               type="action"/>
   </amx:listItem>
</amx:listView>
```

You can add zero or more Set Property Listener components as children of any command component (Button, Link, List Item, Film Strip Item, and a subset of data visualization components). The `type` attribute of the Set Property Listener component defines which gesture, such as `swipeLeft`, `swipeRight`, `tapHold`, and so on, causes the `ActionEvent` to fire.

For more information, see the following:

- Section 12.10, "Using Event Listeners"
- Section 12.3.23, "How to Use the Action Listener"
- Section 12.4, "Enabling Gestures"
- *Tag Reference for Oracle Mobile Application Framework*

12.3.25 How to Use the Client Listener

The Client Listener (`clientListener`) component allows you to declaratively register a JavaScript listener script that is to be executed when a specific event type is fired.

Before using the Client Listener component, you should check whether or not the MAF AMX page contains any existing behavior components, such as the Navigation Drag Behavior or Show Popup Behavior, because these components might eliminate the need for scripts.

In OEPE, the Set Property Listener component is located under **MAF AMX** in the Palette.

The following example demonstrates the `clientListener` element definition in a MAF AMX file. Both attributes are required and should be specified as follows:

- `method`: defines the client-side JavaScript method name to invoke when triggered by an event of the specified type.
- `type`: defines the type of the client-side component event for which to listen. Note that not all events exist for all components and not all events behave consistently across platforms or versions of the same platform. Examples of events include action if the parent component is a Button; `valueChange` if the parent component is an Input Text. Depending on the parent component, there might be some DOM events that you can use, such as `touchstart`, `touchend`, `tap`, `taphold`, and so on. In addition, some components might have special DOM events, such as the View component's `showpagecomplete`, `mafviewvisible`, `mafviewhidden`, `amxnavigatestart`, and `amxnavigateend` (see Section 13.10.11, "What You May Need to Know About Device Properties" for more information on these events).

  The type attribute supports EL for its initial declaration, but it does not support updates to that EL value—the value associated with the expression must not change unless actions are taken that cause the parent component to rerender.

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:clientListener type="showpagecomplete"
method="handleClientListenerBlue"/>
   <amx:clientListener type="mafviewvisible"
method="handleClientListenerBlue"/>
   <amx:clientListener type="mafviewhidden" method="handleClientListenerBlue"/>
   <amx:clientListener type="amxnavigatestart"
method="handleClientListenerBlue"/>
   <amx:clientListener type="amxnavigateend"
method="handleClientListenerBlue"/>
   <amx:panelPage id="pp1">
      <amx:facet name="header">
         <amx:outputText id="header" value="clientListener"/>
      </amx:facet>
      <amx:facet name="primary">
         <amx:commandButton id="back" action="__back" text="Back"/>
      </amx:facet>
      <amx:facet name="secondary">
         <amx:commandButton id="props" text="Properties" action="properties"/>
      </amx:facet>
      <amx:commandButton id="button1" text="Click Me">
         <amx:clientListener type="#{bindings.pType}"
                             method="#{bindings.pMethod}"/>
      </amx:commandButton>
      <amx:verbatim id="v1"><![CDATA[
```

```
          <script type="text/javascript">
          function handleClientListenerGray(clientEvent) {
             _handleClientListener(clientEvent, "gray");
          }
          function handleClientListenerBlue(clientEvent) {
             _handleClientListener(clientEvent, "blue");
          }
          function handleClientListenerOrange(clientEvent) {
             _handleClientListener(clientEvent, "orange");
          }
          function clearRecentEvents(clientEvent) {
             for (var i=9; i>=0; --i) {
                var row = document.getElementsByClassName("recent"+i)[0];
                row.textContent = "n/a";
                row.style.color = "";
             }
          }
          function _handleClientListener(clientEvent, color) {
             try {
                for (var i=9; i>0; --i) {
                   var currentRow =
document.getElementsByClassName("recent"+i)[0];
                   var olderRow = document.getElementsByClassName
                                                      ("recent"+(i-1))[0];
                   currentRow.textContent = olderRow.textContent;
                   currentRow.style.color = olderRow.style.color;
                }
                document.getElementsByClassName("recent0")[0].
                                                   textContent = clientEvent;
                document.getElementsByClassName("recent0")[0].style.color =
color;
                console.log("Handled clientListener: " + clientEvent,
clientEvent);
             }
             catch (problem) {
                console.log("Error in verbatim code: " +
                                        clientEvent, clientEvent, problem);
                alert("Error in verbatim code: " + clientEvent + "\n\n" +
problem);
             }
          }
          </script>
          <style type="text/css">
          .recentLine {
             white-space: normal;
             word-wrap: break-word;
             font-size: 12px;
             color: gray;
          }
          </style>
          <fieldset style="min-width: 50px;">
             <legend style="color: gray;">Recent Events</legend>
             <div id="recent0" class="recent0 recentLine">n/a</div>
             <div id="recent1" class="recent1 recentLine">n/a</div>
             <div id="recent2" class="recent2 recentLine">n/a</div>
             <div id="recent3" class="recent3 recentLine">n/a</div>
             <div id="recent4" class="recent4 recentLine">n/a</div>
             <div id="recent5" class="recent5 recentLine">n/a</div>
             <div id="recent6" class="recent6 recentLine">n/a</div>
             <div id="recent7" class="recent7 recentLine">n/a</div>
```

```
            <div id="recent8" class="recent8 recentLine">n/a</div>
            <div id="recent9" class="recent9 recentLine">n/a</div>
         </fieldset>
      ]]></amx:verbatim>
   </amx:panelPage>
</amx:view>
```

For more information, see the following:

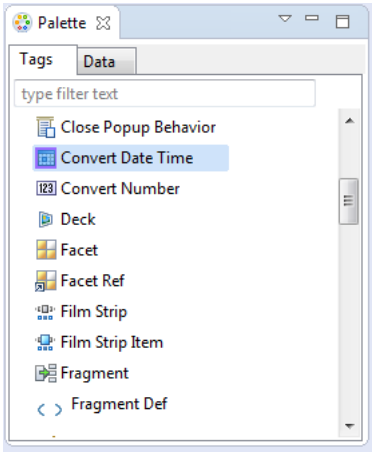
- Section 12.10, "Using Event Listeners"
- *Tag Reference for Oracle Mobile Application Framework*

12.3.26 How to Convert Date and Time Values

The Convert Date Time (`convertDateTime`) is not an independent UI component: it is a converter operation that you use in conjunction with an Output Text and Input Text component to display converted date, time, or a combination of date and time in a variety of formats following the specified pattern.

In OEPE, the Convert Date Time is located under MAF AMX in the Palette (see Figure 12–44).

Figure 12–44 Convert Date Time in Palette



This example demonstrates the `convertDateTime` element declared in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:inputText styleClass="ui-text" value="Order Date" id="it1" >
      <amx:convertDateTime id="cdt1" type="both"/>
   </amx:inputText>
</amx:panelPage>
```

To convert date and time values:

1. From the **Components** window, drag a **Convert Date Time** component and insert it within an Output Text or Input Text component, making it a child element of that component.
2. Open the **Property** editor for the Convert Date Time component and define its attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

Note: The Convert Date Time component does not produce any effect at design time.

The Convert Date Time component allows a level of leniency while converting an input value string to date:

- A converter with associated pattern `MMM` for month, when attached to any value holder, accepts values with month specified in the form `MM` or `M` as valid.
- Allows use of such separators as dash (-) or period (.) or slash (/), irrespective of the separator specified in the associated pattern.
- Leniency in pattern definition set by the `pattern` attribute.

For example, when a pattern on the converter is set to `"MMM/d/yyyy"`, the following inputs are accepted as valid by the converter:

```
Jan/4/2004
Jan-4-2004
Jan.4.2004
01/4/2004
01-4-2004
01.4.2004
1/4/2004
1-4-2004
1.4.2004
```

The converter supports the same parsing and formatting conventions as the `java.text.SimpleDateFormat` (specified using the `dateStyle`, `timeStyle`, and `pattern` attributes), except the case when the time zone is specified to have a long display, such as `timeStyle=full` or a pattern set with `zzzz`. Instead of a long descriptive string, such as "Pacific Standard Time", the time zone is displayed in the General Timezone format (GMT +/- offset) or RFC-822 time zones.

The exact result of the conversion depends on the locale, but typically the following rules apply:

- `SHORT` is completely numeric, such as 12.13.52 or 3:30pm
- `MEDIUM` is longer, such as Jan 12, 1952
- `LONG` is longer, such as January 12, 1952 or 3:30:32pm
- `FULL` is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST

12.3.26.1 What You May Need to Know About Date and Time Patterns

As per `java.text.SimpleDateFormat` definition, date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from `A` to `Z` and from `a` to `z` are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (`'`) to avoid interpretation. `" ' ' "` represents a single quote. All other characters are not interpreted; instead, they are simply copied into the output string during formatting, or matched against the input string during parsing.

Table 12–9 lists the defined pattern letters (all other characters from `A` to `Z` and from `a` to `z` are reserved).

Table 12–9 Date and Time Pattern Letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	■ AD
y	Year	Year	■ 1996 ■ 96
M	Month in year	Month	■ July ■ Jul ■ 07
w	Week in year	Number	■ 27
W	Week in month	Number	■ 2
D	Day in year	Number	■ 189
d	Day in month	Number	■ 10
F	Day of week in month	Number	■ 2
E	Day in week	Text	■ Tuesday ■ Tue
a	Am/pm marker	Text	■ PM
H	Hour in day (0-23)	Number	■ 0
k	Hour in day (1-24)	Number	■ 24
K	Hour in am/pm (0-11)	Number	■ 0
h	Hour in am/pm (1-12)	Number	■ 12
m	Minute in hour	Number	■ 30
s	Second in minute	Number	■ 55
S	Millisecond	Number	■ 978
z	Time zone	General time zone	■ Pacific Standard Time ■ PST ■ GMT-08:00
Z	Time zone	RFC 822 time zone	■ -0800

Pattern letters are usually repeated, as their number determines the exact presentation.

12.3.27 How to Convert Numerical Values

The Convert Number (`convertNumber`) is not an independent UI component: it is a converter operation that you use in conjunction with an Output Text, Input Text, and Input Number Slider components to display converted number or currency figures in a variety of formats following a specified pattern.

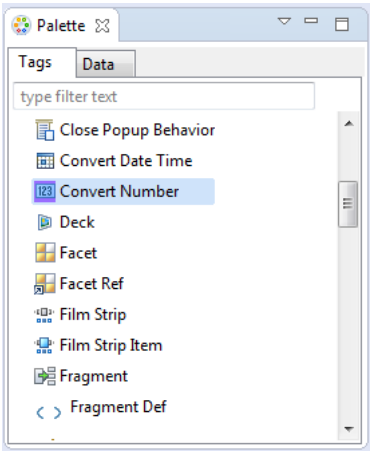
The Convert Number component provides the following types of conversion:

- From value to string, for display purposes.
- From formatted string to value, when formatted input value is parsed into its underlying value.

When the Convert Number is specified as a child of an Input Text component, the numeric keyboard is displayed on a mobile device by default.

In OEPE, the Convert Number is located under MAF AMX in the Palette (see Figure 12–45).

Figure 12–45 Convert Number in Components Window



This example demonstrates the `convertNumber` element defined in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:inputText styleClass="ui-text" value="Product Price" id="it1" >
      <amx:convertNumber id="cn1"
                         type="percent"
                         groupingUsed="false"
                         integerOnly="true"/>
   </amx:inputText>
</amx:panelPage>
```

To convert numerical values:

1. From the **Components** window, drag a **Convert Number** component and insert it within an Output Text, Input Text, or Input Number Slider component, making it a child element of that component.
2. Open the **Property** editor for the Convert Number component and define its attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

> **Note:** The Convert Number component does not produce any effect at design time.

12.3.28 How to Enable Drag Navigation

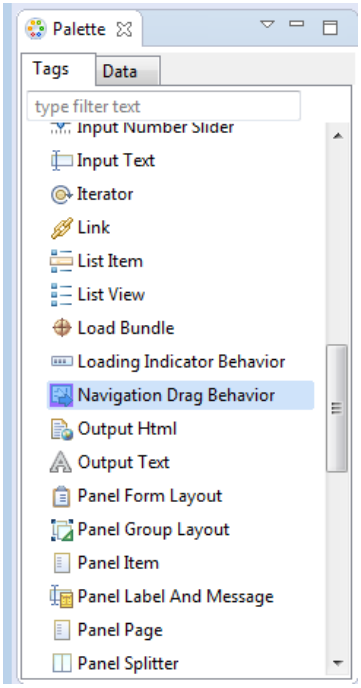
The Navigation Drag Behavior (`navigationDragBehavior`) operation allows you to invoke the action of navigating to the next or previous MAF AMX page by dragging a portion of the mobile device screen in a specified direction (left or right). As the drag in the specified direction occurs, an indicator is displayed on the appropriate side of the screen to hint that an action will be performed if the dragging continues and then stops as soon as the indicator is fully revealed. If the drag is released before the indicator is fully revealed, the indicator slides away and no action is invoked.

> **Note:** This behavior does not occur if another, closer container consumes the drag gesture.

A MAF AMX page cannot contain more than two instances of the `navigationDragBehavior` element: one with its `direction` attribute set to `back` and one set to `forward`.

In OEPE, the Navigation Drag Behavior is located under **MAF AMX** in the Palette (see Figure 12–46).

Figure 12–46 Navigation Drag Behavior in Palette



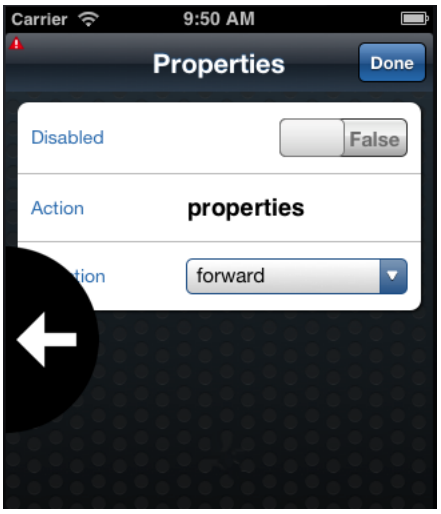
This example demonstrates the `navigationDragBehavior` element defined in a MAF AMX file. Note that this element can only be a child of the `view` element.

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:navigationDragBehavior id="ndb1"
                               direction="forward"
                               action="gotoDetail"/> [1]
   <amx:panelPage id="pp1">
      <amx:facet name="header">
      ...
   </amx:panelPage>
</amx:view>
```

Figure 12–47 shows the Navigation Drag Behavior at runtime (displayed using the `mobileFusionFx` skin).

[1] See Section 12.3.28.1, "What You May Need to Know About the disabled Attribute" for details.

Figure 12–47 Navigation Drag Behavior Operation at Runtime



For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.3.28.1 What You May Need to Know About the disabled Attribute

The value of the `disabled` attribute (see the two examples below) is calculated when one of the following occurs:

- A MAF AMX page is rendered
- A `PropertyChangeListener` updates the component: If you wish to dynamically enable or disable the end user's ability to invoke the Navigation Drag Behavior, you use the `PropertyChangeListener` (similarly to how it is used with other components that require updates from a bean).

The first example below shows a MAF AMX page containing the `navigationDragBehavior` element with a defined disabled attribute. The functionality is driven by the Button (`commandButton`) that, when activated, changes the backing bean `boolean` value (`navDisabled` in the second example below) from which the `disabled` attribute reads its value. The backing bean, in turn, uses the PropertyChangeListener.

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:panelPage id="pp1">
      <amx:facet name="header">
         <amx:outputText value="Header1" id="ot1"/>
      </amx:facet>
      <amx:commandButton id="cb1"
                         text="commandButton1"
                         actionListener="#{pageFlowScope.myBean.doSomething}"/>
   </amx:panelPage>
   <amx:navigationDragBehavior id="ndb1"
                               direction="forward"
                               action="goView2"
                               disabled="#{pageFlowScope.myBean.navDisabled}"/>
</amx:view>
```

The example below shows the backing bean (`myBean` in the example above that provides value for the navigationDragBehavior's `disabled` attribute.

```
public class MyBean {
   boolean navDisabled = true;
   private PropertyChangeSupport propertyChangeSupport =
                                    new PropertyChangeSupport(this);

   public void setNavDisabled(boolean navDisabled) {
      boolean oldNavDisabled = this.navDisabled;
      this.navDisabled = navDisabled;
      propertyChangeSupport.firePropertyChange("navDisabled",
                                               oldNavDisabled,
                                               navDisabled);
   }

   public boolean isNavDisabled() {
      return navDisabled;
   }

   public void doSomething(ActionEvent actionEvent) {
      setNavDisabled(!navDisabled);
   }

   public void addPropertyChangeListener(PropertyChangeListener l) {
      propertyChangeSupport.addPropertyChangeListener(l);
   }

   public void removePropertyChangeListener(PropertyChangeListener l) {
      propertyChangeSupport.removePropertyChangeListener(l);
   }
}
```

12.3.29 How to Use the Loading Indicator

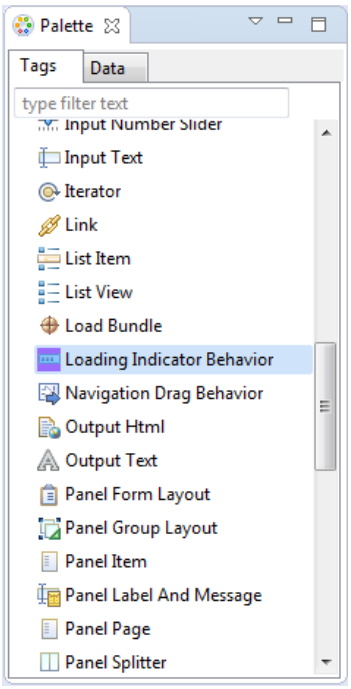
The Loading Indicator Behavior (`loadingIndicatorBehavior`) operation allows you to define the behavior of the loading indicator by overriding the following:

- The duration of the fail-safe timer (in milliseconds).
- An optional JavaScript function name that can be invoked to decide on the course of action when the fail-safe threshold is reached.

For additional information, see the `adf.mf.api.amx.showLoadingIndicator` in .

In OEPE, the Loading Indicator Behavior is located under **MAF AMX** in the Palette (see Figure 12–48).

Figure 12–48 Loading Indicator Behavior in the Palette



The example below demonstrates the `loadingIndicatorBehavior` element defined in a MAF AMX file. Note that this element can only be a child of the `view` element.

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:loadingIndicatorBehavior id="lib1"
                        failSafeDuration="3000"
                        failSafeClientHandler="window.customFailSafeHandler"/>
   <amx:panelPage id="pp1">
      <amx:facet name="header">
      <!-- The loading indicator custom fail safe handler will appear
           if the long running operation runs longer than 3 seconds -->
      <amx:commandButton id="cb1"
                         text="longRunningOperation"
                         actionListener=
                               "#{pageFlowScope.myBean.longRunningOperation} />
   </amx:panelPage>
</amx:view>
```

In the preceding example, the `commandButton` is bound to a long-running method to illustrate that the loading indicator applies to long running operations once the page is loaded (not when the page itself takes a long time to load).

The example below demonstrates the `custom.js` file included with the application feature. It defines the client handler for the `failSafeClientHandler` in the `loadingIndicatorBehavior` page. As per the API requirement, this function returns a `String` of either `hide`, `repeat`, or `freeze`. For more information, see the `adf.mf.api.amx.showLoadingIndicator` in .

```
window.customFailSafeHandler = function() {
   var answer =
      prompt(
         "This is taking a long time.\n\n" +
         "Use \"a\" to hide the indicator.\n" +
```

```
            "Use \"z\" to wait for it.\n" +
            "Otherwise, give it more time.");

   if (answer == "a")
      return "hide"; // don't ask again; hide the indicator
   else if (answer == "z")
      return "freeze" // don't ask again; wait for it to finish
   else
      return "repeat"; // ask again after another duration
};
```

For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application available from **File** > **New** > **MAF Examples**

12.4 Enabling Gestures

You can configure Button, Link, List Item, as well as a number of data visualization components to react to the following gestures:

- Swipe to the right
- Swipe to the left
- Swipe up
- Swipe down
- Tap-and-hold
- Swipe to the start: this gesture is used on the iOS platform to accommodate the right-to-left text direction. This gesture resolves as follows:
  - Swipe to the left for the left-to-right text direction.
  - Swipe to the right for the right-to-left text direction.
- Swipe to the end: this gesture is used on the iOS platform to accommodate the right-to-left text direction. This gesture resolves as follows:
  - Swipe to the right for the left-to-right text direction.
  - Swipe to the left for the right-to-left text direction.

You can define `swipeRight`, `swipeLeft`, `swipeUp`, `swipeDown`, `swipeStart`, `swipeEnd`, and `tapHold` values for the `type` attribute of the following operations:

- Set Property Listener (see Section 12.3.24, "How to Use the Set Property Listener")
- Action Listener (see Section 12.3.23, "How to Use the Action Listener")
- Show Popup Behavior (see Section 12.2.8, "How to Use a Popup Component")
- Close Popup Behavior (see Section 12.2.8, "How to Use a Popup Component")

The values of the `type` attribute are restricted based on the parent component and are supported only for Link (`commandLink`) and List Item (`listItem`) components.

> **Note:** There is no gesture support for the Link Go (`linkGo`) component.

Swiping from start and end is used on the iOS platform to accommodate the right-to-left text direction. It is generally recommended to set the start and end swipe style as opposed to left and right.

The next example demonstrates use of the `tapHold` value of the `type` attribute in a MAF AMX file. In this example, the tap-and-hold gesture triggers the display of a Popup component.

```
<amx:panelPage id="pp1">
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row">
      <amx:listItem id="li1" action="gosomewhere">
         <amx:outputText id="ot1" value="#{row.description}"/>
         <amx:setPropertyListener id="spl1"
                                  from="#{row.rowKey}"
                                  to="#{mybean.currentRow}"
                                  type="tapHold"/>
         <amx:showPopupBehavior id="spb1"
                                type="tapHold"
                                alignid="pp1"
                                popupid="pop1"
                                align="startAfter"/>
      </amx:listItem>
   </amx:listView>>
</amx:panelPage>
<amx:popup id="pop1">
   <amx:panelGroupLayout id="pgl1" layout="horizontal">
      <amx:commandLink id="cm1" actionListener="#{mybean.doX}">
         <amx:image id="i1" source="images/x.png"/>
         <amx:closePopupBehavior id="cpb1" type="action" popupid="pop1"/>
      </amx:commandLink>
      <amx:commandLink id="cm2" actionListener="#{mybean.doY}">
         <amx:image id="i2" source="images/y.png"/>
         <amx:closePopupBehavior id="cpb2" type="action" popupid="pop1"/>
      </amx:commandLink>
      <amx:commandLink id="cm3" actionListener="#{mybean.doZ}">
         <amx:image id="i3" source="images/y.png"/>
         <amx:closePopupBehavior id="cpb3" type="action" popupid="pop1"/>
      </amx:commandLink>
   </amx:panelGroupLayout>
</amx:popup>
```

The example below demonstrates use of the `swipeRight` gesture in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row">
      <amx:listItem id="li1" action="gosomewhere">
         <amx:outputText id="ot1" value="#{row.description}"/>
         <amx:setPropertyListener id="spl1"
                                  from="#{row.rowKey}"
                                  to="#{mybean.currentRow}"
                                  type="swipeRight"/>
         <actionListener id="al1" binding="#{mybean.DoX}" type="swipeRight"/>
      </amx:listItem>
   </amx:listView>>
</amx:panelPage>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

A MAF sample application called GestureDemo demonstrates how to use gestures with a variety of MAF AMX UI components. This sample application is application available from **File** > **New** > **MAF Examples**.

12.5 Providing Data Visualization

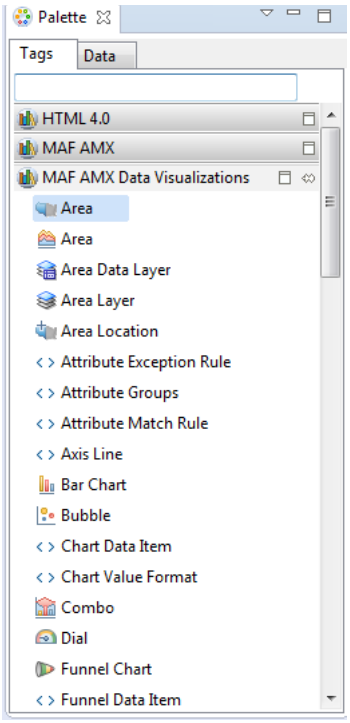
MAF employs a set of data visualization components that you can use to create various charts, gauges, and maps to represent data in your MAF AMX application feature. You can declare the following elements under the `<dvtm>` namespace in a MAF AMX file:

- `areaChart` (see Section 12.5.1, "How to Create an Area Chart")
- `barChart` (see Section 12.5.2, "How to Create a Bar Chart"
- `bubbleChart` (see Section 12.5.4, "How to Create a Bubble Chart")
- `comboChart` (see Section 12.5.5, "How to Create a Combo Chart")
- `lineChart` (see Section 12.5.6, "How to Create a Line Chart")
- `pieChart` (see Section 12.5.7, "How to Create a Pie Chart")
- `scatterChart` (see Section 12.5.8, "How to Create a Scatter Chart")
- `sparkChart` (see Section 12.5.9, "How to Create a Spark Chart")
- `funnelChart` (see Section 12.5.10, "How to Create a Funnel Chart")
- `stockChart` (see Section 12.5.11, "How to Create a Stock Chart")
- `ledGauge` (see Section 12.5.19, "How to Create an LED Gauge")
- `statusMeterGauge` (see Section 12.5.20, "How to Create a Status Meter Gauge")
- `dialGauge` (see Section 12.5.21, "How to Create a Dial Gauge")
- `ratingGauge` (see Section 12.5.22, "How to Create a Rating Gauge")
- `geographicMap` (see Section 12.5.24, "How to Create a Geographic Map Component")
- `thematicMap` (see Section 12.5.25, "How to Create a Thematic Map Component")
- `treemap` (see Section 12.5.27, "How to Create a Treemap Component")
- `sunburst` (see Section 12.5.28, "How to Create a Sunburst Component")
- `timeline` (see Section 12.5.29, "How to Create a Timeline Component")
- `nBox` (see Section 12.5.30, "How to Create an NBox Component")

Chart, gauge, map, and advanced components' elements have a number of attributes that are common to all or most of them. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

In OEPE, data visualization components are located as under **MAF AMX Data Visualizations** in the Palette. Use the scroll bar to view all data visualization components.

Figure 12–49 Data Visualization Components in the Palette



When you drag and drop a data visualization component, a dialog opens, with the content varying depending the information associated with the type of component you are creating:

- **Create Mobile Chart**
- **Create Mobile Gauge**
- **Geographic Map**
- **Create Sunburst** or **Treemap**

> **Note:** After you created the component, you can relaunch the creation dialog by selecting the component in the Source editor or Structure view, and then clicking Edit Component Definition in the Properties window.
>
> You can use the same editing functionality available from the Properties window to edit child components (for example, the Data Point Layer) of some data visualization components.

A MAF sample application called CompGallery demonstrates how to use various data visualization components in your MAF AMX application feature. This sample application is available from **File** > **New** > **MAF Examples**.

For more information on MAF AMX data visualization components, see the following:

- For information on how to add event listeners to data visualization components, see Section 12.10, "Using Event Listeners." Event listeners are applicable to components for the MAF AMX run-time description on both iOS and Android-powered devices, but the listeners do not have any effect at design time.

- For information on databound data visualization components that are created from the Data Controls window, see Section 12.5.32, "How to Create Databound Data Visualization Components."
- For information on providing static data for charts and other data visualization components, see Section 12.5.33, "How to Create Data Visualization Components Based on Static Data"
- For information on chart components' interactivity, see Section 12.5.34, "How to Enable Interactivity in Chart Components"
- For information on creating polar charts, see Section 12.5.35, "How to Create Polar Charts"
- For information on data visualization components' support for accessibility, see Section 12.8, "Understanding MAF Support for Accessibility."

12.5.1 How to Create an Area Chart

You use the Area Chart (`areaChart`) to visually represent data where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties (such as, for example, an area color or pattern). Those properties have to be applied at the series level instead of per each individual data item. You have an option to use the default or custom series styles. For information about defining custom series styles, see Section 12.5.6, "How to Create a Line Chart."

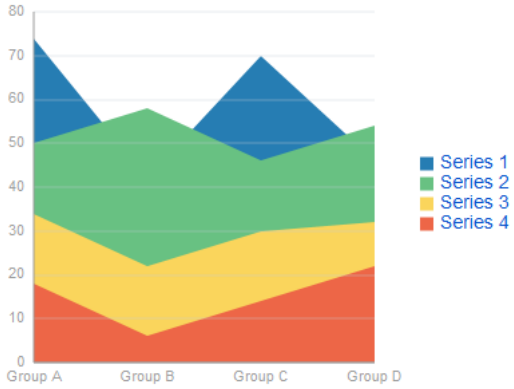
The Area Chart can be zoomed and scrolled along its X Axis. This is enabled through the use of the `zoomAndScroll` attribute.

The Area Chart's orientation (vertical or horizontal) can be controlled by the `orientation` attribute, with the values `vertical` and `horizontal`.

The following example shows the `areaChart` element defined in a MAF AMX file. To create a basic area chart with default series style, you pass it a collection and specify the `dataStamp` facet with a nested `chartDataItem` element.

```
<dvtm:areaChart id="areaChart1"
                value="#{bindings.lineData.collectionModel}"
                var="row"
                inlineStyle="width: 400px; height: 300px;"
                animationOnDisplay="auto"
                animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem
id="areaChartItem1"                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <dvtm:yAxis id="yAxis1"
               axisMaxValue="80.0"
               majorIncrement="20.0"
               title="yAxis Title" />
   <dvtm:legend id="l1" position="end" />
</dvtm:areaChart>
```

Figure 12–50 Area Chart at Design Time



Data items are initialized in the collection model and equipped with the stamping mechanism. At a minimum, each collection `row` must include the following properties:

- `series`: name of the series to which this data item belongs;
- `group`: name of the group to which this data item belongs;
- `value`: the data item value.

The collection `row` might also include other properties, such as `color` or `shape`, applicable to individual data items.

You can use Attribute Groups (`attributeGroups` element) to set style properties for a group of data items based on some grouping criteria, as the example below shows. In this case, the data item `color` and `shape` attributes are set based on the additional grouping expression. The `attributeGroups` can have the following child elements:

- `attributeExceptionRule` from the `dvtm` namespace: replaces an attribute value with another when a particular boolean condition is met.
- `attributeMatchRule` from the `dvtm` namespace: replaces an attribute when the data matches a certain value.
- `attribute` from the `amx` namespace.

```
<dvtm:areaChart id="areaChart1"
                value="#{bindings.lineData.collectionModel}"
                var="row"
                inlineStyle="width: 400px; height: 300px;"
                title="Chart Title"
                animationOnDisplay="auto"
                animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
         <dvtm:attributeGroups id="ag1"
                               type="color"
                               value="#{row.brand}" />
   </amx:facet>
   <dvtm:yAxis id="yAxis1"
               axisMaxValue="80.0"
               majorIncrement="20.0"
               title="yAxis Title" />
   <dvtm:legend id="l1" position="end" />
```

```
</dvtm:areaChart>
```

Note: In the example near the start of this section and Figure 12–48, since custom styles are not set at the series level, series are displayed with the colors based on the default color ramp.

For information on attributes of the `areaChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Area Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-areaChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.2 How to Create a Bar Chart

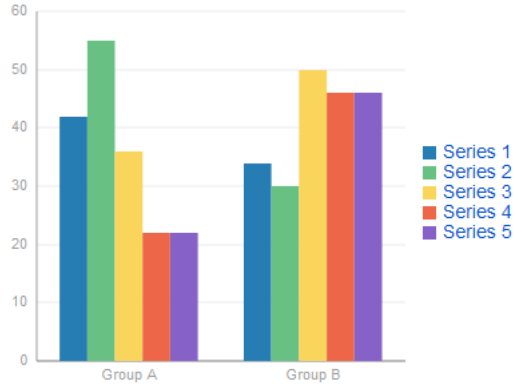
You use a Bar Chart (`barChart`) to visually display data as vertical bars, where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties that you have to apply at the series level instead of per each individual data item.

The Bar Chart can be zoomed and scrolled along its X Axis. This is enabled through the use of the `zoomAndScroll` attribute.

This example shows the `barChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element.

```
<dvtm:barChart id="barChart1"
               value="#{bindings.barData.collectionModel}"
               var="row"
               inlineStyle="width: 400px; height: 300px;"
               animationOnDisplay="zoom"
               animationDuration="3000" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <dvtm:yAxis id="yAxis1"
               axisMaxValue="80.0"
               majorIncrement="20.0"
               title="yAxis Title" />
   <dvtm:legend id="l1" position="start" />
</dvtm:barChart>
```

Figure 12–51 Bar Chart at Design Time



The data model for a bar chart is represented by a collection of items (rows) that describe individual bars. Typically, properties of each bar include the following:

- `series`: name of the series to which this bar belongs;
- `group`: name of the group to which this bar belongs;
- `value`: the data item value (required).

Data must include the same number of groups per series. If any of the series or data pairs are missing, it is passed to the API as `null`.

For information on attributes of the `barChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Bar Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-barChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

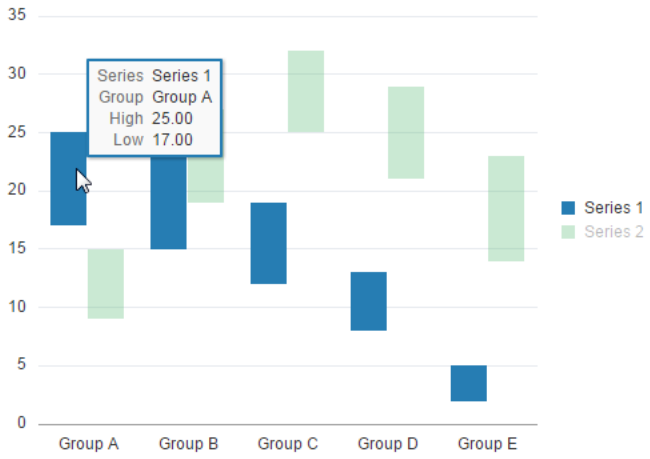
For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.3 How to Create a Range Chart

A range chart allows you to display the low and high values for a data item in a chart.

You can configure an area chart or bar chart to render as a range chart by specifying values for the `low` and `high` attributes that the `<dvtm:chartDataItem>` child component supports. The following example shows how you configure an area chart to render a range chart. Figure 12–52 shows an example of a range chart rendered by the bar chart component.

Figure 12–52 Range Chart Rendering in a Bar Chart



MAF treats the data item as `null` and does not render it on the chart if you only specify a value for one range attribute (`low` or `high`). You must specify values for both the low and high attribute in order to render a range chart. Tool tips and data cursors display the high and low values for the data.

For more information on chart styling, see:

- Section 12.5.2, "How to Create a Bar Chart."
- Section 12.5.1, "How to Create an Area Chart."
- *Tag Reference for Oracle Mobile Application Framework*

12.5.4 How to Create a Bubble Chart

A Bubble Chart (`bubbleChart`) displays a set of data items where each data item has `x`, `y` coordinates and size (bubble). In addition, each data item can have various style attributes, such as `color` and `markerShape`. You can either set properties of each data item individually, or categorize the data items into groups based on various criteria. You may use multiple grouping criteria at the same time, and may also use different style attributes to visualize the relationships of the data items. However, unlike line charts (see Section 12.5.6, "How to Create a Line Chart") or area charts (see Section 12.5.1, "How to Create an Area Chart"), bubble charts do not have a strict notion of the series and groups.

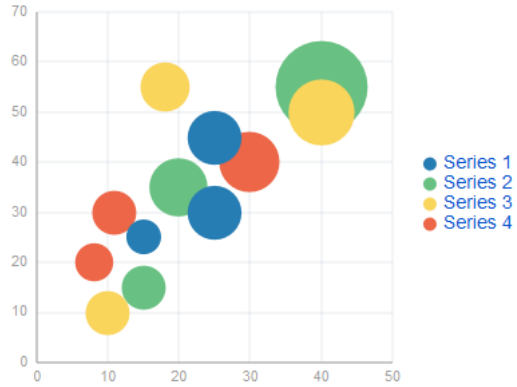
The Bubble Chart can be zoomed and scrolled along its X and Y Axis. This is enabled through the use of the `zoomAndScroll` attribute.

The example below shows the `bubbleChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element. The `color` and `markerShape` attributes of each data item are set individually based on the values supplied in the data model. In addition, the underlying data control must support the respective variable references of `row.label`, `row.size`, and `row.shape`.

```
<dvtm:bubbleChart id="bubbleChart1"
                  value="#{bindings.bubbleData.collectionModel}"
                  inlineStyle="width: 400px; height: 300px;"
                  dataSelection="multiple"
                  rolloverBehavior="dim"
                  animationOnDisplay="auto"
                  var="row">
  <amx:facet name="dataStamp">
```

```
      <dvtm:chartDataItem id="chartDataItem1"
                          group="#{row.group}"
                          x="#{row.x}"
                          y="#{row.y}"
                          markerSize="#{row.size}"
                          color="#{row.color}"
                          markerShape="#{row.shape}" />
   </amx:facet>
</dvtm:bubbleChart>
```

Figure 12–53 Bubble Chart at Design Time



In the next example, the `attributeGroups` element is used to set common style attributes for a related group of data items.

```
<dvtm:bubbleChart id="bubbleChart1"
                  value="#{bindings.bubbleData.collectionModel}"
                  dataSelection="multiple"
                  rolloverBehavior="dim"
                  animationOnDisplay="auto"
                  title="Bubble Chart"
                  var="row">
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          group="#{row.label}"
                          x="#{row.x}"
                          y="#{row.y}" >
         <dvtm:attributeGroups id="ag1" type="color" value="#{row.category}" />
         <dvtm:attributeGroups id="ag2" type="shape" value="#{row.brand}" />
      </dvtm:chartDataItem>
   </amx:facet>
</dvtm:bubbleChart>
```

The data model for a bubble chart is represented by a collection of items (rows) that describe individual data items. Typically, properties of each bar include the following:

- `label`: data item label (optional);
- `x`, `y`: value coordinates (required);
- `z`: the size of data item (required).

The data must include the same number of groups per series. If any of the series or data pairs are missing, it is passed to the API as `null`.

For information on attributes of the `bubbleChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Bubble Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-bubbleChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

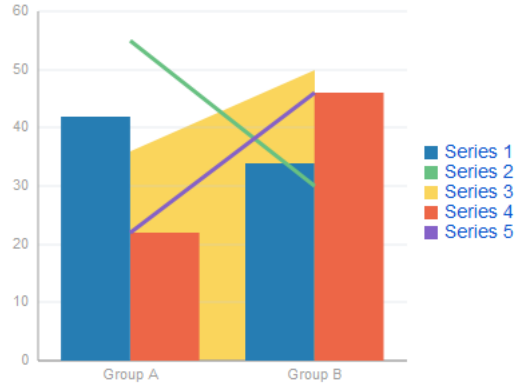
12.5.5 How to Create a Combo Chart

A Combo Chart (`comboChart`) represents an overlay of two or more different charts, such as a line and bar chart.

The example below shows the `comboChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element. The `seriesStamp` facet overrides the default style properties for the series and sets custom series styles using the `seriesStyle` elements.

```
<dvtm:comboChart id="comboChart1"
                 value="#{bindings.barData.collectionModel}"
                 var="row"
                 inlineStyle="width: 400px; height: 300px;"
                 animationOnDisplay="auto"
                 animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle id="seriesStyle1"
                        series="#{row.series}"
                        type="bar"
                        rendered="#{(row.series eq 'Series 1') or
                                    (row.series eq 'Series 2') or
                                    (row.series eq 'Series 3')}" />
      <dvtm:seriesStyle id="seriesStyle2"
                        series="#{row.series}"
                        type="line"
                        lineWidth="5"
                        rendered="#{(row.series eq 'Series 4') or
                                    (row.series eq 'Series 5')}" />
   </amx:facet>
   <dvtm:yAxis id="yAxis1"
               axisMaxValue="80.0"
               majorIncrement="20.0"
               title="yAxis Title" />
   <dvtm:legend position="start" id="l1" />
</dvtm:comboChart>
```

Figure 12–54 Combo Chart at Design Time



For information on attributes of the `comboChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

The Combo Chart's orientation (vertical or horizontal) can be controlled by the `orientation` attribute, with the values `vertical` and `horizontal`.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Combo Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-comboChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.6 How to Create a Line Chart

You use the Line Chart (`lineChart`) to visually represent data where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties (such as, for example, a line color, width, or style). Those properties have to be applied at the series level instead of per each individual data item. You have an option to use the default or custom series styles.

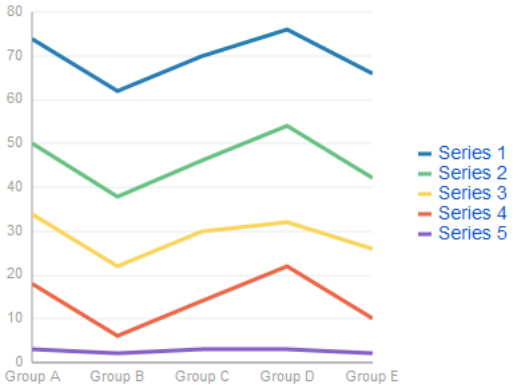
The Line Chart can be zoomed and scrolled along its X Axis. This is enabled through the use of the `zoomAndScroll` attribute.

The following example shows the `lineChart` element defined in a MAF AMX file. To create a basic line chart with default series style, you pass it a collection and specify the `dataStamp` facet with a nested `chartDataItem` element.

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
```

```
                        series="#{row.series}"
                        group="#{row.group}"
                        value="#{row.value}"
                        color="#{row.color}" />
   </amx:facet>
</dvtm:lineChart>
```

Figure 12–55 Line Chart at Design Time



Data items are initialized in the collection model and equipped with the stamping mechanism. At a minimum, each collection `row` must include the following properties:

- `series`: name of the series to which this line belongs;
- `group`: name of the group to which this line belongs;
- `value`: the data item value.

The collection `row` might also include other properties, such as `color` or `shape`, applicable to individual data items.

You can use attribute groups (`attributeGroups` element) to set style properties for a group of data items based on some grouping criteria, as the example below shows. In this case, the data item `color` and `shape` attributes are set based on the additional grouping expression. The `attributeGroups` can have the following child elements:

- `attributeExceptionRule` from the `dvtm` namespace: replaces an attribute value with another when a particular boolean condition is met.
- `attributeMatchRule` from the `dvtm` namespace: replaces an attribute when the data matches a certain value.
- `attribute` from the `amx` namespace.

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
         <dvtm:attributeGroups id="ag1"
```

```
                                type="color"
                                value="#{row.brand}" />
      </dvtm:chartDataItem>
   </amx:facet>
</dvtm:lineChart>
```

Note: In the two examples in this section, since custom styles are not set at the series level, series are displayed with the colors based on the default color ramp.

To override the default style properties for the series, you can define an optional `seriesStamp` facet and set custom series styles using the `seriesStyle` elements, as shown below.

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle series="#{row.series}"
                        lineStyle="#{row.lineStyle}"
                        lineWidth="#{row.lineWidth}" />
   </amx:facet>
</dvtm:lineChart>
```

In the preceding example, the `seriesStyle` elements are grouped based on the value of the `series` attribute. Series with the same name are supposed to share the same set of properties defined by other attributes of the `seriesStyle`, such as `color`, `lineStyle`, `lineWidth`, and so on. When MAF AMX encounters different attribute values for the same series name, it applies the value which was processed last.

Alternatively, you can control the series styles in a MAF AMX charts using the `rendered` attribute of the `seriesStyle` element, as this example shows.

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}"
                          color="#{row.color}" />
   </amx:facet>
   <amx:facet name="seriesStamp">
```

```
      <dvtm:seriesStyle series="#{row.series}"
                        color="red"
                        lineWidth="3"
                        lineStyle="solid"
                        rendered="#{row.series == 'Coke'}" />
      <dvtm:seriesStyle series="#{row.series}"
                        color="blue"
                        lineWidth="2"
                        lineStyle="dotted"
                        rendered="#{row.series == 'Pepsi'}" />
   </amx:facet>
</dvtm:lineChart>
```

The `orientation` attribute allows you to define the Line Chart as either horizontal or vertical.

For information on attributes of the `lineChart` and dvtm child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Line Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-lineChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

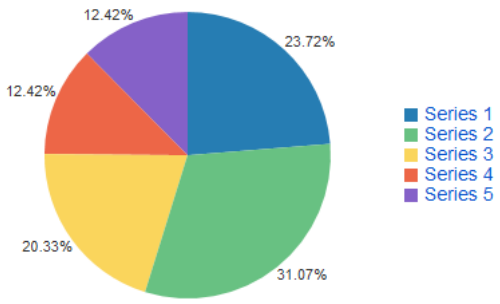
12.5.7 How to Create a Pie Chart

You use a Pie Chart (`pieChart`) to illustrate proportional division of data, with each data item represented by a pie segment (slice). Slices can be sorted by size (from largest to smallest), and small slices can be aggregated into a single “other” slice.

This example shows the `pieChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `pieDataItem` element.

```
<dvtm:pieChart id="pieChart1"
               inlineStyle="width: 400px; height: 300px;"
               value="#{bindings.pieData.collectionModel}"
               var="row"
               animationOnDisplay="zoom"
               animationDuration="3000" >
   <amx:facet name="dataStamp">
      <dvtm:pieDataItem id="pieDataItem1"
                        label="#{row.name}"
                        value="#{row.data}" />
   </amx:facet>
   <dvtm:legend position="bottom" id="l1" />
</dvtm:pieChart>
```

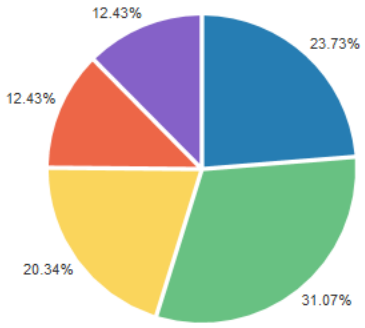
Figure 12–56 Pie Chart at Design Time



You can configure the positioning of the pie slice labels using the `sliceLabelPosition` attribute. By default (`auto`), labels are placed inside of a slice if the slice is big enough to accommodate the label; otherwise the labels are placed outside the slice.

You can also define the explosion (slice separation) effect for a Pie Chart component by setting the `selectionEffect` attribute.

Using the `sliceGaps` attribute, you can create a Pie Chart component that contains gaps between adjacent slices, as the following illustration show. The values of the `sliceGaps` attribute range from 0 (default) for charts with no gaps to 1 for maximum gaps allowed.



The data model for a pie chart is represented by a collection of items that define individual pie data items. Typically, properties of each data item include the following:

- `label`: slice label;
- `value`: slice value.

The model might also define other properties of the data item, such as the following:

- `borderColor`: slice border color;
- `color`: slice color;
- `explode`: slice explosion offset.

For information on attributes of the `pieChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

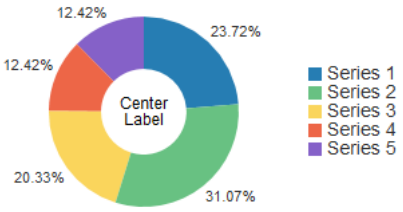
You can define a `facet` child element from the `amx` namespace. The `facet` can have a `pieDataItem` as its child (see Section 12.5.23.4, "Defining Pie Data Item").

12.5.7.1 Configuring the Pie Chart as a Ring Chart

You can create a Pie Chart component with an empty center so it looks like a ring.

The size of the empty space (and, subsequently, the width of the ring) is configured using the `innerRadius` attribute of the `pieChart`. You may also specify text for the center of the ring by setting the `centerLabel` attribute.

Figure 12–57 Ring Chart at Design Time



12.5.7.2 Styling the Pie Chart

You can style the Pie Chart component by overwriting the default CSS settings defined in `dvtm-pieChart` and `dvtm-chartPieLabel`, and `dvtm-chartSliceLabel` classes:

- The top-level element can be styled using

```
.dvtm-pieChart
   - supported properties: all
```

- The pie labels cal be styled using

```
.dvtm-chartPieLabel
   - supported properties:
               font-family, font-size, font-weight, color, font-style
```

- The pie slice labels can be styled using

```
.dvtm-chartSliceLabel
   - supported properties:
               font-family, font-size, font-weight, color, font-style
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For more information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.8 How to Create a Scatter Chart

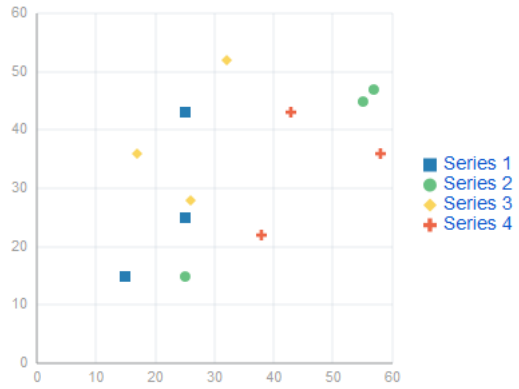
A Scatter Chart (`scatterChart`) displays data as unconnected dots that represent data items, where each item has `x`, `y` coordinates and size. In addition, each data item can have various style attributes, such as `color` and `markerShape`. You can either set properties of each data item individually, or categorize the data items into groups based on various criteria. You may use multiple grouping criteria at the same time, and may also use different style attributes to visualize the data items relationships. However, unlike line charts (see Section 12.5.6, "How to Create a Line Chart") or area charts (see Section 12.5.1, "How to Create an Area Chart"), scatter charts do not have a strict notion of the series and groups.

The Scatter Chart can be zoomed and scrolled along its X and Y Axis. This is enabled through the use of the `zoomAndScroll` attribute.

The example below shows the `scatterChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element. The `color` and `markerShape` attributes of each data item are set individually based on the values supplied in the data model.

```
<dvtm:scatterChart id="scatterChart1"
                   inlineStyle="width: 400px; height: 300px;"
                   animationOnDisplay="zoom"
                   animationDuration="3000"
                   value="#{bindings.scatterData.collectionModel}"
                   var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="chartDataItem1"
                          group="#{row.group}"
                          color="#{row.color}"
                          markerShape="auto"
                          x="#{row.data.x}"
                          y="#{row.data.y}">
         <dvtm:attributeGroups type="color"
                               value="#{row.series}"
                               id="ag1" />
      </dvtm:chartDataItem>
   </amx:facet>
   <dvtm:xAxis id="xAxis1" title="X Axis Title" />
   <dvtm:yAxis id="xAxis2" title="Y Axis Title" />
   <dvtm:legend position="bottom" id="l1" />
</dvtm:scatterChart>
```

Figure 12–58 Scatter Chart at Design Time



The data model for a scatter chart is represented by a collection of items (rows) that describe individual data items. Attributes of each data item are defined by stamping (`dataStamp`) and usually include the following:

- `x`, `y`: value coordinates (required);
- `markerSize`: the size of the marker (optional).

The model might also define other properties of the data item, such as the following:

- `borderColor`: data item border color;
- `color`: data item color;
- `tooltip`: custom tooltip.

For information on attributes of the `scatterChart` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see Section 12.5.23.1, "Defining Chart Data Item").

You can style the Scatter Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-scatterChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.9 How to Create a Spark Chart

A Spark Chart (`sparkChart`) is a simple, condensed chart that displays trends or variations, often in the column of a table. The charts are often used in a dashboard to provide additional context to a data-dense display.

This example shows the `sparkChart` element defined in a MAF AMX file. The `dataStamp` facet is specified with a nested `sparkDataItem` element.

```
<dvtm:sparkChart id="sparkChart1"
                 value="#{bindings.sparkData.collectionModel}"
                 var="row"
                 type="line"
                 inlineStyle="width:400px; height:300px; float:left;">
   <amx:facet name="dataStamp">
      <dvtm:sparkDataItem id="sparkDataItem1" value="#{row.value}" />
   </amx:facet>
</dvtm:sparkChart>
```

Figure 12–59 Spark Chart at Design Time

The data model for a spark chart is represented by a collection of items (rows) that describe individual spark data items. Typically, properties of each data item include the following:

- `value`: spark value.

For information on attributes and dvtm child elements of the sparkChart, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a facet child element from the amx namespace. The facet can have a sparkDataItem as its child (see Section 12.5.23.5, "Defining Spark Data Item").

You can style the Spark Chart component's top-level element by overwriting the default CSS settings defined in the following class:

```
.dvtm-sparkChart
   - supported properties: all
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.10 How to Create a Funnel Chart

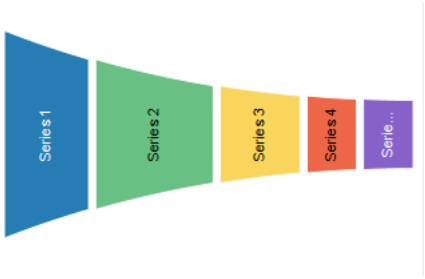
A Funnel Chart (funnelChart) component provides a visual representation of data related to steps in a process. The steps appear as vertical slices across a horizontal cylinder. As the actual value for a given step or slice approaches the quota for that slice, the slice fills. Typically, a Funnel Chart requires actual values and target values against a stage value, which might be time.

This example shows the funnelChart element defined in a MAF AMX file. The dataStamp facet is specified with a nested funnelDataItem element.

```
<dvtm:funnelChart id="funnelChart1"
                  var="row"
                  value="#{bindings.funnelData.collectionModel}"
                  styleClass="dvtm-gallery-component"
                  sliceGaps="on"
                  threeDEffect="#{pageFlowScope.threeD ? 'on' : 'off'}"
                  orientation="#{pageFlowScope.orientation}"
                  dataSelection="#{pageFlowScope.dataSelection}"
                  footnote="#{pageFlowScope.footnote}"
                  footnoteHalign="#{pageFlowScope.footnoteHalign}"
                  hideAndShowBehavior="#{pageFlowScope.hideAndShowBehavior}"
                  rolloverBehavior="#{pageFlowScope.rolloverBehavior}"
                  seriesEffect="#{pageFlowScope.seriesEffect}"
                  subtitle="#{pageFlowScope.titleDisplay ?
                                          pageFlowScope.subtitle : ''}"
                  title="#{pageFlowScope.titleDisplay ? pageFlowScope.title : ''}"
                  titleHalign="#{pageFlowScope.titleHalign}"
                  animationOnDataChange="#{pageFlowScope.animationOnDataChange}"
                  animationDuration="#{pageFlowScope.animationDuration}"
                  animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
                  shortDesc="#{pageFlowScope.shortDesc}">
   <amx:facet name="dataStamp">
      <dvtm:funnelDataItem id="funnelDataItem1"
                           label="#{row.label}"
                           value="#{row.value}"
                           targetValue="#{row.targetValue}"
                           color="#{row.color}"
                           shortDesc="This is a tooltip">
      </dvtm:funnelDataItem>
   </amx:facet>
   <dvtm:legend id="l1"
                position="#{pageFlowScope.legendPosition}"
```

```
                rendered="#{pageFlowScope.legendDisplay}"/>
</dvtm:funnelChart>
```

Figure 12–60 Funnel Chart at Design Time



The data model for a funnel chart is represented by a collection of items (rows) that describe individual funnel data items. Typically, properties of each data item include the following:

- `value`: funnel value
- `label`: funnel slice label

For information on attributes and `dvtm` child elements of the `funnelChart`, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `funnelDataItem` as its child (see Section 12.5.23.6, "Defining Funnel Data Item").

You can style the Funnel Chart component by overwriting the default CSS settings defined in `dvtm-funnelChart` and `dvtm-funnelDataItem` classes:

- The top-level element can be styled using

```
.dvtm-funnelChart
   - supported properties: all
```

- The Funnel Chart data items cal be styled using

```
.dvtm-funnelDataItem
   - supported properties: border-color, background-color
```

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For more information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.11 How to Create a Stock Chart

A Stock Chart (`stockChart`) component displays open, close, minimum, and maximum value for a stock at different points in time during a specific day. The candle bars displaying opening and closing prices for a stock are typically colored green when the price of the stock has risen during the day, and red when the closing price is lower than the opening price.

The following example shows the `stockChart` element defined in a MAF AMX file. The `dataStamp` facet contains a `stockDataItem` element.

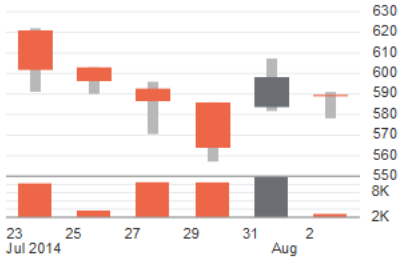
```
<dvtm:stockChart id="stockChart1"
                 dataCursor="#{pageFlowScope.dataCursor}"
                 dataCursorBehavior="#{pageFlowScope.dataCursorBehavior}"
```

```
                  dataSelection="#{pageFlowScope.dataSelection}"
                  emptyText="No data found"
                  footnote=""
                  footnoteHalign="#{pageFlowScope.footnoteHalign}"
                  inlineStyle="width: 100%; height:#{DvtProperties.hostedMode ?
                              '400px' :
deviceScope.hardware.screen.availableHeight-200}px"
                  shortDesc="Stock Chart"
                  styleClass="dvtm-gallery-component"
                  subtitle="#{pageFlowScope.subtitle}"
                  title="#{pageFlowScope.title}"
                  titleHalign="#{pageFlowScope.titleHalign}"
                  value="#{bindings.stockChartData.collectionModel}"
                  var="row"
                  volumeColor="#{pageFlowScope.volumeColor}"
                  zoomAndScroll="#{pageFlowScope.zoomAndScroll}"
                  timeAxisType="mixedFrequency"
                  animationOnDataChange="auto"
                  animationOnDisplay="auto"
                  viewportChangeListener="#{StockChartDataList.ViewportListener}">
   <amx:facet name="dataStamp">
      <dvtm:stockDataItem id="cdi1
                           close="#{row.close}"
                           high="#{row.high}"
                           low="#{row.low}"
                           open="#{row.open}"
                           volume="#{row.volume}"
                           x="#{row.x}"
                           series="BTC"
                           shortDesc="Stock Data Item">
      </dvtm:stockDataItem>
   </amx:facet>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle series="BTC"
                        type="#{pageFlowScope.seriesType}"
                        id="ss1">
      </dvtm:seriesStyle>
   </amx:facet>
   <amx:facet name="overview">
      <dvtm:overview id="ovw" rendered="#{pageFlowScope.overview}">
      </dvtm:overview>
   </amx:facet>
   <dvtm:xAxis id="xAxis"
               viewportMinValue="#{pageFlowScope.viewportMinValue}"
               viewportMaxValue="#{pageFlowScope.viewportMaxValue}">
   </dvtm:xAxis>
   <dvtm:y2Axis id="y2Axis">
      <dvtm:tickLabel id="y2TickLabel"
                      rendered="#{pageFlowScope.showY2}"
                      scaling="none">
         <amx:convertNumber id="cn5"
                            type="number"
                            minFractionDigits="1"
                            maxFractionDigits="1"/>
      </dvtm:tickLabel>
   </dvtm:y2Axis>
   <dvtm:chartValueFormat id="cvf2label"
                          type="close">
      <amx:convertNumber id="closeConvertNumber"
                         type="currency"
```

```
                        minFractionDigits="1"
                        maxFractionDigits="1"
                        currencySymbol="$"/>
</dvtm:chartValueFormat>
<dvtm:chartValueFormat id="cvf2label1"
                        type="high"
                        scaling="none">
   <amx:convertNumber id="highConvertNumber"
                        type="currency"
                        minFractionDigits="1"
                        maxFractionDigits="1"
                        currencySymbol="$"/>
</dvtm:chartValueFormat>
<dvtm:chartValueFormat id="cvf2label2"
                        type="low"
                        scaling="none">
   <amx:convertNumber id="lowConvertNumber"
                        type="currency"
                        minFractionDigits="1"
                        maxFractionDigits="1"
                        currencySymbol="$"/>
</dvtm:chartValueFormat>
<dvtm:chartValueFormat id="cvf2label3"
                        type="open"
                        scaling="none">
   <amx:convertNumber id="openConvertNumber"
                        type="currency"
                        minFractionDigits="1"
                        maxFractionDigits="1"
                        currencySymbol="$"/>
</dvtm:chartValueFormat>
<dvtm:chartValueFormat id="cvf2label4"
                        type="volume"
                        scaling="none">
   <amx:convertNumber id="cn6"
                        type="number"
                        minFractionDigits="1"
                        maxFractionDigits="1"/>
</dvtm:chartValueFormat>
<dvtm:yAxis id="yAxis">
   <dvtm:tickLabel id="tc1" scaling="none">
         <amx:convertNumber id="yAxisConvertNumber"
                              type="currency"
                              minFractionDigits="1"
                              maxFractionDigits="1"
                              currencySymbol="$"/>
   </dvtm:tickLabel>
   <dvtm:referenceLine id="rl2"
                        color="rgb(255,128,0)"
                        lineWidth="1"
                        lineStyle="solid"
                        location="front"
                        lineType="straight"
                        text="Technical analysis"
                        shortDesc="Technical Analysis"
                        displayInLegend="off"
                        rendered="#{pageFlowScope.technicalAnalysis}">
      <amx:iterator var="ref"
                    value="#{bindings.stockReferenceData2.collectionModel}"
                    id="i2">
```

```
            <dvtm:referenceLineItem value="#{ref.value}" x="#{ref.x}" id="rli2"/>
         </amx:iterator>
      </dvtm:referenceLine>
      <dvtm:referenceLine id="rl1"
                          color="#008000"
                          lineWidth="1"
                          lineStyle="solid"
                          location="front"
                          lineType="straight"
                          text=""
                          shortDesc="Total Transaction Fees"
                          displayInLegend="off"
                          rendered="#{pageFlowScope.transactionFees}">
         <amx:iterator var="ref"
                       value="#{bindings.stockReferenceData.collectionModel}"
                       id="i1">
            <dvtm:referenceLineItem value="#{ref.value}" x="#{ref.x}" id="rli1"/>
         </amx:iterator>
      </dvtm:referenceLine>
   </dvtm:yAxis>
   <dvtm:chartValueFormat id="cvf1"
                          type="y"
                          scaling="none"/>
</dvtm:stockChart>
```

Figure 12–61 Stock Chart at Design Time



The data model for a stock chart is represented by a collection of items (rows) that describe individual stock data items.

For information on attributes and `dvtm` child elements of the `stockChart`, see *Tag Reference for Oracle Mobile Application Framework*.

You can define a `facet` child element from the `amx` namespace. The facet can have a `stockDataItem` as its child (see Section 12.5.23.7, "Defining Stock Data Item").

You can style the Stock Chart component by overwriting the default CSS settings defined in the the following classes:

- `dvtm-stockChart-rising`
- `dvtm-stockChart-falling`
- `dvtm-stockChart-range`

For more information on chart styling, see Section 12.5.12, "How to Style Chart Components."

For more information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components."

12.5.12 How to Style Chart Components

With the exception of the Spark Chart, you can style chart components by overwriting the default CSS settings defined in the following classes:

- A chart component's legend can be styled using

```
.dvtm-legend
   - supported properties used for text styling:
               font-family, font-size, font-weight, color, font-style
   - supported properties used for background styling: background-color
   - supported properties used for border styling:
               border-color (used when border width > 0)

.dvtm-legendTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-legendSectionTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style
```

- A chart component's title, subtitle, and so on, can be styled using

```
.dvtm-chartTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartSubtitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartFootnote
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartTitleSeparator
   - supported properties:
               visibility (is title separator rendered),
               border-top-color, border-bottom-color
```

- A chart component's axes can be styled using

```
.dvtm-chartXAxisTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartYAxisTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartY2AxisTitle
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartXAxisTickLabel
   - supported properties:
               font-family, font-size, font-weight, color, font-style

.dvtm-chartYAxisTickLabel
   - supported properties:
```

```
                font-family, font-size, font-weight, color, font-style

.dvtm-chartY2AxisTickLabel
   - supported properties:
                font-family, font-size, font-weight, color, font-style
```

In addition to styling the chart component's top-level element, you can style specific child elements of some charts.

12.5.13 How to Use Events with Chart Components

You can use the `ViewportChangeEvent` to handle zooming and scrolling of chart components. When either zooming or scrolling occurs, the component fires an event loaded with information that defines the new viewport.

You can specify the `viewportChangeListener` as an attribute of Area Chart, Bar Chart, Horizontal Bar Chart, Combo Chart, and Line Chart components.

You can use the `DrillEvent` to handle drilling of chart components. When drilling occurs, the component fires this event.

You can specify the `drillListener` as an attribute of any chart component. In addition, you can use the `drilling` attribute of the `chartDataItem`, `funnelDataItem`, `pieDataItem`, and `seriesStyle` to provide a fine-grained drilling control.

For more information, see the following:

- Section 12.10, "Using Event Listeners"
- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*

12.5.14 What You May Need to Know About Customization of Chart Tooltips

The Chart Value Format (`chartValueFormat`) child component of MAF AMX charts allows you to customize a chart component's tooltip by specifying labels and disabling the display of values within the tooltip, as the following example shows.

```
<dvtm:barChart id="bc1" var="row" value="bindings.Data.collectionModel">
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem id="cdi1"
                          series="row.series"
                          group="row.group"
                          value="row.value"/>
   </amx:facet>
   <dvtm:chartValueFormat id="cvf1" type="value" tooltipLabel="Revenue">
      <amx:convertNumber ... />
   </dvtm:chartValueFormat>
   <dvtm:chartValueFormat id="cvf2" type="series" tooltipLabel="Region"/>
   <dvtm:chartValueFormat id="cvf3" type="groups" tooltipLabel="Product Type"/>
</dvtm:barChart>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.15 How to Enable Sorting of Charts with Categorical Axis

You can use the `sorting` attribute of the Bar Chart, Line Chart, Area Chart, and Combo Chart components to enable sorting of chart categories by their values. For example, countries represented by bars in a Bar Chart can be sorted by their GDP and displayed in either ascending or descending order. By default, sorting is disabled.

12.5.16 How to Define the Initial Zooming of Charts

You can use the `initialZooming` attribute of the Bar Chart, Line Chart, Area Chart, and Combo Chart components to specify whether the chart should initially display the first or the last data points while the chart's zoom level is automatically set to be usable at the current chart size. By default, the initial zooming is disabled.

12.5.17 How to Define Stacking of Specific Chart Series

Bar Chart, Horizontal Bar Chart, Line Chart, Area Chart, and Combo Chart components support a `stack` attribute that allows the data series to be rendered stacked. If this attribute is used by its own, series stacking only allows for stacking to be applied to all of the data series in a chart or none. To enable stacking of some series within the chart and not others, you can use the `stackCategory` attribute of the Series Style (`seriesStyle`) child component in conjunction with the stack attribute of the parent chart: when the chart's stack attribute is set to `on`, you specify the `stackCategory` attribute of the Series Style to define how specific series within the chart are to be stacked.

12.5.18 How to Enable Split Dual-Y Axis in Charts

You can use the `splitDualY` attribute of the Bar Chart, Line Chart, Area Chart, and Combo Chart components to allow charts that use Y2 axis to render two data sets separately in stacked plot areas that share the same X axis. By default, this functionality is disabled.

12.5.19 How to Create an LED Gauge

Unlike charts, gauges focus on a single data point and examine that point relative to minimum, maximum, and threshold indicators to identify problem areas. A LED (lighted electronic display) gauge (`ledGauge`) graphically depicts a measurement, such as key performance indicator (KPI). There are several styles of LED gauges. The ones with arrows are used to indicate good (up arrow), fair (left- or right-pointing arrow), or poor (down arrow). You can specify any number of thresholds for a gauge. However, some LED gauges (such as those with arrow or triangle indicators) support a limited number of thresholds because there is a limited number of meaningful directions for them to point. For arrow or triangle indicators, the threshold limit is three.

This example shows the `ledGauge` element defined in a MAF AMX file.

```
<dvtm:ledGauge id="ledGauge1"
               value="65"
               type="circle"
               inlineStyle="width: 100px; height: 80px; float: left;
                            border-color: navy; background-color: lightyellow;">
   <dvtm:threshold id="threshold1" text="Low" maxValue="40" />
   <dvtm:threshold id="threshold2" text="Medium" maxValue="60" />
   <dvtm:threshold id="threshold3" text="High" maxValue="80" />
</dvtm:ledGauge>
```

Figure 12–62 LED Gauge at Design Time

The data model for a LED gauge is represented by a single metric value which is specified by the `value` attribute.

For information on attributes of the `ledGauge` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define the following `amx` child elements:

- `showPopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `closePopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `validationBehavior` (see Section 12.9, "Validating Input")

12.5.20 How to Create a Status Meter Gauge

A Status Meter Gauge (`statusMeterGauge`) indicates the progress of a task or the level of some measurement along a horizontal rectangular bar or a circle. One part of the component shows the current level of a measurement against the ranges marked on another part. In addition, thresholds can be displayed behind the indicator whose size can be changed.

MAF AMX data visualization provides support for the reference line (`referenceLine`) on its status meter gauge component. You can use this line to produce a bullet graph.

This example shows the `statusMeterGauge` element defined in a MAF AMX file.

```
<dvtm:statusMeterGauge id="meterGauge1"
                       value="65"
                       animationOnDisplay="auto"
                       animationDuration="1000"
                       inlineStyle="width: 300px;
                                    height: 30px;
                                    float: left;
                                    border-color: black;
                                    background-color: lightyellow;"
                       minValue="0"
                       maxValue="100">
   <dvtm:metricLabel/>
   <dvtm:threshold id="threshold1" text="Low" maxValue="40" />
   <dvtm:threshold id="threshold2" text="Medium" maxValue="60" />
   <dvtm:threshold id="threshold3" text="High" maxValue="80" />
</dvtm:statusMeterGauge>
```

Figure 12–63 Rectangular Status Meter Gauge at Design Time

To create a Status Meter Gauge represented by a circle (see Figure 12–64), you set its `orientation` attribute to `circular`. By default, this attribute is set to `horizontal` resulting in a horizontal rectangle.

To create a Status Meter Gauge represented by a circle (see Figure 12–64), you set its orientation attribute to circular.

Figure 12–64 Circular Status Meter Gauge at Design Time

The data model for a status meter gauge is a single metric value which is specified by the `value` attribute. In addition, the minimum and maximum values can also be specified by the `minValue` and `maxValue` attributes.

For information on attributes of the `statusMeterGauge` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define the following `amx` child elements:

- `showPopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `closePopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `validationBehavior` (see Section 12.9, "Validating Input")

12.5.21 How to Create a Dial Gauge

A Dial Gauge (`dialGauge`) specifies ranges of values (thresholds) that vary from poor to excellent. The gauge indicator specifies the current value of the metric while the graphic allows for evaluation of the status of that value.

This example shows the `dialGauge` element defined in a MAF AMX file.

```
<dvtm:dialGauge id="dialGauge1"
                background="#{pageFlowScope.background}"
                indicator="#{pageFlowScope.indicator}"
                value="#{pageFlowScope.value}"
                minValue="#{pageFlowScope.minValue}"
                maxValue="#{pageFlowScope.maxValue}"
                animationDuration="1000"
                animationOnDataChange="auto"
                animationOnDisplay="auto"
                shortDesc="#{pageFlowScope.shortDesc}"
                inlineStyle="#{pageFlowScope.inlineStyle}"
                styleClass="#{pageFlowScope.styleClass}"
                readOnly="true">
</dvtm:dialGauge>
```

Figure 12–65 Dial Gauge at Design Time

The data model for a dial gauge is a single metric value which is specified by the `value` attribute. In addition, the minimum and maximum values can be specified by the `minValue` and `maxValue` attributes.

For information on attributes of the `dialGauge` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

The example below shows the definition of `dialGauge` element with the dark background theme and custom tick labels setting a range from -5000 to 5000.

```
<dvtm:dialGauge id="dialGauge1"
                background="circleDark"
                indicator="needleDark"
                value="#{pageFlowScope.value}"
                minValue="-5000"
                maxValue="5000"
                readOnly="false">
   <dvtm:metricLabel id="metricLabel1"
                     scaling="thousand"
                     labelStyle="font-family: Arial, Helvetica;
                                 font-size: 20; color: white;"/>
   <dvtm:tickLabel id="tickLabel1"
                   scaling="thousand"
                   labelStyle="font-family: Arial, Helvetica;
                               font-size: 18; color: white;"/>
</dvtm:dialGauge>
```

Figure 12–66 Dial Gauge with Metric and Tick Labels at Design Time

You can define the following `amx` child elements for the `dialGauge`:

- `showPopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `closePopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `validationBehavior` (see Section 12.9, "Validating Input")

12.5.22 How to Create a Rating Gauge

A Rating Gauge (`ratingGauge`) provides means to view and modify ratings on a predefined visual scale. By default, a rating unit is represented by a star. You can configure it as a circle, human, rectangle, star, triangle, or diamond by setting the shape attribute of the ratingGauge. You can also configure it to render vertically or horizontally by setting a value for its `orientation` property. By default, it renders horizontally.

This example shows the `ratingGauge` element defined in a MAF AMX file.

```
<dvtm:ratingGauge id="ratingGauge1"
                  value="#{pageFlowScope.value}"
                  minValue="0"
                  maxValue="5"
                  inputIncrement="full"
                  shortDesc="#{pageFlowScope.shortDesc}"
                  inlineStyle="#{pageFlowScope.inlineStyle}"
                  readOnly="true"
                  shape="circle"
                  unselectedShape="circle">
</dvtm:ratingGauge>
```

Figure 12–67 Rating Gauge at Design Time

The data model for a rating gauge is a single metric value which is specified by the `value` attribute. In addition, the minimum and maximum values can be specified by the `minValue` and `maxValue` attributes.

For information on attributes of the `ratingGauge` and `dvtm` child elements that you can define for this component, see *Tag Reference for Oracle Mobile Application Framework*.

You can define the following `amx` child elements for the `ratingGauge`:

- `showPopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `closePopupBehavior` (see Section 12.2.8, "How to Use a Popup Component")
- `validationBehavior` (see Section 12.9, "Validating Input")

12.5.22.1 Overwriting the shortDesc Attribute

You can overwrite the value of the `ratingGauge`'s `shortDesc` attribute by setting the `shortDesc` attribute of the `threshold` child element. If provided, the `threshold`'s `shortDesc` replaces its parent's `shortDesc` every time the `ratingGauge`'s value attribute falls within the specified threshold.

The following example shows how to overwrite the `shortDesc` attribute of the Rating Gauge component.

```
<dvtm:ratingGauge id="ratingGauge1"
                  value="#{pageFlowScope.value}"
                  minValue="#{pageFlowScope.minValue}"
                  maxValue="#{pageFlowScope.maxValue}"
                  shortDesc="#{pageFlowScope.shortDesc}"
                  inputIncrement="#{pageFlowScope.inputIncrement}"
                  inlineStyle="#{pageFlowScope.inlineStyle}"
   <dvtm:threshold id="tr1" maxValue="2" shortDesc="Performance: Poor"/>
   <dvtm:threshold id="tr2" maxValue="3" shortDesc="Performance: Average"/>
   <dvtm:threshold id="tr3" maxValue="4" shortDesc="Performance: Good"/>
   <dvtm:threshold id="tr4" maxValue="5" shortDesc="Performance: Excellent"/>
</dvtm:ratingGauge>
```

12.5.22.2 Applying Custom Styling to the Rating Gauge Component

Depending on the action performed by the user on a rating gauge component, its units (images) can acquire one of the following states:

- `selected`: the unit is selected.
- `unselected`: the unit is not selected.
- `changed`: the unit has been changed.
- `hover`: the unit is being hovered over.

> **Note:** On mobile devices with touch interface, the hover state is invoked through the tap-and-hold gesture.

Each state can be represented by its own array of images, as well as properties that define color and border color.

By default, the `shape` attribute of the `ratingGauge` determines the selection of the hover and changed states. The unselected state can be set separately using the `unselectedShape` attribute of the `ratingGauge`.

You can style the Rating Gauge component by overwriting the default CSS settings. For more information on how to extend CSS files, see Section 12.6.4, "How to Style Data Visualization Components".

The following shows the default CSS style definitions for the `color` and `borderColor` of each state of the rating gauge unit.

```
.dvtm-ratingGauge {
}

.dvtm-ratingGauge .dvtm-ratingGaugeSelected {
   border-width: 1px;
   border-style: solid;
   border-color: #FFC61A;
   color: #FFBB00;
}

.dvtm-ratingGauge .dvtm-ratingGaugeUnselected {
   border-width: 1px;
   border-style: solid;
   border-color: #D3D3D3;
   color: #F4F4F4;
}

.dvtm-ratingGauge .dvtm-ratingGaugeHover {
   border-width: 1px;
   border-style: solid;
   border-color: #6F97CF;
   color: #7097CF;
}

.dvtm-ratingGauge .dvtm-ratingGaugeChanged {
   border-width: 1px;
   border-style: solid;
   border-color: #A8A8A8;
   color: #FFBB00;
}
```

12.5.23 How to Define Child Elements for Chart and Gauge Components

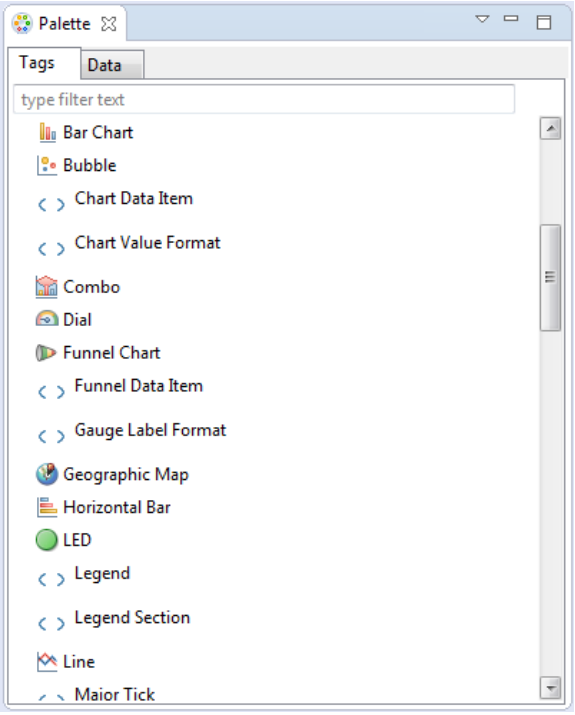
You can define a variety of child elements for charts and gauges. The following are some of these child elements:

- `chartDataItem` (see Section 12.5.23.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 12.5.23.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 12.5.23.2, "Defining and Configuring Legend")
- `pieDataItem` (see Section 12.5.23.4, "Defining Pie Data Item")
- `sparkDataItem` (see Section 12.5.23.5, "Defining Spark Data Item")
- `threshold` (see Section 12.5.23.8, "Defining Threshold")
- `funnelDataItem` (see Section 12.5.23.6, "Defining Funnel Data Item")
- `stockDataItem` (see Section 12.5.23.7, "Defining Stock Data Item")

For more information on these and other child elements, see *Tag Reference for Oracle Mobile Application Framework*.

In OEPE, child components of data visualization components are located under their respective visualization types (see Figure 12–68).

Figure 12–68 Creating Chart and Gauge Child Components



12.5.23.1 Defining Chart Data Item

The Chart Data Item (`chartDataItem`) element specifies the parameters that chart data items use in all supported charts, except the pie chart.

You can enable the text display on Chart Data Items and control its label, the label position, and the label style by setting relevant attributes of the `chartDataItem` element, as well as the `dataLabelPosition` attribute of the chart itself to specify the position of all data labels in a given chart.

> **Note:** The Spark Chart, Pie Chart, and Funnel Chart components do not support the `dataLabelPosition` attribute.

For information on attributes of the `chartDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.2 Defining and Configuring Legend

The Legend (`legend`) element specifies the legend parameters.

You can customize sizes of chart areas dedicated to legend using the Legend component's `size` and `maxSize` attributes.

For information on attributes of the `legend` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.3 Defining X Axis, YAxis, and Y2Axis

X Axis (`xAxis`) and Y Axis (`yAxis`) elements define the X and Y axis for a chart. Y2Axis (`y2Axis`) defines an optional Y2 axis. These elements are declared as follows in a MAF AMX file:

```
<dvtm:xAxis id="xAxis1" scrolling="on" axisMinValue="0.0" axisMaxValue="50.0" />
```

You can customize sizes of chart areas dedicated to axis using the `size` and `maxSize` attributes of the X Axis, Y Axis, and Y2Axis components. In addition, you can customize color, width, and style of the axis baseline by configuring its Major Tick child element.

For information on attributes and child elements of `xAxis`, `yAxis`, and `y2Axis` elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.4 Defining Pie Data Item

The Pie Data Item (`pieDataItem`) element specifies the parameters of the pie chart slices (see Section 12.5.7, "How to Create a Pie Chart").

For information on attributes of the `pieDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.5 Defining Spark Data Item

The Spark Data Item (`sparkDataItem`) element specifies the parameters of the spark chart items (see Section 12.5.9, "How to Create a Spark Chart").

For information on attributes of the `sparkDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.6 Defining Funnel Data Item

The Funnel Data Item (`funnelDataItem`) element specifies the parameters of the funnel chart items (see Section 12.5.10, "How to Create a Funnel Chart").

For information on attributes of the `funnelDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.23.7 Defining Stock Data Item

The Stock Data Item (`stockDataItem`) element specifies the parameters of the stock chart items (see Section 12.5.11, "How to Create a Stock Chart").

12.5.23.8 Defining Threshold

The Threshold (`threshold`) element specifies the threshold ranges of a gauge (see Section 12.5.19, "How to Create an LED Gauge" and Section 12.5.20, "How to Create a Status Meter Gauge").

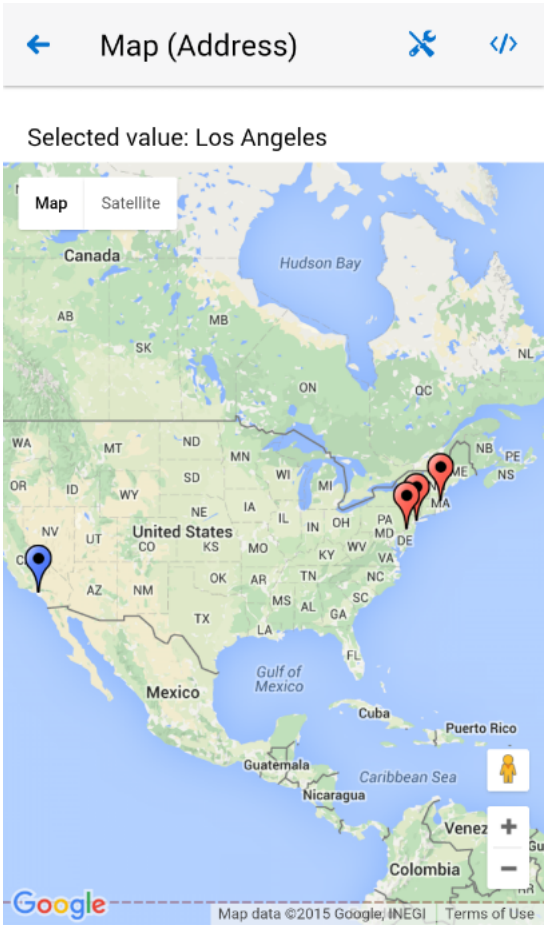
For information on attributes of the `threshold` element, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.24 How to Create a Geographic Map Component

A Geographic Map (`geographicMap`) represents data in one or more interactive layers of information superimposed on a single map. You configure this component to use either Google Maps or the Oracle Maps Cloud service (Oracle maps) as the underlying map provider. If you do not specify a map provider, the component uses Google Maps.

The following example shows the `geographicMap` component in the CompGallery sample application.

Figure 12–69 Geographic Map in CompGallery Sample Application



You can define a `pointDataLayer` child element for the `geographicMap`. The `pointDataLayer` allows you to display data associated with a point on the map.The `pointDataLayer` can have a `pointLocation` as a child element. The `pointLocation` specifies the columns in the data layer's model that determine the location of the data points. These locations can be represented either by address or by X and Y coordinates.

The `pointLocation` can have a `marker` as a child element. The `marker` is used to stamp out predefined or custom shapes associated with data points on the map. The `marker` supports a set of properties for specifying a URI to an image that is to be rendered as a marker. The marker can have a `convertNumber` as its child element (see Section 12.3.27, "How to Convert Numerical Values"). In addition, you can enable a Popup (see Section 12.2.8, "How to Use a Popup Component") to be displayed on a `geographicMap`'s marker. To do so, you declare the `showPopupBehavior` element as a child of the ***marker*** element, and then set the `showPopupBehavior`'s `alignId` attribute to the value of the `marker`'s `id` attribute, as the following example shows.

```
<dvtm:geographicMap id="geographicMap_1" shortDesc="#{pageFlowScope.shortDesc}">
   <dvtm:pointDataLayer id="pdl1"
                        var="row"
                        value=
                          "#{bindings.geographicMapPointData.collectionModel}">
 <dvtm:pointLocation id="pl1"
```

```
                        pointX="#{row.pointX}"
                        pointY="#{row.pointY}">
            <dvtm:marker id="marker1"
                         shortDesc="#{row.shortDesc}"
                         rendered="true">
               <amx:showPopupBehavior id="spb1"
                                      popupId="popup1"
                                      alignId="marker1"
                                      align="topCenter"
                                      decoration="anchor"/>
               <amx:setPropertyListener from="#{row.shortDesc}"
                                        to="#{pageFlowScope.currentCity}"
                                        type="action"/>
            </dvtm:marker>
         </dvtm:pointLocation>
      </dvtm:pointDataLayer>
</dvtm:geographicMap>
...
<amx:popup id="popup1" backgroundDimming="off" autoDismiss="true">
   <amx:outputText id="otTest" value="City: #{pageFlowScope.currentCity}"/>
   ...
</amx:popup>
```

For information on attributes of the `geographicMap` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

The Geographic Map component allows for insertion of a pin (creation of a point on the map) using a touch gesture. You can configure this functionality by using the `mapInputListener`. For more information, see Section 12.5.26, "How to Use Events with Map Components."

For more information about related tasks with the Geographic Map component, see:

- Section 12.5.24.1, "Configuring Geographic Map Components With the Map Provider Information"
- Section 12.5.24.2, "Displaying Route in Geographic Map Components"

12.5.24.1 Configuring Geographic Map Components With the Map Provider Information

To configure a Geographic Map component to use a specific provider for the underlying map (Google or Oracle), you can set the following properties as name-value pairs in the application's `adf-config.xml` file:

- `mapProvider`: specify either `oraclemaps` or `googlemaps`.
- `geoMapKey`: specify the license key if the `mapProvider` is set to `googlemaps`.
- `geoMapClientId`: if the `mapProvider` is set to `googlemaps`, specify the client ID for Google maps business license.
- `mapViewerUrl`: if the `mapProvider` is set to `oraclemaps`, specify the map viewer URL for Oracle maps.
- `baseMap`: if the `mapProvider` is set to `oraclemaps`, specify the base map to use with Oracle maps.

> **Note:** To configure the Geographic Map component to use Google maps, you must obtain an appropriate license from Google.

This example shows the configuration for Google maps.

```
<adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="mapProvider" value="googlemaps"/>
   <adf-property name="geoMapKey" value="your key"/>
</adf-properties-child>
```

Without a license for Google Maps, you have limited access to the geocoding service that enables address resolution. MAF provides a handler for error messages produced by the geocoding service when you exceed the permitted limit. These messages are displayed at runtime if the maximum allowed number of address points is exceeded. The number of requests is limited to 10 requests per second and redundant requests are not sent to the geocoding API.

Monitor the error messages listed in the following table.

Table 12–10 Error messages

Error ID	Message	Description
`OVER_QUERY_LIMIT`	GeoCoder quota has been exceeded.	Indicates that you are over your quota.
`REQUEST_DENIED`	Request denied! Check your API key and client ID.	Indicates that the request was denied, possibly because the request includes a result_type or location_type parameter but does not include an API key or client ID.

> **Note:** To use Oracle maps, you must abide by the Terms of Use and also abide by the Supplier Notices. Applications developed using Oracle maps must present the Terms of Use and the Supplier Notices to end users either in documentation or through links accessible from your application.

The example below shows the configuration for Oracle maps.

```
<adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="mapProvider" value="oraclemaps"/>
   <adf-property name="mapViewerUrl"
                 value="http://elocation.oracle.com/mapviewer"/>
   <adf-property name="baseMap" value="ELOCATION_MERCATOR.WORLD_MAP"/>
</adf-properties-child>
```

MAF applications that run on devices using iOS 9 or later must use `HTTPS` for all connections from the application to services to meet the requirements of Apple iOS's App Transport Security (ATS) policy. If you deploy your MAF application to iOS 9 or later, configure your application using one of the following options, so that the `geographicMap` component can render Oracle maps on an iOS 9 device:

1. Disable ATS when you deploy the MAF application. For more information, see Section 27.4.1, "How to Create an iOS Deployment Configuration.". This option is not recommended.
2. Configure the Oracle maps service to accept `HTTPS` requests and configure the MAF application so that the `geographicMap` component uses `HTTPS`. You perform the latter configuration in the application's `adf-config.xml` file. The following example demonstrates how you configure the application's `adf-config.xml` file so that the `geographicMap` component can render Oracle maps on an iOS 9 device.

```
    <adf:adf-properties-child
    xmlns="http://xmlns.oracle.com/adf/config/properties">
        <adf-property name="mapProvider" value="oracleMaps"/>
        <adf-property name="mapViewerUrl"
    value="https://elocation.myserver.com/mapviewer"/>
        <adf-property name="eLocationUrl"
    value="https://elocation.myserver.com/elocation"/>
      </adf:adf-properties-child>
    . . .
    </adf-config>
```

For information on the `adf-config.xml` file, see Section C.2, "About the Assembly-Level Resources".

12.5.24.2 Displaying Route in Geographic Map Components

When using Google Maps as a provider for the `geographicMap` component, you can specify route between two points with possible waypoints by adding a Route (`route`) child component.

Each geographicMap component can have multiple Route child components, with each specifying a single route. Route origin, destination and optional waypoints can be specified using the Geographic Map's Point Location child component. By convention, the first Point Location in the set defines the origin and the last defines the destination. All points between these two Point Locations represent route waypoints.

You can define the color, width, and opacity of the line used for visualizing the route in the map. In addition, you can specify a hint indicating whether the route should preferably follow driving routes, bicycling tracks, or walking paths.

The following example shows how to define a `route` element in a MAF AMX page.

```
<dvtm:geographicMap id="gm1">

   <!-- route defined using a collection model -->
   <dvtm:route travelMode="driving" id="d1">
      <amx:iterator value="#{el.collectionModel}" var="row">
         <dvtm:pointLocation address="#{row.address}" type="address"/>
      </amx:iterator>
   </dvtm:route>

   <!-- route with explicitly defined start and destination -->
   <dvtm:route travelMode="driving|walking|bicycling" id="d2">
      <!-- route origin -->
      <dvtm:pointLocation address="#{pageFlowScope.origin}" type="address">
      <!-- route destination -->
      <dvtm:pointLocation address="#{pageFlowScope.destination}" type="address"/>
   </dvtm:route/>

   <dvtm:pointDataLayer id="pdl1">
      ...
   </dvtm:pointDataLayer>

   <dvtm:pointDataLayer id="pdl2">
      ...
   </dvtm:pointDataLayer>

</dvtm:geographicMap>
```

When the end user clicks or taps on the line representing the route, an `ActionEvent` is fired. The event can be used to either drive navigation through the `action` attribute or

to invoke a handler in the Java layer using the `actionListener` attribute. The action can also be used to trigger event listeners and behaviors specified in child `setPropertyListener`, `actionListener`, `showPopupBehavior`, and `showPopupBehavior` elements. For more information, see Section 12.10, "Using Event Listeners."

12.5.25 How to Create a Thematic Map Component

A Thematic Map (`thematicMap`) represents business data as patterns in stylized areas or associated markers. Thematic maps focus on data without the geographic details.

The example below shows the `thematicMap` element and its children defined in a MAF AMX file.

```
<dvtm:thematicMap id="tm1"
                  animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
                  animationOnMapChange="#{pageFlowScope.animationOnMapChange}"
                  animationDuration="#{pageFlowScope.animationDuration}"
                  basemap="#{pageFlowScope.basemap}"
                           tooltipDisplay="#{pageFlowScope.tooltipDisplay}"
                  inlineStyle="#{pageFlowScope.inlineStyle}"
                  zooming="#{pageFlowScope.zooming}"
                  panning="#{pageFlowScope.panning}"
                  initialZooming="#{pageFlowScope.initialZooming}">
   <dvtm:areaLayer id="areaLayer1"
                   layer="#{pageFlowScope.layer}"
                   animationOnLayerChange=
                      "#{pageFlowScope.animationOnLayerChange}"
                   areaLabelDisplay="#{pageFlowScope.areaLabelDisplay}"
                   labelType="#{pageFlowScope.labelType}"
                   areaStyle="background-color"
                   rendered="#{pageFlowScope.rendered}">
      <dvtm:areaDataLayer id="areaDataLayer1"
                          animationOnDataChange=
                             "#{pageFlowScope.dataAnimationOnDataChange}"
                          animationDuration=
                             "#{pageFlowScope.dataAnimationDuration}"
                          dataSelection="#{pageFlowScope.dataSelection}"
                          var="row"
                          value="#{bindings.thematicMapData.collectionModel}">
         <dvtm:areaLocation id="areaLoc1" name="#{row.name}">
            <dvtm:area action="sales" id="area1" shortDesc="#{row.name}">
               <amx:setPropertyListener id="spl1"
                                        to=
                   "#{DvtProperties.areaChartProperties.dataSelection}"
                                        from="#{row.name}"
                                        type="action"/>
               <dvtm:attributeGroups id="ag1" type="color" value="#{row.cat1}" />
            </dvtm:area>
         </dvtm:areaLocation>
      </dvtm:areaDataLayer>
   </dvtm:areaLayer>
   <dvtm:legend id="l1" position="end">
      <dvtm:legendSection id="ls1" source="ag1"/>
   </dvtm:legend>
</dvtm:thematicMap>
```

Figure 12–70 Thematic Map at Design Time

Using the `markerZoomBehavior` attribute, you can enable scaling of the Thematic Map's markers when the map experiences zooming. You can enable the Marker rotation by setting its `rotation` attribute, whose value represents the angle at which the marker rotates in clockwise degrees around the center of the image.

MAF AMX Thematic Map supports the following advanced functionality:

- Custom markers (see Section 12.5.25.1, "Defining Custom Markers")
- Area isolation (see Section 12.5.25.3, "Defining Isolated Areas")
- Initial zooming (see Section 12.5.25.4, "Enabling Initial Zooming"
- Custom base maps (see Section 12.5.25.5, "Defining a Custom Base Map"

For information on attributes of the `thematicMap` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.25.1 Defining Custom Markers

MAF AMX Thematic Map does not support MAF AMX Image component. To use an image in the map's `pointLocation`, you can specify an image within the `pointLocation`'s `marker` child element by using its `source` attribute. If the `source` attribute is set on the Marker, its `shape` attribute is ignored by MAF AMX.

The `sourceHover`, `sourceSelected`, and `sourceHoverSelected` attributes allow you to specify images for hover and selection effects. If one of these is not specified, the image specified by the `source` attribute is used for that particular marker state. If `sourceSelected` is specified, then its value is used if `sourceHoverSelected` is not specified. The image can be of any format supported by the mobile device's browser, including PNG, JPG, SVG, and so on.

12.5.25.2 Defining Isolated Area Layers

A region outline is not always needed to convey the geographic location of data. Instead, since the Thematic Map component has the option of centering an image or marker within an area, you have the option of defining invisible area layers where region outlines are not drawn.

To define an invisible area layer, you use the `areaStyle` attribute of the `areaLayer` which accepts the CSS values of `background-color` and `border-color` as follows:

```
<dvtm:areaLayer id="areaLayer1"
                ...
                areaStyle="background-color:transparent;border-color:transparent">
```

This attribute allows you to override the default area layer color and border treatments without using the `dvtm-area` skinning key.

12.5.25.3 Defining Isolated Areas

You can configure the MAF AMX Thematic Map component to render and zoom to fit on a single isolated area of the map by using the `isolatedRowKey` attribute of the `areaDataLayer`, in which case the rest of the areas in the area or area data layers is not rendered.

Note: You can isolate only one area on a map.

12.5.25.4 Enabling Initial Zooming

The initial zooming allows the map component to be rendered as usual, and then zoom to fit on the data objects which includes both markers and areas. To enable this functionality, you use the `initialZooming` attribute of the Thematic Map.

12.5.25.5 Defining a Custom Base Map

As part of the custom base map support, MAF AMX allows you to specify the following for the Thematic Map component:

- Layers with images for different resolutions.
- Point layers with named points that can be referenced from the Point Location (`pointLocation`).
- The Thematic Map's `source` attribute that points to the custom base map metadata XML file.

Note: MAF AMX does not support the following for custom base maps:

- Stylized areas: since area layers cannot be defined for custom base maps, use point layers.
- Resource bundles: if you want to add locale-specific tool tips, you can use EL in the `shortDesc` attribute of the Marker (`marker`).

To create a custom base map, you specify an area layer which points to a definition in the metadata file (see the following example). To define a basic custom base map, you specify a background layer and a pointer data layer. In the metadata file, you can specify different images for different screen resolutions and display directions, similar to MAF AMX gauge components. Just like a gauge-type component, the Thematic Map chooses the correct image for the layer based on the screen resolution and direction. The display direction is left-to-right.

You can define any number of layers. All named points are accessible in all the layers. The X and Y positions of the named points are mapped to the image dimensions of the first image. The Thematic Map component calculates the position of the points when one of the following occurs:

- Zooming in is performed.
- A different image is displayed in a different resolution.

```
<basemap id="car" >
   <layer id="exterior" >
      <image source="/maps/car-800x800.png"
             width="2560"
             height="1920" />
      <image source="/maps/car-200x200.png"
```

```
            width="640"
            height="480" />
   </layer>
</basemap>
```

The following example shows a MAF AMX file that declares a custom area layer with points. The MAF AMX file points to the metadata file shown in the preceding example containing a list of possible images, which are, in fact, scaled versions of the same image.

```
<dvtm:thematicMap id="tm1" basemap="car" source="customBasemaps/map1.xml" >
   <dvtm:areaLayer id="al1" layer="exterior" >
      <dvtm:pointDataLayer id="pdl1"
                           var="row"
                           value="{bindings.thematicMapData.collectionModel}" >
         <dvtm:pointLocation id="pl1"
                             type="pointXY"
                             pointX="#{row.x}"
                             pointY="#{row.y}" >
            <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
         </dvtm:pointLocation>
      </dvtm:pointDataLayer>
   </dvtm:areaLayer>
</dvtm:thematicMap>
```

To create a custom base map with static points, you specify the points by name in the metadata file shown in the following example. This process is similar to adding city markers for a predefined base map.

```
<basemap id="car" >
   <layer id="exterior" >
      <image source="/maps/car-800x800.png"
             width="2560"
             height="1920" />
      <image source="/maps/car-800x800-rtl.png"
             width="2560"
             height="1920"
             dir="rtl" />
      <image source="/maps/car-200x200.png"
             width="640"
             height="480" />
      <image source="/maps/car-200x200-rtl.png"
             width="640"
             height="480"
             dir="rtl" />
   </layer>
   <points >
      <point name="hood" x="219.911" y="329.663" />
      <point name="frontLeftTire" x="32.975" y="32.456" />
      <point name="frontRightTire" x="10.334" y="97.982" />
   </points>
</basemap>
```

The X and Y positions of the named points are assumed to be mapped to the image dimensions of the first `image` element in the `layer`.

> **Note:** Since the points are global in scope within the base map and apply to all layers, you cannot define points for a specific layer and its images.

The following example shows a MAF AMX file that declares a custom area layer with named points.

```
<dvtm:thematicMap id="demo1" basemap="car" source="customBasemaps/map1.xml" >
   <dvtm:areaLayer id="al1" layer="exterior" />
   <dvtm:pointDataLayer id="pdl1"
                        var="row"
                        value="#{bindings.thematicMapData.collectionModel}" >
      <dvtm:pointLocation id="pl1" type="pointName" pointName="#{row.name}" >
         <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
      </dvtm:pointLocation>
   </dvtm:pointDataLayer>
</dvtm:thematicMap>
```

The preceding MAF AMX file refers to the metadata file shown in the following example containing a list of points and their names.

```
<basemap id="car" >
   <layer id="exterior" >
      <image source="/maps/car-800x800.png"
             width="2560"
             height="1920" />
      <image source="/maps/car-200x200.png"
             width="640"
             height="480" />
   </layer>
</basemap>
```

12.5.25.6 What You May Need to Know About the Marker Support for Event Listeners

MAF AMX data visualization does not support the `actionListener` attribute for the `marker`. Instead, the same functionality can be achieved by using the `action` attribute.

12.5.25.7 Applying Custom Styling to the Thematic Map Component

You can style the Thematic Map component by overwriting the default CSS settings or using a custom JavaScript file. For more information on how to extend these files, see Section 12.6.4, "How to Style Data Visualization Components."

This example shows the default CSS styles for the Thematic Map component.

```
.dvtm-thematicMap {
   background-color: #FFFFFF;
   -webkit-user-select: none;
   -webkit-touch-callout: none;
   -webkit-tap-highlight-color: rgba(0,0,0,0);
}

.dvtm-areaLayer {
   background-color: #B8CDEC;
   border-color: #FFFFFF;
   border-width: 0.5px;
   /* border style and color must be set when setting border width */
   border-style: solid;
   color: #000000;
   font-family: tahoma, sans-serif;
   font-size: 13px;
   font-weight: bold;
   font-style: normal;
```

```
}

.dvtm-area {
   border-color: #FFFFFF;
   border-width: 0.5px;
   /* border style and color must be set when setting border width */
   border-style: solid;
}

.dvtm-marker {
   background-color: #61719F;
   opacity: 0.7;
   color: #FFFFFF;
   font-family: tahoma, sans-serif;
   font-size: 13px;
   font-weight: bold;
   font-style: normal;
   border-style: solid
   border-color: #FFCC33
   border-width: 12px
}
```

Some of the style settings cannot be specified using CSS. Instead, you must define them using a custom JavaScript file. The example below shows how to apply custom styling to the Thematic Map component without using CSS.

my-custom.js:

```
   CustomThematicMapStyle = {
      // selected area properties
      'areaSelected': {
         // selected area border color
         'borderColor': "#000000",
         // selected area border width
         'borderWidth': '1.5px'
      },

      // area properties on mouse hover
      'areaHover': {
         // area border color on hover
         'borderColor': "#FFFFFF",
         // area border width on hover
         'borderWidth': '2.0px'
      },

      // marker properties
      'marker': {
         // separator upper color
         'scaleX': 1.0,
         // separator lower color
         'scaleY': 1.0,
         // should display title separator
         'type': 'circle'
      },

      // thematic map legend properties
      'legend': {
         // legend position, such as none, auto, start, end, top, bottom
         'position': "auto"
      }
```

```
   };

})();
```

> **Note:** You cannot change the name and the property names of the `CustomThematicMapStyle` object. Instead, you can modify specific property values to suit the needs of your application.

When the `attributeGroups` attribute is defined for the Thematic Map component, you can use the `CustomThematicMapStyle` to define a default set of shapes and colors for that component. In this case, the CustomThematicMapStyle object must have the structure that the next example shows, where `styleDefaults` is a nested object containing the following fields:

- `colors`: represents a set of colors to be used for areas and markers.
- `shapes`: represents a set of shapes to be used for markers.

```
window['CustomThematicMapStyle'] =
{
   // custom style values
   'styleDefaults': {
      // custom color palette
      'colors': ["#000000", "#ffffff"],
      // custom marker shapes
      'shapes' : ['circle', 'square']
   }
};
```

12.5.26 How to Use Events with Map Components

You can use the `MapBoundsChangeEvent` to handle the following map view property changes in the Geographic Map component:

- Changes to the zoom level.
- Changes to the map bounds.
- Changes to the map center.

When these changes occur, the component fires an event loaded with new map view property values.

You can define the `mapBoundsChangeListener` as an attribute of the Geographic Map.

You can use the `MapInputEvent` to handle the end user actions, such as taps and mouse clicks, in the Geographic and Thematic Map components. When these actions occur, the component fires an event loaded with the information on the latitude and longitude for the map, as well as the type of the action (for example, mouse down, mouse up, click, and so on).

You can define the `mapInputListener` as an attribute of the Geographic Map component.

For more information, see the following:

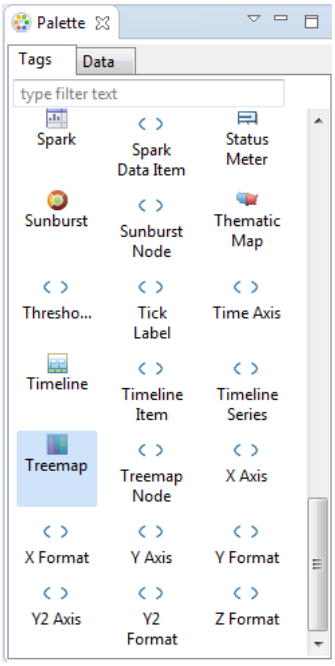
- Section 12.10, "Using Event Listeners"
- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*

12.5.27 How to Create a Treemap Component

A Treemap (`treemap`) displays hierarchical data across two dimensions represented by the size and color of its nodes (`treemapNode`).

In the Palette, the Treemap is located under **MAF AMX Data Visualizations > Treemap** (see Figure 12–71).

Figure 12–71 Treemap Location in the Palette

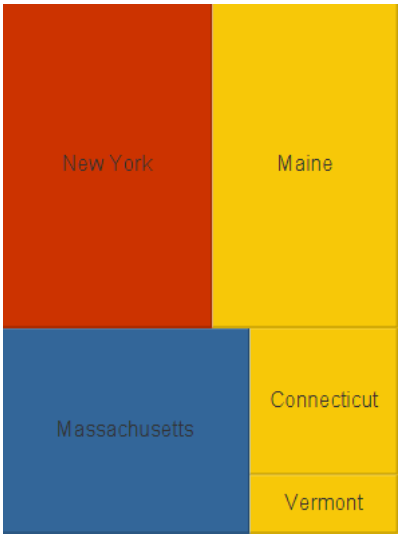


This example shows the `treemap` element and its children defined in a MAF AMX file.

```
<dvtm:treemap id="treemap1"
              value="#{bindings.treemapData.collectionModel}"
              var="row"
              animationDuration="#{pageFlowScope.animationDuration}"
              animationOnDataChange="#{pageFlowScope.animationOnDataChange}"
              animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
              layout="#{pageFlowScope.layout}"
              nodeSelection="#{pageFlowScope.nodeSelection}"
              rendered="#{pageFlowScope.rendered}"
              emptyText="#{pageFlowScope.emptyText}"
              inlineStyle="#{pageFlowScope.inlineStyle}"
              sizeLabel="#{pageFlowScope.sizeLabel}"
              styleClass="dvtm-gallery-component"
              colorLabel="#{pageFlowScope.colorLabel}"
              sorting="#{pageFlowScope.sorting}"
              selectedRowKeys="#{pageFlowScope.selectedRowKeys}"
              isolatedRowKey="#{pageFlowScope.isolatedRowKey}"
              legendSource="ag1">
  <dvtm:treemapNode id="node1"
                    fillPattern="#{pageFlowScope.fillPattern}"
                    label="#{row.label}"
                    labelDisplay="#{pageFlowScope.labelDisplay}"
                    value="#{row.marketShare}"
                    labelHalign="#{pageFlowScope.labelHalign}"
                    labelValign="#{pageFlowScope.labelValign}">
```

```
      <dvtm:attributeGroups id="ag1"
                            type="color"
                            value="#{row.deltaInPosition}"
                            attributeType="continuous"
                            minLabel="-1.5%"
                            maxLabel="+1.5%"
                            minValue="-1.5"
                            maxValue="1.5" >
         <amx:attribute id="a1" name="color1" value="#ed6647" />
         <amx:attribute id="a2" name="color2" value="#f7f37b" />
         <amx:attribute id="a3" name="color3" value="#68c182" />
      </dvtm:attributeGroups>
   </dvtm:treemapNode>
</dvtm:treemap>
```

Figure 12–72 Treemap at Design Time



By setting the `attributeType` attribute of the `attributeGroups` element to `continuous`, you can enable visualization of a value associated with the Treemap item using a gradient color where the color intensity represents the relative value within a specified range.

For information on attributes of the `treemap` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.27.1 Applying Custom Styling to the Treemap Component

You can style the Treemap component by overwriting the default CSS settings or using a custom JavaScript file. For more information on how to extend these files, see Section 12.6.4, "How to Style Data Visualization Components."

This example shows the Treemap component's default CSS styles that you can override.

```
.dvtm-treemap {
   border-style: solid;
   border-color: #E2E8EE;
   border-radius: 3px;
   background-color: #EDF2F7;
   ...
}
```

This example shows the Treemap Node's default CSS styles that you can override.

```
.dvtm-treemapNodeSelected {
   // Selected node outer border color
   border-top-color: #E2E8EE;
   // Selected node inner border color
   border-bottom-color: #EDF2F7;
}
```

This example shows the Treemap Node's `label` text CSS properties that you can style using custom CSS.

```
.dvtm-treemapNodeLabel {
   font-family: Helvetica, sans-serif;
   font-size: 14px;
   font-style: normal;
   font-weight: normal;
   color: #7097CF;
   ...
}
```

Some of the style settings cannot be specified using CSS. Instead, you must define them using a custom JavaScript file. The next example shows how to apply custom styling to the Treemap component without using CSS.

```
my-custom.js:

   window["CustomTreemapStyle"] = {

      // treemap properties
      "treemap" : {
         // Specifies the animation effect when the data changes - none, auto
         "animationOnDataChange": "auto",

         // Specifies the animation that is shown on initial display - none, auto
         "animationOnDisplay": "auto",

         // Specifies the animation duration in milliseconds
         "animationDuration": "500",

         // The text of the component when empty
         "emptyText": "No data to display",

         // Specifies the layout of the treemap -
         // squarified, sliceAndDiceHorizontal, sliceAndDiceVertical
         "layout": "squarified",

         // Specifies the selection mode - none, single, multiple
         "nodeSelection": "multiple",

         // Specifies whether or not the nodes are sorted by size - on, off
         "sorting": "on"
      },

      // treemap node properties
      "node" : {
         // Specifies the label display behavior for nodes - node, off
         "labelDisplay": "off",

         // Specifies the horizontal alignment for labels displayed
```

```
        // within the node - center, start, end
        "labelHalign": "end",

        // Specifies the vertical alignment for labels displayed
        // within the node - center, top, bottom
        "labelValign": "center"
      },
   }
```

12.5.28 How to Create a Sunburst Component

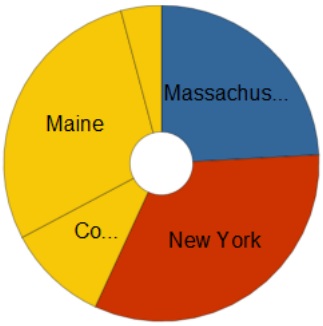
A Sunburst (`sunburst`) displays hierarchical data across two dimensions represented by the size and color of its nodes (`sunburstNode`).

In the Palette, the Sunburst is located under **MAF AMX Data Visualizations**, as are its child nodes.

This example shows the `sunburst` element and its children defined in a MAF AMX file.

```
<dvtm:sunburst id="sunburst1"
               value="#{bindings.sunburstData.collectionModel}"
               var="row"
               animationDuration="#{pageFlowScope.animationDuration}"
               animationOnDataChange="#{pageFlowScope.animationOnDataChange}"
               animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
               colorLabel="#{pageFlowScope.colorLabel}"
               emptyText="#{pageFlowScope.emptyText}"
               inlineStyle="#{pageFlowScope.inlineStyle}"
               nodeSelection="#{pageFlowScope.nodeSelection}"
               rendered="#{pageFlowScope.rendered}"
               rotation="#{pageFlowScope.rotation}"
               shortDesc="#{pageFlowScope.shortDesc}"
               sizeLabel="#{pageFlowScope.sizeLabel}"
               sorting="#{pageFlowScope.sorting}"
               rotationAngle="#{pageFlowScope.startAngle}"
               styleClass="#{pageFlowScope.styleClass}"
               legendSource="ag1">
   <dvtm:sunburstNode id="node1"
                      fillPattern="#{pageFlowScope.fillPattern}"
                      label="#{row.label}"
                      labelDisplay="#{pageFlowScope.labelDisplay}"
                      value="#{pageFlowScope.showRadius ? 1 : row.marketShare}"
                      labelHalign="#{pageFlowScope.labelHalign}"
                      radius="#{pageFlowScope.showRadius ? row.booksCount : 1}">
      <dvtm:attributeGroups id="ag1"
                            type="color"
                            value="#{row.deltaInPosition}"
                            attributeType="continuous"
                            minLabel="-1.5%"
                            maxLabel="+1.5%"
                            minValue="-1.5"
                            maxValue="1.5">
         <amx:attribute id="a1" name="color1" value="#ed6647" />
         <amx:attribute id="a2" name="color2" value="#f7f37b" />
         <amx:attribute id="a3" name="color3" value="#68c182" />
      </dvtm:attributeGroups>
   </dvtm:sunburstNode>
</dvtm:sunburst>
```

Figure 12–73 Sunburst at Design Time



By setting the `attributeType` attribute of the `attributeGroups` element to `continuous`, you can enable visualization of a value associated with the Sunburst item using a gradient color where the color intensity represents the relative value within a specified range.

For information on attributes of the `sunburst` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.28.1 Applying Custom Styling to the Sunburst Component

You can style the Sunburst component by overwriting the default CSS settings or using a custom JavaScript file. For more information on how to extend these files, see Section 12.6.4, "How to Style Data Visualization Components."

This example shows the Sunburst component's default CSS styles that you can override.

```
.dvtm-sunburst {
   border-style: solid;
   border-color: #E2E8EE;
   border-radius: 3px;
   background-color: #EDF2F7;
   ...
}
```

This example shows the Sunburst Node's default CSS styles that you can override.

```
.dvtm-sunburstNode {
   // Node border color
   border-color: "#000000";
}

.dvtm-sunburstNodeSelected {
   // Selected node border color
   border-color: "#000000";
}
```

The next example shows the Sunburst Node's `label` text CSS properties that you can style using custom CSS.

```
.dvtm-sunburstNodeLabel {
   font-family: Helvetica, sans-serif;
   font-size: 14px;
   font-style: normal;
   font-style: normal;
   color: #7097CF;
   ...
}
```

Some of the style settings cannot be specified using CSS. Instead, you must define them using a custom JavaScript file. The next example shows how to apply custom styling to the Sunburst component without using CSS.

my-custom.js:

```
window["CustomSunburstStyle"] = {
   // sunburst properties
   "sunburst" : {
      // Specifies whether or not the client side rotation is enabled - on, off
      "rotation": "off",

      // The text of the component when empty
      "emptyText": "No data to display",

      // Specifies the selection mode - none, single, multiple
      "nodeSelection": "multiple",

      // Animation effect when the data changes - none, auto
      "animationOnDataChange": "auto",

      // Specifies the animation that is shown on initial display - none, auto
      "animationOnDisplay": "auto",

      // Specifies the animation duration in milliseconds
      "animationDuration": "500",

      // Specifies the starting angle of the sunburst
      "startAngle": "90",

      // Specifies whether or not the nodes are sorted by size - on, off
      "sorting": "on"
   },

   // sunburst node properties
   "node" : {
      // Specifies whether or not the label is displayed - on, off
      "labelDisplay": "off"
   }
}
```

12.5.29 How to Create a Timeline Component

A Timeline (`timeline`) is an interactive component that allows viewing of events in chronological order, as well as navigating forward and backward within a defined yet adjustable time range that can be used for zooming.

Events are represented by Timeline Item components (`timelineItem`) that include the title, description, and duration fill color. You can configure the Timeline component to display an overview window (`overview` child element) showing all available events. The end user can zoom in and out of the events using pinch and spread gestures on a mobile device. In addition, you can configure a dual timeline to display two series of events for a side-by-side comparison of related information.

You can define the Timeline component as either horizontal or vertical using its `orientation` attribute.

Note: MAF AMX does not support the following functionality, child elements, and properties that are often available in components similar to the Timeline:

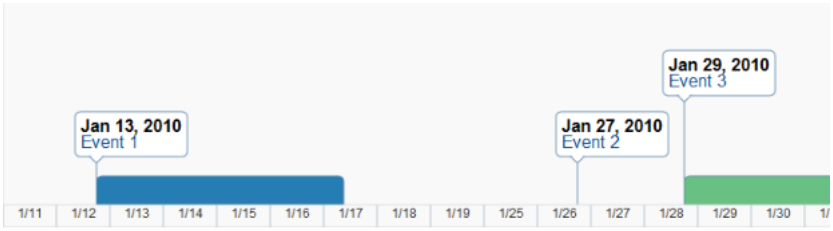
- Nested UI components
- Animation
- Attribute and time range change awareness
- Time fetching
- Custom time scales
- Time currency
- Partial triggers
- Data sorting
- Formatted time ranges
- Time zone
- Visibility

In the Palette, the Timeline is located under **MAF AMX Data Visualizations**, as are its child components Timeline Item and Timeline Series.

This example shows the `timeline` element and its children defined in a MAF AMX file.

```
<dvtm:timeline id="tl"
               itemSelection="#{pageFlowScope.itemSelection}"
               startTime="#{pageFlowScope.startTime}"
               endTime="#{pageFlowScope.endTime}">
   <dvtm:timelineSeries id="ts1"
                        label="#{pageFlowScope.s1Label}"
                        value="#{bindings.series1Data.collectionModel}"
                        var="row"
                        selectionListener=
                            "#{PropertyBean.timelineSeries1SelectionHandler}">
      <dvtm:timelineItem id="ti1"
                         startTime="#{row.startDate}"
                         endTime="#{row.endDate}"
                         title="#{row.title}"
                         description="#{row.description}"
                         durationFillColor="#AAAAAA"/>
   </dvtm:timelineSeries>
   <dvtm:timeAxis id="ta1" scale="#{pageFlowScope.scale}"/>
</dvtm:timeline>
```

Figure 12–74 Timeline at Design Time



You can control the fill color of a specific Timeline Item's duration bar using its `durationFillColor` attribute.

To display two time scales at the same time on the Timeline, use the Time Axis' `scale` attribute that determines the scale of the second axis.

The Timeline can be scrolled horizontally as well as vertically. When the component is scrollable (that is, contains data outside of the visible display area), it is indicated by arrows pointing in the direction of the scroll.

> **Tip:** If the Timeline start time is set to the same value as the start time of the first Timeline Item, the bubbles of corresponding Timeline Item components might appear truncated. In addition, if the Timeline end time is set to the same value as the end time of the last Timeline Item, the bubbles of corresponding Timeline Item components might appear truncated. You should set the start and end time of the Timeline component such that it ensures full visibility of all Timeline Item bubbles.

For information on attributes of the `timeline` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.29.1 Applying Custom Styling to the Timeline Component

You can style the Timeline component by overwriting the default CSS settings or using a custom JavaScript file. For more information on how to extend these files, see Section 12.6.4, "How to Style Data Visualization Components."

The following CSS style classes that you can override are defined for the Timeline and its child components:

- `.dvtm-timeline`

  supported properties: `all`

- `.timelineSeries-backgroundColor`

  supported properties: `color`

  `.timelineSeries-labelStyle`

  supported properties: `font-family, font-size, font-weight, color, font-style`

- `.timelineItem-backgroundColor`

  supported properties: `color`

  `.timelineItem-selectedBackgroundColor`

supported properties: `color`

`.timelineItem-borderColor`

supported properties: `color`

`.timelineItem-selectedBorderColor`

supported properties: `color`

`.timelineItem-borderWidth`

supported properties: `width`

`.timelineItem-selectedBorderWidth`

supported properties: `width`

`.timelineItem-feelerColor`

supported properties: `color`

`.timelineItem-selectedFeelerColor`

supported properties: `color`

`.timelineItem-feelerWidth`

supported properties: `width`

`.timelineItem-selectedFeelerWidth`

supported properties: `width`

`.timelineItem-descriptionStyle`

- supported properties: `font-family, font-size, font-weight, color, font-style`

`.timelineItem-titleStyle`

- supported properties: `font-family, font-size, font-weight, color, font-style`

- `.timeAxis-separatorColor`

  supported properties: `color`

  `.timeAxis-backgroundColor`

  supported properties: `color`

  `.timeAxis-borderColor`

  supported properties: `color`

  `.timeAxis-borderWidth`

  supported properties: `width`

  `.timeAxis-labelStyle`

  - supported properties: `font-family, font-size, font-weight, color, font-style`

This example shows a custom JavaScript file that you could use to override the default styles of the Timeline component.

```
// Custom timeline style definition with listing
// of all properties that can be overriden
window["CustomTimelineStyle"] = {
   // Determines if items in the timeline are selectable
```

```
   "itemSelection": none

   // Timeline properties
   "timelineSeries" : {
      // Duration bars color palette
      "colors" : [comma separated list of hex colors]
   }
}
```

12.5.30 How to Create an NBox Component

An NBox (`nBox`) component presents data across two dimensions, with each dimension split into a number of ranges whose intersections form distinct cells into which each data item is placed.

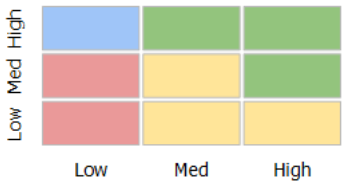
In the Palette, the NBox is located under **MAF AMX Data Visualizations**.

This example shows the `nBox` element and its children defined in a MAF AMX file.

```
<dvtm:nBox id="nBox1"
           var="item"
           value="#{bindings.NBoxNodesDataList.collectionModel}"
           columnsTitle="#{pageFlowScope.columnsTitle}"
           emptyText="#{pageFlowScope.emptyText}"
           groupBy="#{pageFlowScope.groupBy}"
           groupBehavior="#{pageFlowScope.groupBehavior}"
           highlightedRowKeys="#{pageFlowScope.showHighlightedNodes ?
                                 pageFlowScope.highlightedRowKeys : ''}"
           inlineStyle="#{pageFlowScope.inlineStyle}"
           legendDisplay="#{pageFlowScope.legendDisplay}"
           maximizedColumn="#{pageFlowScope.maximizedColumn}"
           maximizedRow="#{pageFlowScope.maximizedRow}"
           nodeSelection="#{pageFlowScope.nodeSelection}"
           rowsTitle="#{pageFlowScope.rowsTitle}"
           selectedRowKeys="#{pageFlowScope.selectedRowKeys}"
           shortDesc="#{pageFlowScope.shortDesc}">
   <amx:facet name="rows">
      <dvtm:nBoxRow value="low" label="Low" id="nbr1"/>
       <dvtm:nBoxRow value="medium" label="Med" id="nbr2"/>
       <dvtm:nBoxRow value="high" label="High" id="nbr3"/>
   </amx:facet>
   <amx:facet name="columns">
      <dvtm:nBoxColumn value="low" label="Low" id="nbc2"/>
      <dvtm:nBoxColumn value="medium" label="Med" id="nbc1"/>
      <dvtm:nBoxColumn value="high" label="High" id="nbc3"/>
   </amx:facet>
   <amx:facet name="cells">
      <dvtm:nBoxCell row="low"
                     column="low"
                     label=""
                     background="rgb(234,153,153)"
                     id="nbc4"/>
      <dvtm:nBoxCell row="medium"
                     column="low"
                     label=""
                     background="rgb(234,153,153)"
                     id="nbc5"/>
      <dvtm:nBoxCell row="high"
                     column="low"
                     label=""
                     background="rgb(159,197,248)"
```

```
                    id="nbc6"/>
      <dvtm:nBoxCell row="low"
                    column="medium"
                    label=""
                    background="rgb(255,229,153)"
                    id="nbc7"/>
      <dvtm:nBoxCell row="medium"
                    column="medium"
                    label=""
                    background="rgb(255,229,153)"
                    id="nbc8"/>
      <dvtm:nBoxCell row="high"
                    column="medium"
                    label=""
                    background="rgb(147,196,125)"
                    id="nbc9"/>
      <dvtm:nBoxCell row="low"
                    column="high"
                    label=""
                    background="rgb(255,229,153)"
                    id="nbc10"/>
      <dvtm:nBoxCell row="medium"
                    column="high"
                    label=""
                    background="rgb(147,196,125)"
                    id="nbc11"/>
      <dvtm:nBoxCell row="high"
                    column="high"
                    label=""
                    background="rgb(147,196,125)"
                    id="nbc12"/>
   </amx:facet>
   <dvtm:nBoxNode id="nbn1"
                  row="#{item.row}"
                  column="#{item.column}"
                  label="#{item.name}"
                  labelStyle="font-style:italic"
                  secondaryLabel="#{item.job}"
                  secondaryLabelStyle="font-style:italic"
                  shortDesc="#{item.name + ': ' + item.job}">
      <dvtm:attributeGroups id="ag1"
                            type="indicatorShape"
                            value="#{item.indicator1}"
                            rendered="#{pageFlowScope.showIndicator}"/>
      <dvtm:attributeGroups id="ag2"
                            type="indicatorColor"
                            value="#{item.indicator2}"
                            rendered="#{pageFlowScope.showIndicator}"/>
      <dvtm:attributeGroups id="ag3"
                            type="color"
                            value="#{item.group}"
                            rendered="#{pageFlowScope.showColors}"/>
   </dvtm:nBoxNode>
</dvtm:nBox>
```

Figure 12–75 NBox at Design Time



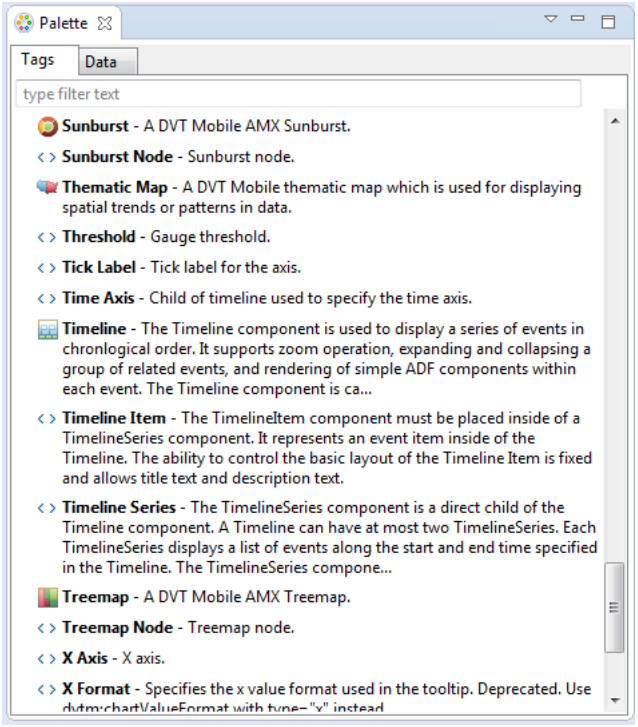
For information on attributes of the `nBox` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.31 How to Define Child Elements for Map Components, Sunburst, Treemap, Timeline, and NBox

You can define a variety of child elements for map components, Sunburst, Treemap, Timeline, and NBox. For information on available child elements and their attributes, see *Tag Reference for Oracle Mobile Application Framework*.

In OEPE, the Map, Sunburst, Treemap, Timeline, and NBox child components are located under **MAF AMX Data Visualizations** in the Palette (see Figure 12–76). In the figure, the Details option has been selected from **Palette > Layout**. This displays the tooltip content beside each icon in the MAF AMX Data Visualizations list.

Figure 12–76 Creating Map, Sunburst, Treemap, Timeline, and NBox Child Components

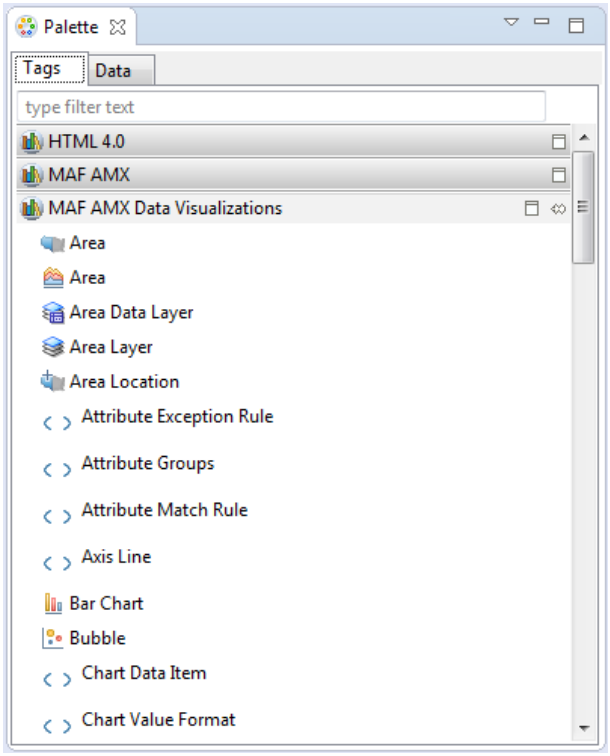


12.5.32 How to Create Databound Data Visualization Components

You can declaratively create a databound data visualization component using a data collection inserted from the Data Controls window (see Section 11.3.2.3, "Adding Data Controls to the View"). The **palette** selection that Figure 12–87 shows allows you to

choose from a number of data visualization component categories, types, and layout options, all available under the heading **MAF AMX Data Visualizations**.

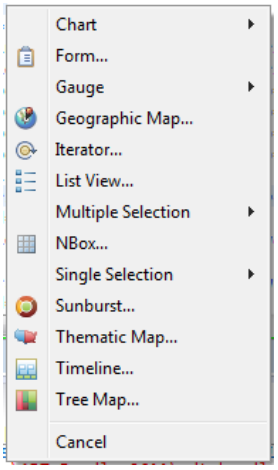
Figure 12–77 Palette Selection to Create Chart Components



Note: Some data visualization component types require very specific kinds of data. If you bind a component to a data collection that does not contain sufficient data to display the component type requested, then the component is not displayed and a message about insufficient data appears.

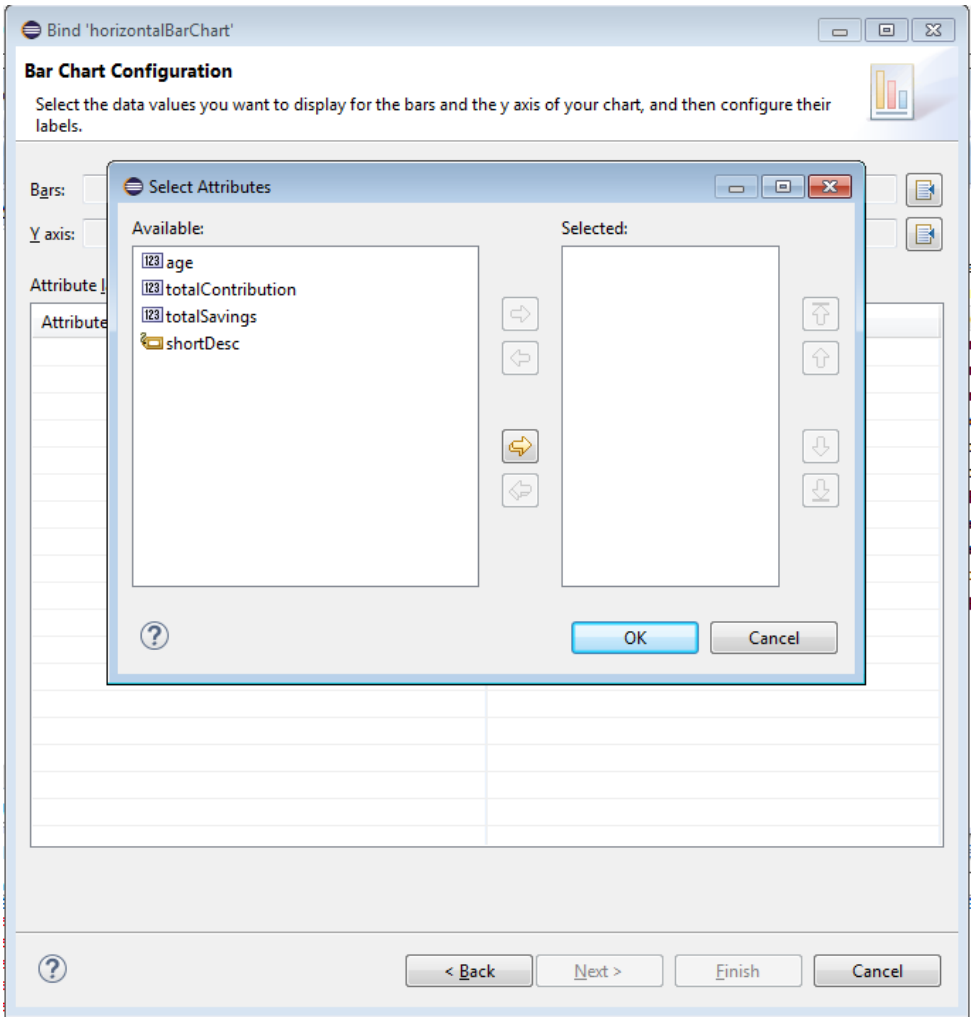
To trigger the display of the **Component Gallery**, you drag and drop a collection from the Data Controls window onto a MAF AMX page, and then select either **MAF Chart**, **MAF Gauge**, or **MAF Thematic Map** from the context menu that appears (see Figure 12–78).

Figure 12–78 Creating Databound Data Visualization Components



After you select the category, type, and layout for your new databound component from the Component Gallery and click **OK**, you can start setting values the data collection attributes in the data visualization component. The name of the gallery, the dialog and the input field labels depend on the category and type of the data visualization component that you selected. For example, if you drag a data control of type `chartData` into your app, then select **Chart > Horizontal Bar** as the type, then the name of the dialog that appears is **Bind Horizontal Bar Chart**, giving you the ability to select the bound data elements (`age`, `totalContribution`, `totalSavings`, and `shortDesc`) and apply them either to the bars or to the Y axis, as Figure 12–79 shows.

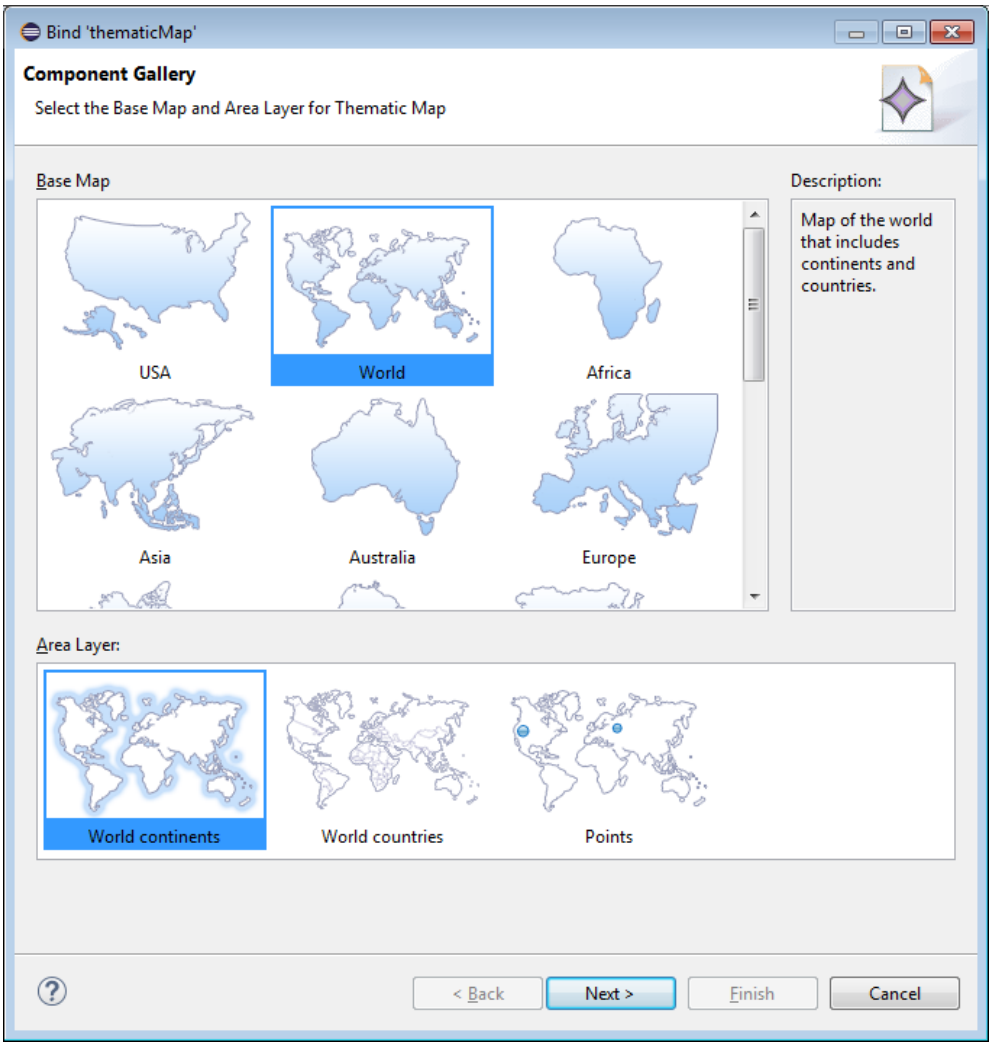
Figure 12–79 Specifying Data Values for Databound Chart



The attributes in a data collection can be data values or categories of data values. Data values are numbers represented by markers, like bar height, or points in a scatter chart. Categories of data values are members represented as axis labels. The role that an attribute plays in the bindings (either data values or identifiers) is determined by both its data type (chart requires numeric data values) and where it is mapped (for example, Bars or X Axis).

If you use the Component Gallery to create a databound thematic map component, then the name of the dialog that appears is **Bind 'thematicMap'** and the Component Gallery lets you select the base map, with detail selections as the area layer. For example, if you select World as the base map and World continents as the area layer, the dialog shown in Figure 12–80 opens.

Figure 12–80 Bind 'thematicMap' Dialog



After completing one or more data binding dialogs, you can use the Properties window to specify settings for the component attributes. You can also use the child elements associated with the component to further customize it (see Section 12.5.23, "How to Define Child Elements for Chart and Gauge Components").

When you select **MAF Geographic Map**, **MAF Sunburst**, **MAF NBox**, **MAF Timeline**, or **MAF Treemap** from the context menu upon dropping a collection onto a MAF AMX page, one of the following dialogs appear:

Figure 12–81 Creating Geographic Map

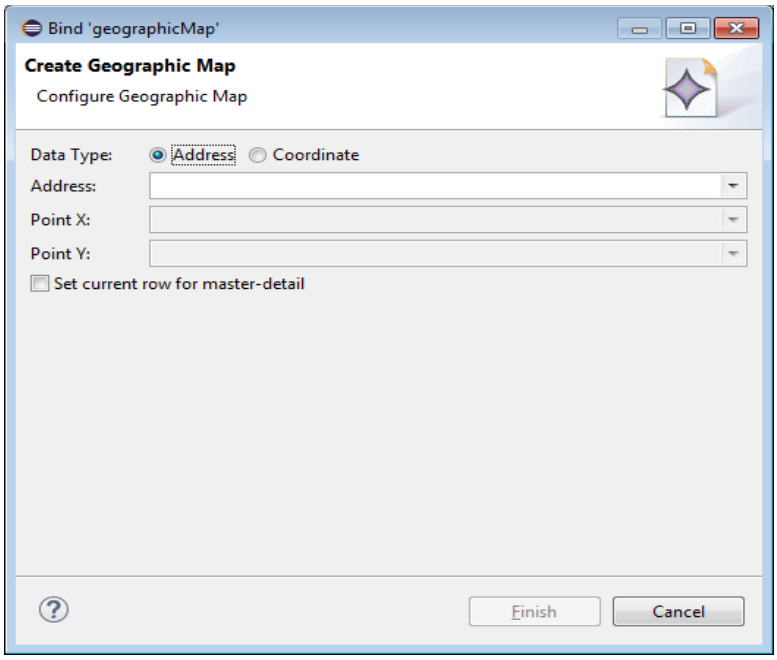


Figure 12–82 Creating Sunburst

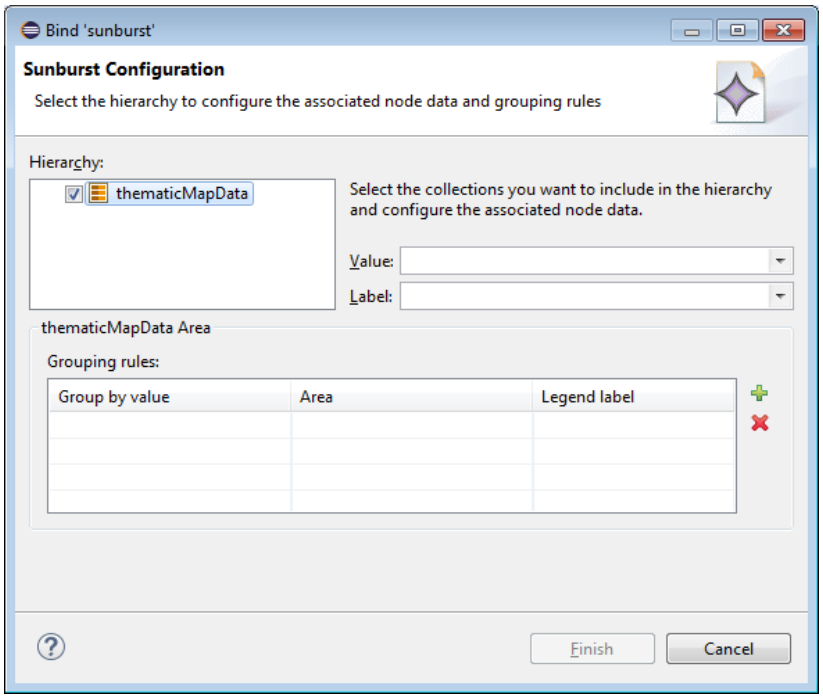


Figure 12–83 Creating Databound Timeline

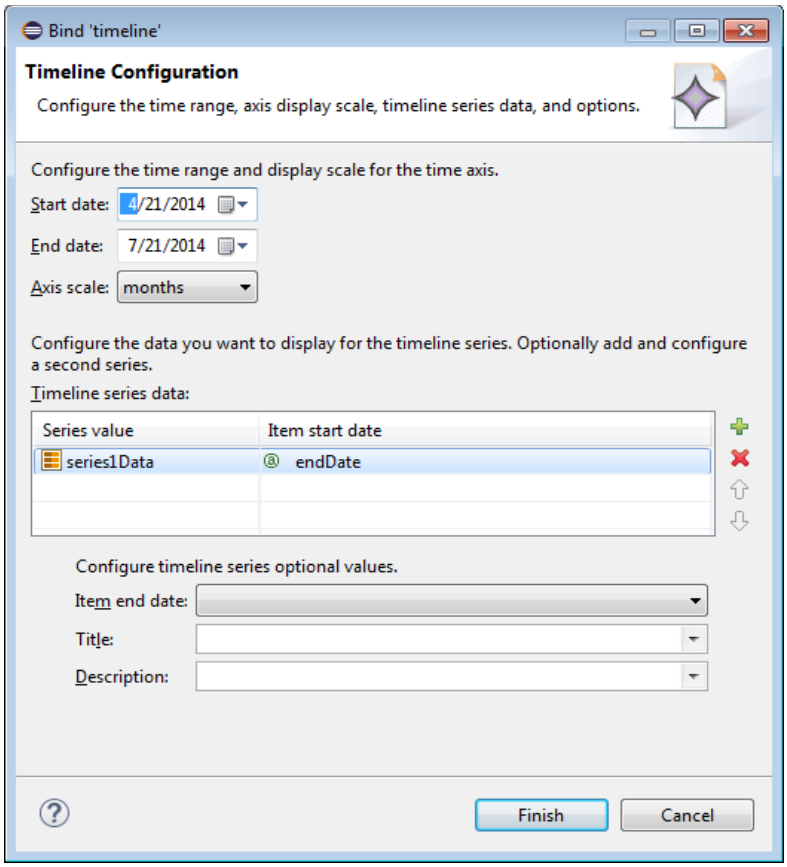


Figure 12–84 Creating Databound Treemap

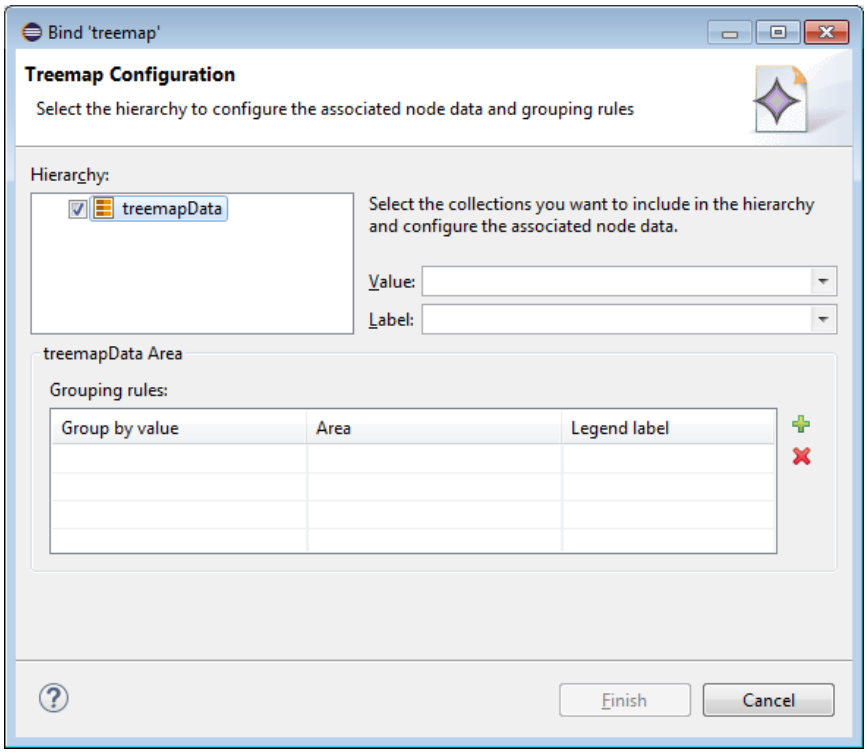
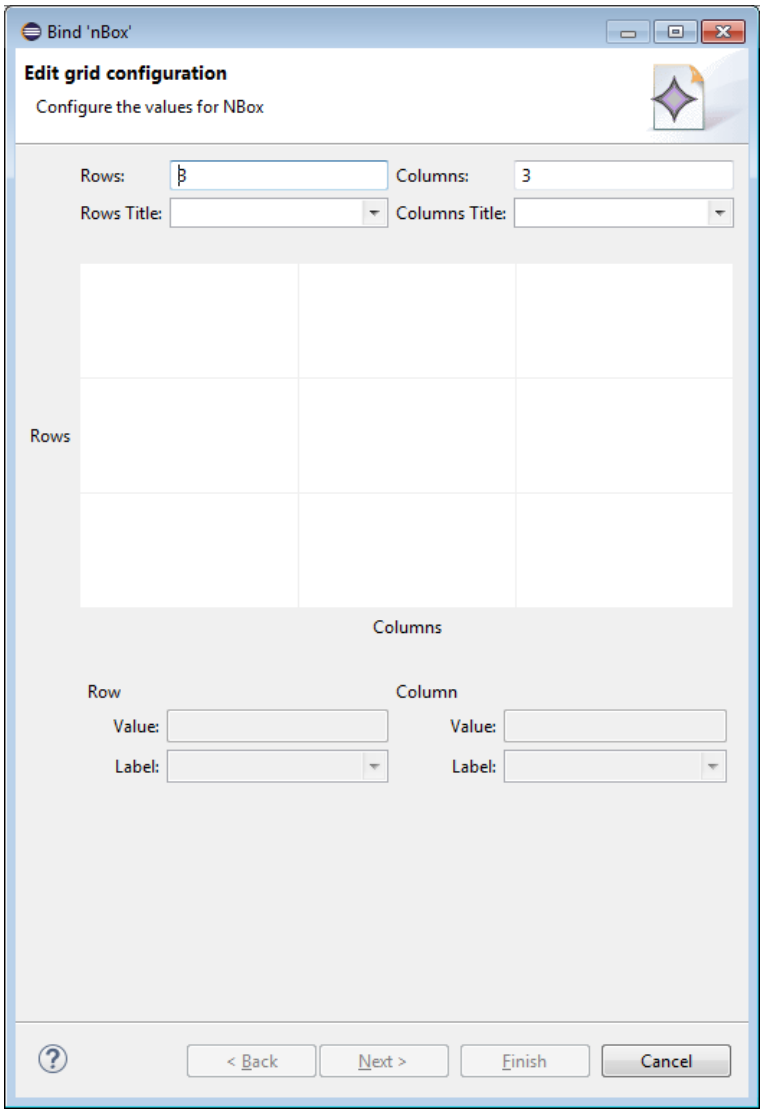
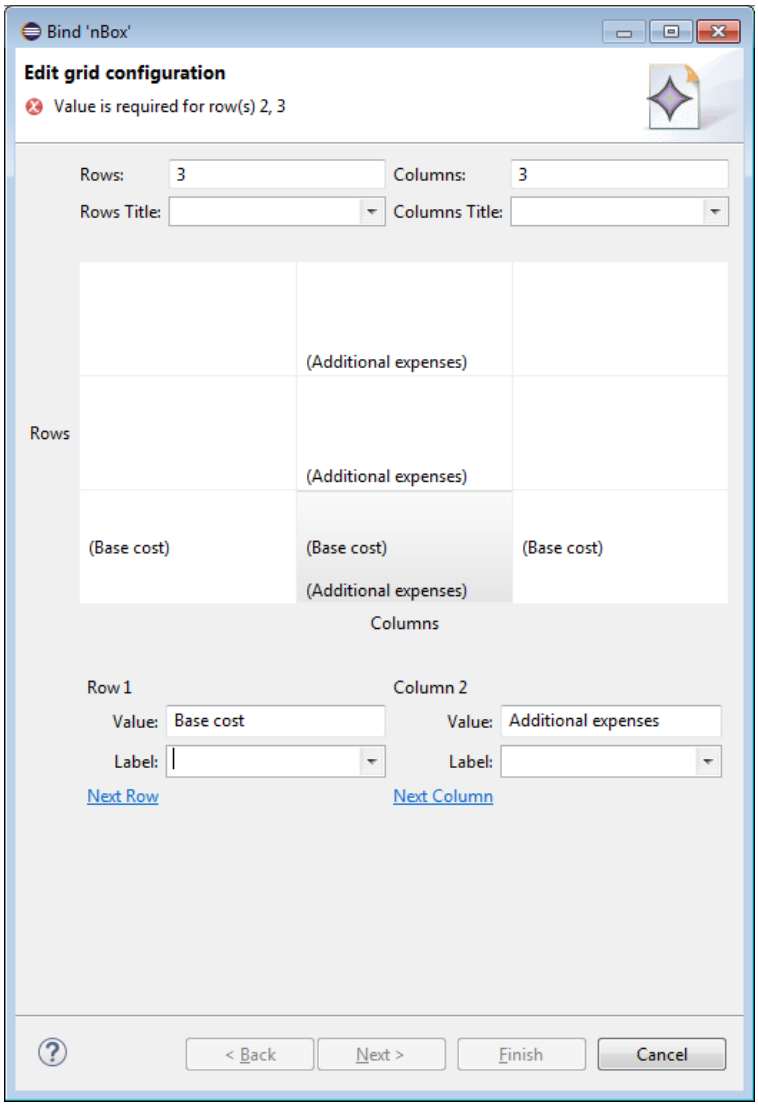


Figure 12–85 Creating Databound NBox



To complete the Create NBox dialog, you start by defining the number of rows and columns. Then you can select a cell on the box and specify values for the whole row or column in the bottom portion of the dialog, as Figure 12–86 shows.

Figure 12–86 Setting Row and Column Values for Databound NBox



The NBox component is created when you complete all pages of the series of dialogs by clicking **Next**.

12.5.32.1 What You May Need to Know About Setting Series Style for Databound Chart Components

When creating databound chart components from the Data Controls window, you can declaratively specify styling information for the chart series data by adding `seriesStyle` elements and then using the Properties window to open an editor for the `series` attribute of the `seriesStyle` element. Additionally, you can make the chart component clickable by using the `action` property of the `seriesStyle` element. This editor is already populated with the values of `series` attribute based on the values of the `chartDataItem` elements within the `dataStamp` facet.

12.5.33 How to Create Data Visualization Components Based on Static Data

For charts, as well as Treemap, Sunburst, and Timeline, you may choose not to specify their `var` and `value` attributes. Instead, you can define the component structure statically by enumerating elements that correspond to data items (for example,

charDataItem elements for charts, or timelineItems for Timeline Series). You can add as many of these static items as necessary, which is useful when you know the data at design time.

The following example shows a Pie Chart component that uses static data defined through its pieDataItem child components.

```
<dvtm:pieChart id="pieChart1" >
   <amx:facet name="dataStamp">
      <dvtm:pieDataItem id="di1" value="80000" label="Salary"/>
      <dvtm:pieDataItem id="di2" value="7500" label="Bonus"/>
      <dvtm:pieDataItem id="di3" value="12000" label="Commision"/>
   </amx:facet>
   <dvtm:legend position="none" id="l1"/>
</dvtm:pieChart>
```

The following example shows the pieDataItem child component whose value is specified based on an attribute binding instead of a collection.

```
<dvtm:pieDataItem id="di1" value="#{bindings.Salary.inputValue}" label="Salary"/>
```

12.5.34 How to Enable Interactivity in Chart Components

You can enable the end user interaction through tap with some chart components by defining event-driven triggers for the following child components of charts:

- Chart Data Item
- Pie Data Item
- Series Style

In addition to using the supported operations, such as Set Property Listener and Show Popup Behavior (see *Tag Reference for Oracle Mobile Application Framework* for more information), you can set the action attribute to define the type of action to be fired.

```
<amx:panelPage id="pp1" styleClass="dvtm-gallery-panelPage">
...
   <dvtm:lineChart id="lineChart1"
                   var="row"
                   value="#{bindings.lineData1.collectionModel}"
                   ... >
      <amx:facet name="dataStamp">
         <dvtm:chartDataItem group="#{row.group}"
                             value="#{row.value}"
                             series=" #{row.series}"
                             label="#{pageFlowScope.labelDisplay ?
                                      row.value : ''}" >
            <amx:showPopupBehavior popupId="pAdvancedOptions"
                                   type="action"
                                   align="overlapTopCenter"
                                   alignId="pflOptionsForm"
                                   decoration="anchor"/>
         </dvtm:chartDataItem>
      </amx:facet>
   ...
   </dvtm:lineChart>
   ...
</amx:panelPage>
<amx:popup id="pAdvancedOptions" styleClass="dvtm-gallery-options-dialog">
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.5.35 How to Create Polar Charts

You can enable the polar view for the following chart components by setting their `coordinateSystem` attribute to `polar`:

- Area chart
- Bar chart
- Bubble chart
- Combo chart
- Line chart
- Scatter chart

When the polar setting is applied to any of the previous chart types except Bar, you can define its polar grid as either circular or polygonal by using the `polarGridShape` attribute.

The polar chart's radial axis can be customized through its Y Axis Child component.

For more information, see*Tag Reference for Oracle Mobile Application Framework*.

12.6 Styling UI Components

MAF enables you to employ CSS to apply style to UI components.

12.6.1 How to Use Component Attributes to Define Style

You style your UI components by setting the following attributes:

- `styleClass` attribute defines a CSS style class to use for your layout component:

  ```
  <amx:panelPage styleClass="#{pageFlowScope.pStyleClass}">
  ```

  You can define the style class for layout, command, and input components in a MAF AMX page or in a skinning CSS file, in which case a certain style is applied to all components within the MAF AMX application feature (see Section 12.6.2, "What You May Need to Know About Skinning"). Alternatively, you can use the public style classes provided by MAF.

  > **Note:** The CSS file is not accessible from OEPE. Instead, MAF injects this file into the package at build or deploy time, upon which the CSS file appears in the `css` directory under the `ViewContent` root directory.

- `inlineStyle` attribute defines a CSS style to use for any UI component and represents a set of CSS styles that are applied to the root DOM element of the component:

  ```
  <amx:outputText inlineStyle="color:red;">
  ```

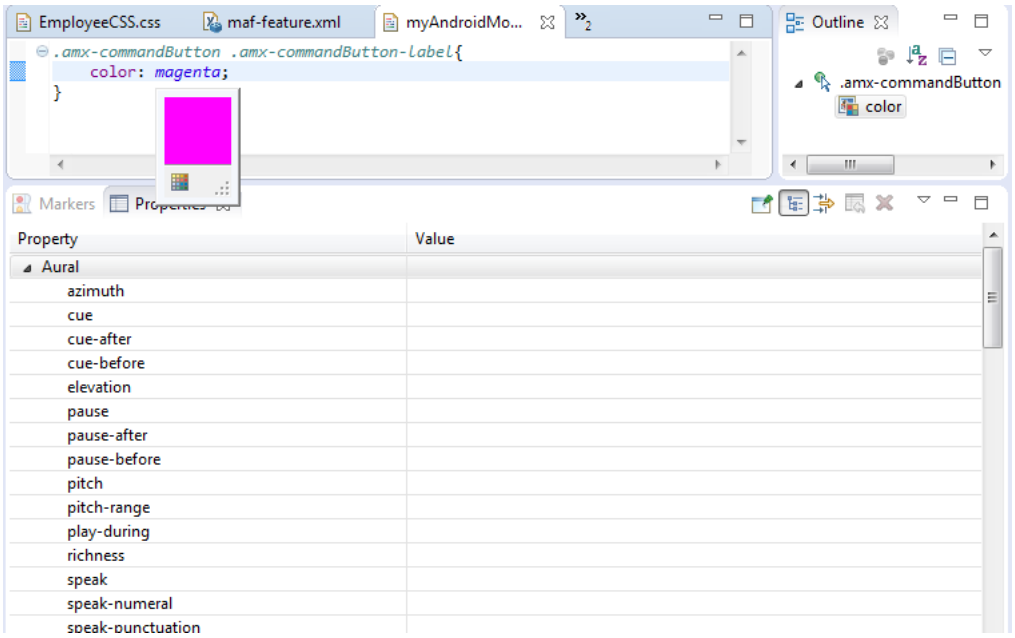
  You should use this attribute when basic style changes are required.

Note: Some UI components are rendered with such subelements as HTML `div` elements and more complex markup. As a result, setting the `inlineStyle` attribute on the parent component may not produce the desired effect. In such cases, you should examine the generated markup and, instead of defining the `inlineStyle` attribute, apply a CSS class that would propagate the style to the subelement.

For information on how to configure JavaScript debugging, see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript."

These attributes are displayed in the **Property** section in the Properties window, as Figure 12–87 shows. Use the **Value** column to specify the value of each property to define your style sheet.

Figure 12–87 Style Section of the Properties Window



For more information, see *Tag Reference for Oracle Mobile Application Framework*.

12.6.2 What You May Need to Know About Skinning

Skinning allows you to define and apply a uniform style to all UI components within a MAF AMX application feature to create a theme for the entire feature.

The default skin family for MAF is called `mobileAlta` and the default version is the latest version of that skin. For more information, see Chapter 7, "Skinning MAF Applications."

12.6.3 What You May Need to Know About Using CSS ID Selectors for Skinning

MAF AMX does not support the use of CSS ID selectors in skinning elements. As a result, a markup such as the following would cause rough MAF AMX page transitions:

```
#tbl {
   position:absolute;
```

```
   overflow:hidden;
   width: 300px;
   background-color: rgb(90,148,0);
}
```

The reason for this condition is that when a transition between MAF AMX pages occurs, two pages are rendered on the screen at the same time, and therefore, to prevent ID collisions, the page from which the transition occurs is stripped of all its IDs just before the transition.

Instead of using CSS ID selectors, you must use class names. The following example shows a MAF AMX UI component defined in a MAF AMX page, with its `styleClass` attribute set to a specific custom class.

```
<amx:panelPage styleClass="MySpecialClassName"/>
```

The following example show how to use the custom class for skinning.

```
.MySpecialClassName {
   height: 420px;
}
```

12.6.4 How to Style Data Visualization Components

Most of the style properties of MAF AMX data visualization components are defined in the `dvtm.css` file located in the `css` directory. You can override the default values by adding a custom CSS file with custom style definitions at the application feature level.

Some of the style properties cannot be mapped to CSS and have to be defined in custom JavaScript files. These properties include the following:

- Background and needle images for the Dial Gauge component (see Section 12.5.21, "How to Create a Dial Gauge").
- Duration bars color palette for the Timeline component (see Section 12.5.29, "How to Create a Timeline Component").
- Base maps for the Thematic Map component (see Section 12.5.25.5, "Defining a Custom Base Map").
- Style properties of the Geographic Map component (see Section 12.5.24, "How to Create a Geographic Map Component").
- Style properties of the Thematic Map component (see Section 12.5.25.7, "Applying Custom Styling to the Thematic Map Component").
- Selected and unselected states of the Rating Gauge component (see Section 12.5.22.2, "Applying Custom Styling to the Rating Gauge Component").

You should specify these custom JavaScript files in the Includes section at the application feature level. By doing so, you override the default style values defined in the XML style template. The following example shows a JavaScript file similar to the one you would add to your MAF project that includes the MAF AMX application feature with data visualization components which require custom styling of properties that cannot be styled using CSS.

```
my-custom.js:

   CustomChartStyle = {

      // common chart properties
      'chart': {
         // text to be displayed, if no data is provided
```

```
   'emptyText': null,
   // animation effect when the data changes
   'animationOnDataChange': "none",
   // animation effect when the chart is displayed
   'animationOnDisplay': "none",
   // time axis type - disabled, enabled, mixedFrequency
   'timeAxisType': "disabled"
},

// chart title separator properties
'titleSeparator': {
   // separator upper color
   'upperColor': "#74779A",
   // separator lower color
   'lowerColor': "#FFFFFF",
   // should display title separator
   'rendered': false
},

// chart legend properties
'legend': {
   // legend position - none, auto, start, end, top, bottom
   'position': "auto"
},

// default style values
'styleDefaults': {
   // default color palette
   'colors': ["#003366", "#CC3300", "#666699", "#006666", "#FF9900",
              "#993366", "#99CC33", "#624390", "#669933", "#FFCC33",
              "#006699", "#EBEA79"],
   // default shapes palette
   'shapes': ["circle", "square", "plus", "diamond",
              "triangleUp", "triangleDown", "human"],
   // series effect
   'seriesEffect': "gradient",
   // animation duration in ms
   'animationDuration': 1000,
   // animation indicators - all, none
   'animationIndicators': "all",
   // animation up color
   'animationUpColor': "#0099FF",
   // animation down color
   'animationDownColor': "#FF3300",
   // default line width for line chart
   'lineWidth': 3,
   // default line style for line chart - solid, dotted, dashed
   'lineStyle': "solid",
   // should markers be displayed for line and area charts
   'markerDisplayed': false,
   // default marker color
   'markerColor': null,
   // default marker shape
   'markerShape': "auto",
   // default marker size
   'markerSize': 8,
   // pie feeler color for pie chart
   'pieFeelerColor': "#BAC5D6",
   // slice label position and text type for pie chart
   'sliceLabel': {
```

```
            'position': "outside",
            'textType': "percent" }
      }
   };

   CustomGaugeStyle = {
      // default animation duration in milliseconds
      'animationDuration': 1000,
      // default animation effect on data change
      'animationOnDataChange': "none",
      // default animation effect on gauge display
      'animationOnDisplay': "none",
      // default visual effect
      'visualEffects': "auto"
   };

   CustomTimelineStyle = {
      'timelineSeries' : {
      // duration bars color palette
      'colors' : ["#267db3", "#68c182", "#fad55c", "#ed6647"]
   };
...
}
```

After the JavaScript file has been defined, you can uncomment and modify any values. You add this file as an included feature in the `maf-feature.xml` file, as the example below shows.

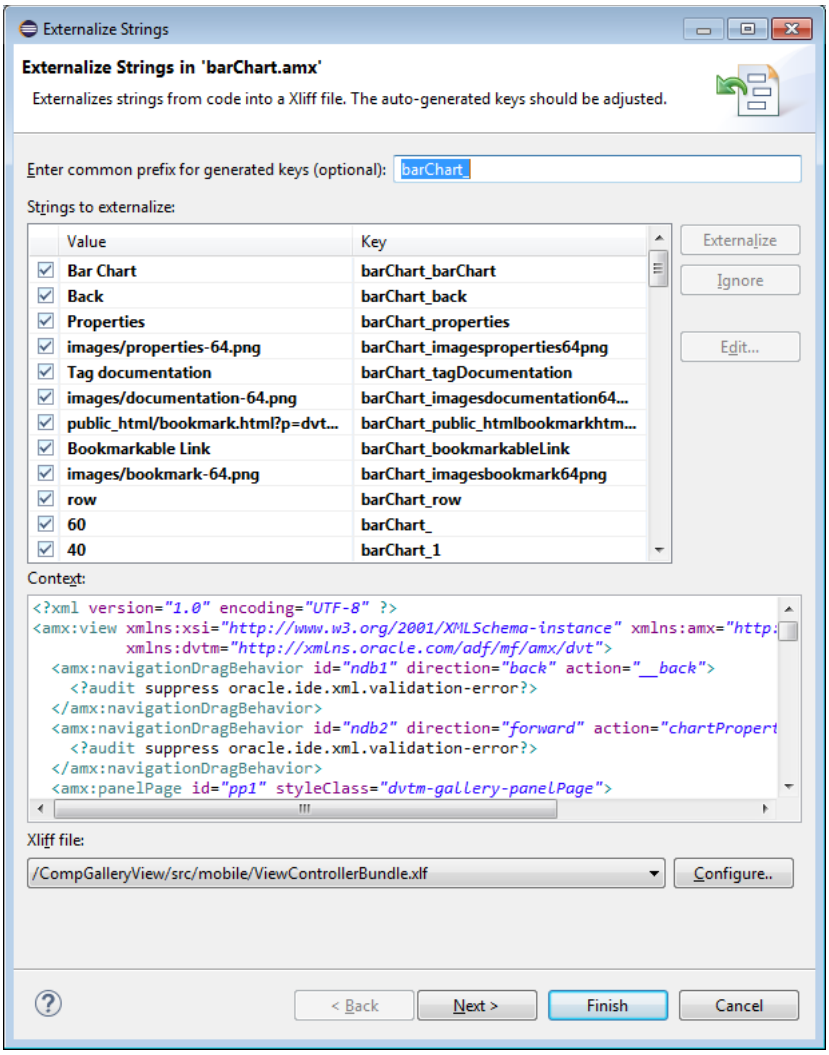
```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
   <adfmf:feature id="feature1" name="feature1">
      <adfmf:content id="feature1.1">
         <adfmf:amx file="feature1/untitled1.amx">
            <adfmf:includes>
               <adfmf:include type="StyleSheet" file="css/custom.css"/>
               <adfmf:include type="JavaScript" file="feature1/js/my-custom.js"/>
            </adfmf:includes>
         </adfmf:amx>
      </adfmf:content>
   </adfmf:feature>
</adfmf:features>
```

12.7 Localizing UI Components

In your MAF AMX page, you can localize the text that UI components display by using the standard XLIFF provided by OEPE. You do so by selecting an `.amx` file, right-clicking on a string, and then selecting **Source > Externalize Strings** in the context menu. This displays the Externalize Strings dialog (see Figure 12–88).

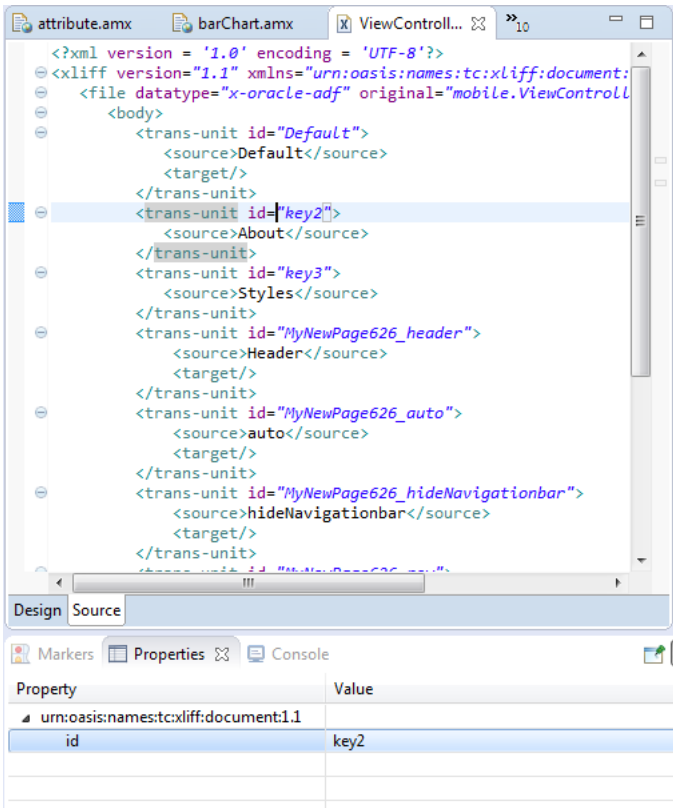
> **Note:** You can also externalize a file by right-clicking the file in the Project Explorer and selecting **Source > Externalize Strings**.

Figure 12–88 Externalize Strings dialog



Select the Value/Key pair you wish to externalize, then select **Edit** to change the value. When you have edited all the desired pairs, click **Finish** to create the XLIFF file. The default name of the XLIFF file is shown; to change it, click on **Configure**. Once you have created the XLIFF file, you can edit it, as shown in Figure 12–89, to change the values of the externalized strings.

Figure 12–89 Select Text Resource Dialog



An XLIFF (XML Localization Interchange File Format) file is created, if it is not already present. If it is present, the new entry is added to the existing XLIFF file. In addition, a corresponding Load Bundle (`loadBundle`) component is created as a child of the View component that points to the `ViewControllerBundle.xlf` file (default name, but basically it would match the name of the project).

Note: The `ViewControllerBundle.xlf` is a default file name. This name matches the name of the project.

Figure 12–90 shows the changes in the MAF AMX file.

Figure 12–90 Localized String in MAF AMX File

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/jdev/amx"
          xmlns:dvtm="http://xmlns.oracle.com/jdev/dvtm">
  <amx:panelPage id="pp1">
    <amx:facet name="header">
      <amx:outputText styleClass="amx-header-title"
                      value="#{viewcontrollerBundle.PRODUITS}"
                      id="outputText1"/>
    </amx:facet>
    <amx:listView value="#{ProductListBean.products}"
                  var="row"
                  id="listView1">
      <amx:listItem action="details" id="listItem1">
        <amx:outputText value="#{row.name}++"
                        id="outputText2"/>
        <amx:setPropertyListener from="#{row}"
                                 to="#{pageFlowScope.product}"
                                 type="action"/>
      </amx:listItem>
    </amx:listView>
    <amx:facet name="footer">
      <amx:outputText styleClass="ui-title"
                      value="Acme Inc"
                      id="outputText3"/>
    </amx:facet>
  </amx:panelPage>
  <amx:loadBundle basename="mobile.ViewControllerBundle"
                  var="viewcontrollerBundle"/>
</amx:view>
```

For more information, see Chapter 6, "Localizing MAF Applications."

12.8 Understanding MAF Support for Accessibility

When developing MAF applications, you may need to accommodate visually and physically impaired users by addressing accessibility issues. User agents, such as web browsers rendering to nonvisual media (for example, a screen reader) can read text descriptions of UI components to provide useful information to impaired users. MAF AMX UI and data visualization components are designed to be compliant with the following accessibility standards:

- The Accessible Rich Internet Applications (WAI-ARIA) 1.0 specification.

  For more information, see the following:

  - "Introduction" to WAI-ARIA 1.0 specification at `http://www.w3.org/TR/wai-aria/introduction`
  - "Using WAI-ARIA" at `http://www.w3.org/TR/wai-aria/usage`
  - Section 12.8.2, "What You May Need to Know About the Basic WAI-ARIA Terms"

- The Oracle Global HTML Accessibility Guidelines (OGHAG).

  For more information, see Section 12.8.3, "What You May Need to Know About the Oracle Global HTML Accessibility Guidelines."

- iOS Accessibility guidelines.

  For more information, see the *Accessibility Programming Guide for iOS*.

Accessible components do not change their appearance nor is the application logic affected by the introduction of such components.

To enable the proper functioning of the accessibility in your MAF AMX application feature, follow these guidelines:

- The navigation must not be more than three levels deep and it must be easy for the user to traverse back to the home screen.
- Keep scripting to a minimum.
- Do not provide direct interaction with the DOM.
- Do not use JavaScript time-outs.
- Avoid unnecessary focus changes
- Provide explicit popup triggers
- If needed, utilize the WAI-ARIA live region (see Section 12.8.2, "What You May Need to Know About the Basic WAI-ARIA Terms").
- Keep CSS use to a minimum.
- Try not to override the default component appearance.
- Choose scalable size units.
- Do not use CSS positioning.

For more information, see the following:

- "Mobile Accessibility" at `http://www.w3.org/WAI/mobile`
- "Web Content Accessibility and Mobile Web: Making a Web Site Accessible Both for People with Disabilities and for Mobile Devices" at `http://www.w3.org/WAI/mobile/overlap.html`

12.8.1 How to Configure UI and Data Visualization Components for Accessibility

MAF AMX UI and data visualization components have a built-in accessibility support, with most components being subject to the accessibility audit (see Figure 12–91).

Table 12–11 lists components and their attributes that you can set through the Accessibility section of the Properties window.

Table 12–11 UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Button (`commandButton`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Select Button (`selectOneButton`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Link (`commandLink`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Link Go (`goLink`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Carousel (`carousel`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
CarouselItem (`carouselItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.

Table 12–11 (Cont.) UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
List Item (`listItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Popup (`popup`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Image (`image`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be specified. If the image is used for decorative purposes, it can be empty.
Input Text (`inputText`)	Hint Text (`hintText`)	The `hintText` attribute should be present and describe what the field should contain.
Panel Group Layout (`panelGroupLayout`)	Landmark (`landmark`)	NA [1]
Deck (`deck`)	Landmark (`landmark`)	NA [1]
Flex Layout (`flexLayout`)	Landmark (`landmark`)	NA [1]
Table Layout (`tableLayout`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Cell Format (`cellFormat`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Film Strip (`filmStrip`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Film Strip Item (`filmStripItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Area Chart (`areaChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Bar Chart (`barChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Bubble Chart (`bubbleChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Combo Chart (`comboChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Line Chart (`lineChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Scatter Chart (`scatterChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Spark Chart (`sparkChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.

Table 12–11 (Cont.) UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Pie Chart (`pieChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
NBox Node (`nBoxNode`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Reference Object (`referenceObject`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Reference Area (`referenceArea`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Reference Line (`referenceLine`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Chart Data Item (`chartDataItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Pie Data Item (`pieDataItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Funnel Data Item (`funnelDataItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Area (`area`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Marker (`marker`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Threshold (`threshold`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Treemap Node (`treemapNode`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Sunburst Node (`sunburstNode`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Led Gauge (`ledGauge`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Status Meter Gauge (`statusMeterGauge`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Dial Gauge (`dialGauge`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Rating Gauge (`ratingGauge`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.

Table 12–11 (Cont.) UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Geographic Map (`geographicMap`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Thematic Map (`thematicMap`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Funnel Chart (`funnelChart`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
NBox (`nBox`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Sunburst (`sunburst`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Timeline (`timeline`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Treemap (`treemap`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.

[1] The landmark attribute has a default value (none) and is not subject to the accessibility audit.

You use the `shortDesc` attribute for different purposes for different components. For example, if you set the `shortDesc` attribute for the Image component, in the MAF AMX file it will appear as a value of the `alt` attribute of the `image` element.

The value of the `shortDesc` attribute can be localized.

For the Panel Group Layout and Deck components, you define the landmark role type (see Table 12–16, " Landmark Roles") that is applicable as per the context of the page. You can set one of the following values for the `landmark` attribute:

- default (none)
- application
- banner
- complementary
- contentinfo
- form
- main
- navigation
- search

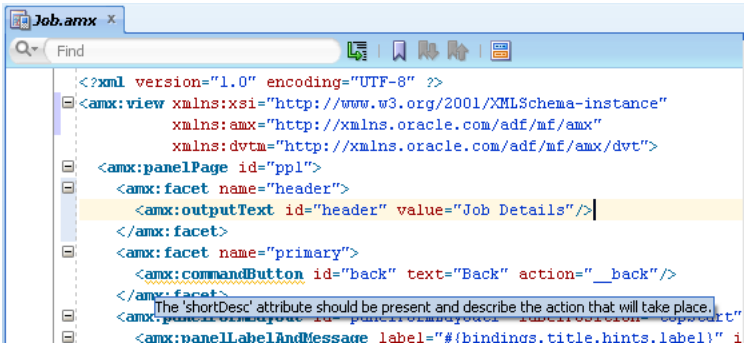
Table 12–12 lists UI components whose accessible attributes defined by WAI-ARIA specification are automatically applied at run time and that you cannot modify.

Table 12–12 UI Components with Static Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Input Date (`inputDate`)	Label (`label`)	`inputDate` is not labeled. The label attribute of `inputDate` should be specified.
Input Number Slider (`inputNumberSlider`)	Label (`label`)	`inputNumberSlider` is not labeled. The label attribute of `inputNumberSlider` should be specified.
Panel Label and Message (`panelLabelAndMessage`)	Label (`label`)	`panelLabelAndMessage` is not labeled. The label attribute of `panelLabelAndMessage` should be specified.
Select Item (`selectItem`)	Label (`label`)	`selectItem` is not labeled. The label attribute of `selectItem` should be specified.
Checkbox (`selectBooleanCheckbox`)	Label (`label`)	`selectBooleanCheckbox` is not labeled. The label attribute of `selectBooleanCheckbox` should be specified.
Boolean Switch (`selectBooleanSwitch`)	Label (`label`)	`selectBooleanSwitch` is not labeled. The label attribute of `selectBooleanSwitch` should be specified.
Radio Button (`selectOneRadio`)	Label (`label`)	`selectOneRadio` is not labeled. The label attribute of `selectOneRadio` should be specified.
Select Many Checkbox (`selectManyCheckbox`)	Label (`label`)	`selectManyCheckbox` is not labeled. The label attribute of `selectManyCheckbox` should be specified.
Select Many Choice (`selectManyChoice`)	Label (`label`)	`selectManyChoice` is not labeled. The label attribute of `selectManyChoice` should be specified.
Choice (`selectOneChoice`)	Label (`label`)	`selectOneChoice` is not labeled. The label attribute of `selectOneChoice` should be specified.
Output Text (`outputText`)	Value (`value`)	NA [1]

Figure 12–91 shows the accessibility audit warning displayed in OEPE.

Figure 12–91 Accessibility Audit Warning



For information on how to test your accessible MAF AMX application feature, see Section 30.2.1, "How to Perform Accessibility Testing on iOS-Powered Devices."

> **Note:** WAI-ARIA accessibility functionality is not supported on Android for data visualization components.
>
> Other MAF AMX UI components might not perform as expected when the application is run in the Android screen reader mode.

12.8.2 What You May Need to Know About the Basic WAI-ARIA Terms

As stated in the WAI-ARIA 1.0 specification, complex web applications become inaccessible when assistive technologies cannot determine the semantics behind portions of a document or when the user is unable to effectively navigate to all parts of it in a usable way. WAI-ARIA divides the semantics into roles (the type defining a user interface element), and states and properties supported by the roles. The following semantic associations form the base for the WAI-ARIA terms:

- Role
- Landmark
- Live region

For more information, see "Important Terms" at `http://www.w3.org/TR/wai-aria/terms`.

The following tables list role categories (as defined in the WAI-ARIA 1.0 specification) that are applicable to MAF.

Table 12–13 lists abstract roles that are used to support the WAI-ARIA role taxonomy for the purpose of defining general role concepts.

Table 12–13 Abstract Roles

Abstract Role	Description
input	A generic type of widget that allows the user input.
landmark	A region of the page intended as a navigational landmark.
select	A form widget that allows the user to make selections from a set of choices.
widget	An interactive component of a graphical user interface.

Table 12–14 lists widget roles that act as standalone user interface widgets or as part of larger, composite widgets.

Table 12–14 Widget Roles

Widget Role	Description	Widget Required States
alertdialog	A type of dialog that contains an alert message, where initial focus moves to an element within the dialog.	aria-labelledby, aria-describedby
button	An input that allows for user-triggered actions when clicked or pressed.	aria-expanded (state), aria-pressed (state)
checkbox	A checkable input that has three possible values: `true`, `false`, or `mixed`.	aria-checked (state)
dialog	A dialog represented by an application window that is designed to interrupt the current processing of an application in order to prompt the user to enter information or require a response.	aria-labelledby, aria-describedby
link	An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource.	aria-disabled (state), aria-describedby
option	A selectable item in a select list.	aria-labelledby, aria-checked (state), aria-selected (state)
radio	A checkable input in a group of radio roles, only one of which can be checked at a time.	aria-checked (state), aria-disabled (state)
slider	A user input where the user selects a value from within a given range.	aria-valuemax, aria-valuemin, aria-valuenow, aria-disabled (state)
listbox	A widget that allows the user to select one or more items from a list of choices.	aria-live
radiogroup	A group of radio buttons.	aria-disabled (state)
listitem	A single item in a list or directory.	aria-describedby
textbox	Input that allows free-form text as its value.	aria-labelledby, aria-readonly, aria-required, aria-multiline, aria-disabled (state)

Table 12–15 lists document structure roles that describe structures that organize content in a page. Typically, document structures are not interactive.

Table 12–15 Document Structure Roles

Document Structure Role	Description
img	A container for a collection of elements that form an image.
list	A group of non-interactive list items.
listitem	A single item in a list or directory.

Table 12–16 lists landmark roles that represent regions of the page intended as navigational landmarks.

Table 12–16 Landmark Roles

Landmark Role	Description
application	A region declared as a web application (as opposed to a web document).
banner	A region that contains mostly site-oriented content (rather than page-specific content).
complementary	A supporting section of a document designed to be complementary to the main content at a similar level in the DOM hierarchy, but that remains meaningful when separated from the main content.
contentinfo	A large perceivable region that contains information about the parent document.
form	A region that contains a collection of items and objects that, as a whole, combine to create a form.
main	The main content of a document.
navigation	A collection of navigational elements (usually links) for navigating the document or related documents.
search	A region that contains a collection of items and objects that, as a whole, combine to create a search facility.

For the majority of MAF UI components, you cannot modify accessible WAI-ARIA attributes. For some components, you can set special accessible attributes at design time, and for the Panel Group Layout and Deck, you can use the WAI-ARIA landmark role type. For more information, see Section 12.8.1, "How to Configure UI and Data Visualization Components for Accessibility."

12.8.3 What You May Need to Know About the Oracle Global HTML Accessibility Guidelines

The Oracle Global HTML Accessibility Guidelines (OGHAG) is a set of scripting standards for HTML that Oracle follows. These standards represent a combination of Section 508 (see `http://www.section508.gov`) and Web Content Accessibility Guidelines (WCAG) 1.0 level AA (see `http://www.w3.org/TR/WCAG10`), with improved wording and checkpoint measurements.

For more information, see *Oracle's Accessibility Philosophy and Policies* at `http://www.oracle.com/us/corporate/accessibility/policies/index.html`.

12.9 Validating Input

MAF allows you to inform the end user about data input errors and other conditions that occur during data input. Depending on their type (error or warning), validation messages have a different look and feel.

The user input validation is triggered when an input is submitted: Input Text components are automatically validated when the end user leaves the field; for selection components, such as a Checkbox or Choice, the validation occurs when the end user makes a selection. For validation purposes, UI components on a MAF AMX page are grouped together within a Validation Group operation (`validationGroup`) to define components whose input is to be validated when the submit operation takes place. A Validation Behavior (`validationBehavior`) component defines which Validation Group is to be validated before a command component's action is taken. A command component can have multiple child Validation Behavior components.

Validation does not occur if a component does not have a Validation Behavior defined for it.

Note: You cannot define nested Validation Group operations.

The following is an invalid definition of a Validation Group:

```
<amx:view>
   <amx:panelPage>
      <amx:validationGroup>
         <amx:panelGroupLayout>
            <amx:validationGroup/>
         <amx:panelGroupLayout/>
      </amx:validationGroup>
   </amx:panelPage>
</amx:view>
```

The following is a valid definition:

```
<amx:view>
   <amx:panelPage>
      <amx:validationGroup>
   </amx:panelPage>
   <amx:popup>
      <amx:validationGroup>
   </amx:popup>
</amx:view>
```

If a MAF AMX page contains any validation error messages, you can use command components, such as List Item, Link, and Button, to prevent the end user from navigating off the page. Messages containing warnings do not halt the navigation.

The example below shows how to define validation elements, including multiple Validation Group and Validation Behavior operations, in a MAF AMX file.

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="outputText1" value="Validate"/>
   </amx:facet>
   <amx:facet name="secondary">
      <amx:commandButton id="commandButton2" action="go" text="Save">
         <amx:validationBehavior id="vb1"
                                 disabled="#{pageFlowScope.myPanel ne 'panel1'}"
                                 group="group1"/>
         <amx:validationBehavior id="vb2"
                                 disabled="#{pageFlowScope.myPanel ne 'panel2'}"
                                 group="group2"/>
         <!-- invalid, should be caught by audit rule but for any reason
         if group not found at run time, this validate is ignored -->
         <amx:validationBehavior id="vb3" disabled="false" group="groupxxx"/>
         <!-- group is not found at run time, this validate is ignored -->
         <amx:validationBehavior id="vb4" disabled="false" group="group3"/>
      </amx:commandButton>
   </amx:facet>
   <amx:panelSplitter id="ps1" selectedItem="#{pageFlowScope.myPanel}">
      <amx:panelItem id="pi1">
         <amx:validationGroup id="group1">
            <amx:panelFormLayout id="pfl1">
               <amx:inputText value="#{bindings.first.inputValue}"
                              required="true"
```

```
                              label="#{bindings.first.hints.label}"
                              id="inputText1"/>
                  <amx:inputText value="#{bindings.last.inputValue}"
                              label="#{bindings.last.hints.label}"
                              id="inputText2"/>
               </amx:panelFormLayout>
            </amx:validationGroup>
         </amx:panelItem>
         <amx:panelItem id="pi2">
            <amx:validationGroup id="group2">
               <amx:panelFormLayout id="pfl2">
                  <amx:inputText value="#{bindings.salary.inputValue}"
                              label="#{bindings.first.hints.label}"
                              id="inputText3"/>
                  <amx:inputText value="#{bindings.last.inputValue}"
                              label="#{bindings.last.hints.label}"
                              id="inputText4"/>
               </amx:panelFormLayout>
            </amx:validationGroup>
         </amx:panelItem>
      </amx:panelSplitter>
      <amx:panelGroupLayout id="pgl1" rendered="false">
         <amx:validationGroup id="group3">
            <amx:panelFormLayout id="pfl4">
               <amx:inputText value="#{bindings.salary.inputValue}"
                           label="#{bindings.first.hints.label}"
                           id="inputText5"/>
               <amx:inputText value="#{bindings.last.inputValue}"
                           label="#{bindings.last.hints.label}"
                           id="inputText6"/>
            </amx:panelFormLayout>
         </amx:validationGroup>
      </amx:panelGroupLayout>
   </amx:panelPage>
```

This example shows how to define a validation message displayed in a popup in a MAF AMX file.

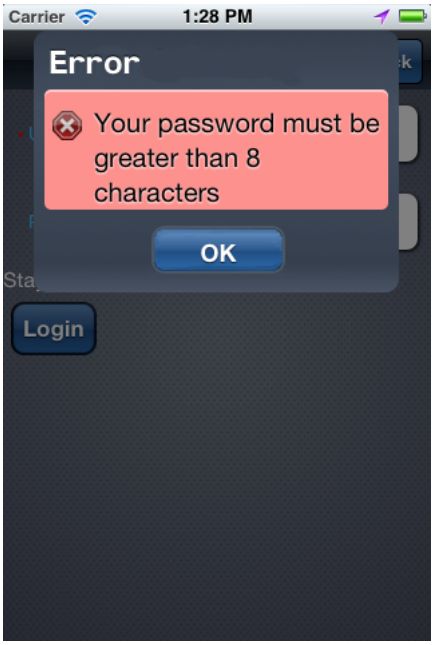
```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="outputText1" value="Login Demo"/>
   </amx:facet>
   <amx:facet name="secondary">
      <amx:commandButton id="btnBack" action="__back" text="Back"/>
   </amx:facet>
   <amx:panelGroupLayout id="panelGroupLayout1">
      <amx:validationGroup id="group1">
         <amx:panelGroupLayout id="panelGroupLayout2">
            <amx:inputText value="#{bindings.userName.inputValue}"
                           label="#{bindings.userName.hints.label}"
                           id="inputText1"
                           showRequired="true"
                           required="true"/>
            <amx:inputText value="#{bindings.password.inputValue}"
                           label="#{bindings.password.hints.label}"
                           id="inputText2"
                           required="true"
                           showRequired="true"
                           secret="true"/>
            <amx:outputText id="outputText2"
                            value="#{bindings.timeToStayLoggedIn.hints.label}:
```

```
                          #{bindings.timeToStayLoggedIn.inputValue} minutes"/>
        </amx:panelGroupLayout>
      </amx:validationGroup>
      <amx:commandButton id="commandButton2"
                         text="Login"
                         action="navigationSuccess">
        <amx:validationBehavior id="validationBehavior2" group="group1"/>
      </amx:commandButton>
    </amx:panelGroupLayout>
</amx:panelPage>
```

Validation messages are displayed in a Popup component (see Section 12.2.8, "How to Use a Popup Component"). You cannot configure the title of a validation popup, which is automatically determined by the relative message severity: the most severe of all of the current messages becomes the title of the validation popup. That is, if all validation messages are of type `WARNING`, then the title is "Warning"; if some of the messages are of type `WARNING` and others are of type `ERROR`, then the title is set to "Error".

Figure 12–92 shows a popup validation message produced at runtime.

Figure 12–92 Validation Message on iPhone



12.10 Using Event Listeners

To invoke Java code from your MAF AMX pages and perform the application logic, you define listeners as attributes of UI components in one of the following ways:

- Manually in the source of your MAF AMX file.
- From the **Property** editor of the selected component. For more information, see *Tag Reference for Oracle Mobile Application Framework*

You may use the following listeners to add awareness of the UI-triggered events to your MAF AMX page:

- `valueChangeListener`: listens to `ValueChangeEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing an old value
  - `java.lang.Object` representing a new changed value
- `actionListener`: listens to `ActionEvent` that is constructed without parameters;
- `selectionListener`: listens to `SelectionEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing an old row key
  - `java.lang.String[]` representing selected row keys
- `moveListener`: listens to `MoveEvent` that is constructed with the following parameters: of the `RowKey` type representing an old row key;
  - `java.lang.Object` representing the moved row key
  - `java.lang.String[]` representing the row key before which the moved row key was inserted
- `rangeChangeListener`: listens to `RangeChangeEvent` that is constructed without parameters.
- `mapBoundsChangeListener`: listens to `MapBoundsChangeEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing the X coordinate (longitude) of minimum map bounds
  - `java.lang.Object` representing the Y coordinate (latitude) of minimum map bounds
  - `java.lang.Object` representing the X coordinate (longitude) of maximum map bounds
  - `java.lang.Object` representing the Y coordinate (latitude) of maximum map bounds
  - `java.lang.Object` representing the X coordinate (longitude) of the map center
  - `java.lang.Object` representing the Y coordinate (latitude) of the map center
  - `int` representing the current zoom level
- `mapInputListener`: listens to `MapInputEvent` that is constructed with the following parameters:
  - `java.lang.String` representing the event type
  - `java.lang.Object` representing the X coordinate of the event point
  - `java.lang.Object` representing the Y coordinate of the event point
- `viewportChangeListener`: listens to `ViewportChangeEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing the minimum X coordinate
  - `java.lang.Object` representing the maximum X coordinate
  - `java.lang.Object` representing the minimum Y coordinate
  - `java.lang.Object` representing the maximum Y coordinate
  - `java.lang.Object` representing the first visible group

  - `java.lang.Object` representing the last visible group
- `drillListener`: listens to `DrillEvent` that is constructed with the following parameters::
  - `java.lang.Object` representing the ID of the drilled object
  - `java.lang.Object` representing the rowkey of the drill data item
  - `java.lang.Object` representing the group name of the drilled object
  - `java.lang.Object` representing the series name of the drilled object

The value for your listener must match the pattern `#{*}` and conform to the following requirements:

- Type name: EL Expression
- Base type: string
- Primitive type: string

For information on EL events, see Section 13.3.6, "About EL Events."

Most MAF AMX event classes extend the `oracle.adfmf.amx.event.AMXEvent` class. When defining event listeners in your Java code, you need to pass the `oracle.adfmf.amx.event.AMXEvent` class.

For more information, see the following:

- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*
- Section 18.2.2, "How to Use AmxEvent Classes"

MAF allows you to create managed bean methods for listeners so that your managed bean methods use MAF AMX-specific event classes.The three examples below demonstrate a Button and a Link component calling the same managed bean method. The source value of the `AMXEvent` determines which object invoked the event by showing a message box with the component's ID.

This example demonstrates calling a bean method from a MAF AMX file.]]

```
<amx:commandButton text="commandButton1"
                   id="commandButton1"
                   actionListener="#{applicationScope.Bean.actionListenerMethod}">
</amx:commandButton>
<amx:commandLink text="commandLink1"
                 id="commandLink1"
                 actionListener="#{applicationScope.Bean.actionListenerMethod}">
</amx:commandLink>
```

This example demonstrates using an `AMXEvent`.]]

```
private void actionListenerMethod(AMXEvent amxEvent) {
   // Some Java handling
}
```

This example demonstrates invoking the event method.]]

```
public Object invokeMethod(String methodName, Object[] params) {
   if (methodName.equals("actionListenerMethod")) {
      actionListenerMethod((AMXEvent) params[0]);
   }
   return null;
}
```

For additional examples, see a MAF sample application called APIDemo application available from **File** > **New** > **MAF Examples**. This sample demonstrates how to call listeners from JavaBeans.

12.10.1 What You May Need to Know About Constrained Type Attributes for Event Listeners

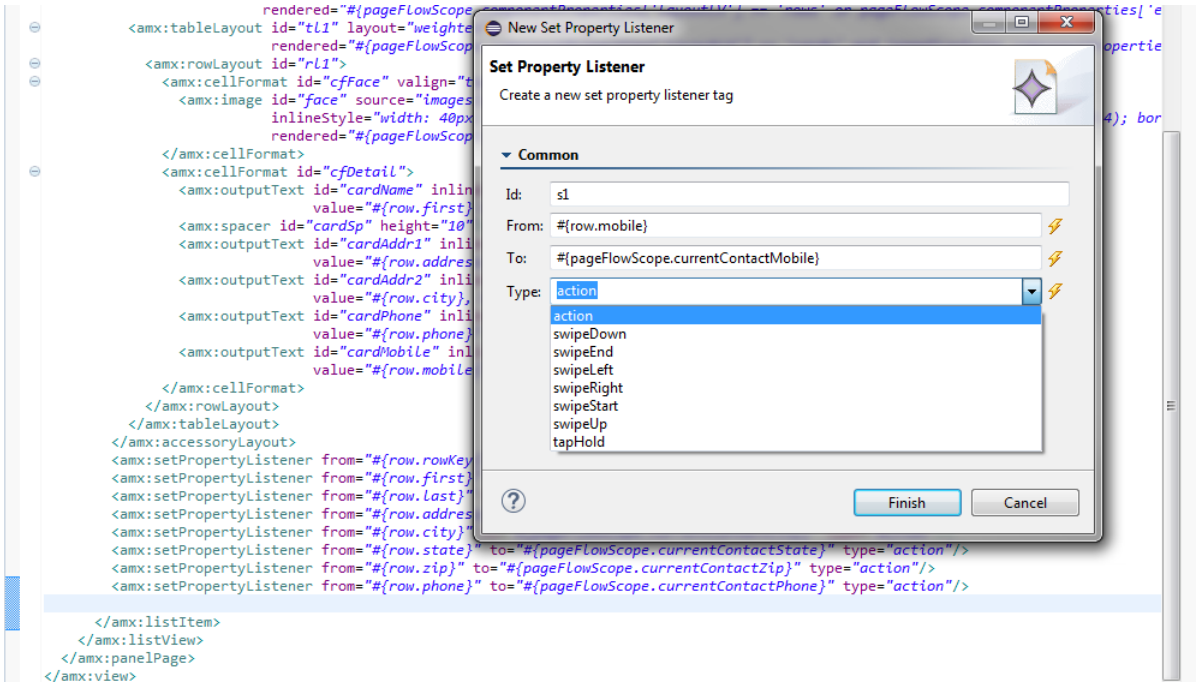
You can define event listeners as children of some MAF AMX UI components. The listeners' `type` attribute identifies which event they are to be registered to handle. Since each parent UI component supports only a subset of the events (suitable for that particular component), these supported events are presented in a constrained list of types that you can select for a listener.

Consult the *Tag Reference for Oracle Mobile Application Framework* for information about parent UI components, event listeners they can have as children, and event types they support.

The `type` attribute of each of the child event listeners has a base set of values that match the listener events. These values are filtered based on the information presented in *Tag Reference for Oracle Mobile Application Framework* such that when the child event listener is within the context of the identified parent UI component, only the events that the parent supports are shown. For example, under a Button component, the Action Listener or Set Property Listener child would show only the `action` Type value, as well as gestures.

Figure 12–93 shows values available in the constrained Type list of the Set Property Listener for a parent List Item component.

Figure 12–93 Selecting Event Type



13

Using Bindings and Creating Data Controls in MAF AMX

This chapter describes how to use data bindings, data controls, and the data binding expression language (EL) with the Mobile Application Framework (MAF). In addition, object scope lifecycles, managed beans, UI hints, validation, and data change events are also discussed.

This chapter includes the following sections:

- Section 13.1, "Introduction to Bindings and Data Controls"
- Section 13.2, "About Object Scope Lifecycles"
- Section 13.3, "Creating EL Expressions"
- Section 13.4, "Creating and Using Managed Beans"
- Section 13.5, "Exposing Business Services with Data Controls"
- Section 13.6, "Creating Databound UI Components from the Data Controls Palette"
- Section 13.7, "What Happens at Runtime: How the Binding Context Works"
- Section 13.8, "Working with Data Control Attributes"
- Section 13.9, "Creating and Using Bean Data Controls"
- Section 13.10, "Using the DeviceFeatures Data Control"
- Section 13.11, "Validating Attributes"
- Section 13.12, "About Data Change Events"

13.1 Introduction to Bindings and Data Controls

Mobile Application Framework implements two concepts that enable the decoupling of the user interface (UI) technology from the business service implementation: *data controls* and *declarative bindings*. Data controls abstract the implementation technology of a business service by using standard metadata interfaces to describe the service's operations and data collections, including information about the properties, methods, and types involved. Using OEPE, you can view that information as icons that you can drag and drop onto a page. Declarative bindings abstract the details of accessing data from data collections in a data control and invoking its operations. At runtime, the model layer reads the information describing the data controls and bindings from the appropriate XML files and then implements the two-way connection between the user interface and the business service.

The group of bindings supporting the user interface components on a page are described in a page-specific XML file called the page definition file. The model layer uses this file at runtime to instantiate the page's bindings. These bindings are held in a request-scoped map called the binding container, accessible during each page request using the EL expression `#{bindings}`. This expression always evaluates to the binding container for the current page. You can design a databound user interface by dragging an item from the Palette and dropping it on a page as a specific UI component. When you use data controls to create a UI component, OEPE automatically creates the code and objects needed to bind the component to the data control you selected.

The Mobile Application Framework comes with two out-of-the box data controls: the DeviceFeatures data control and the ApplicationFeatures data control. The DeviceFeatures data control appears within the Palette in OEPE, enabling you to drag and drop the primary data attributes of data controls to your application as (text) fields, and the operations of data controls as command objects (buttons). These drag and drop actions will generate EL bindings in your application and the appropriate properties for the controls that are created. The bindings are represented in page definition file, which points at the data control source, and the page bindings link the specific page's reference to the data control.

For more information about data controls and bindings, see the following:

- Section 13.5, "Exposing Business Services with Data Controls"
- Section 13.6, "Creating Databound UI Components from the Data Controls Palette"
- Section 13.7, "What Happens at Runtime: How the Binding Context Works"
- Section 13.9, "Creating and Using Bean Data Controls"
- Section 13.9, "Creating and Using Bean Data Controls"
- Section 13.10, "Using the DeviceFeatures Data Control"

13.2 About Object Scope Lifecycles

At runtime, you pass data to pages by storing the needed data in an object scope where the page can access it. The scope determines the lifespan of an object. Once you place an object in a scope, it can be accessed from the scope using an EL expression. For example, you might create a managed bean named `foo`, and define the bean to live in the view scope. To access that bean, you would use the expression `#{viewScope.foo}`.

Mobile Application Framework variables and managed bean references are defined within different object scopes that determine the variable's lifetime and visibility. MAF supports the following scopes, listed in order of decreasing visibility:

- Application scope—The object is available for the duration of the application (across features).
- Page flow scope—The object is available for the duration of a feature (single feature boundary).
- View scope—The object is available for the duration of the view (single page of a feature).

Object scopes are analogous to global and local variable scopes in programming languages. The wider the scope, the higher the availability of an object. During their lifespan, these objects may expose certain interfaces, hold information, or pass variables and parameters to other objects. For example, a managed bean defined in application scope will be available for use during multiple page requests for the

duration of the application. However, a managed bean defined in view scope will be available only for the duration of one page request within a feature.

EL expressions defined in the application scope namespace are available for the life of the application, across feature boundaries. You can define an application scope in one view of an application, and then reference it in another. EL expressions defined in the page flow scope namespace are available for the duration of a feature, within the bounds of a single feature. EL expressions defined in the view scope namespace are available for the duration of the view, within the bounds of a single page of a feature. In addition to these variable-containing scopes, MAF defines scopes that can expose information about device properties and application preferences. These scopes have application-level lifetime and visibility. For more information, see Section 13.3.5.2, "About the Managed Beans Category" and Section 13.3.5.3, "About the Mobile Application Framework Objects Category."

When determining what scope to register a managed bean with or to store a value in, always try to use the narrowest scope possible. Use the application scope only for information that is relevant to the whole application, such as user or context information. Avoid using the application scope to pass values from one page to another.

Note: Every object you put in a memory scope is serialized to a JSON `DataChangeEvent`, and objects returned by any getter method inside this object are also serialized. This can lead to deeply nested object trees that are serialized, which will decrease performance. To avoid serialization of a chain of nested objects, you should define them as transient. See Section 13.9.1, "What You May Need to Know About Serialization of Bean Class Variables" for more information.

13.2.1 What You May Need to Know About Object Scopes and Task Flows

When determining what scope to use for variables within a task flow, you should use only view or page flow scopes. The application scope will persist objects in memory beyond the life of the task flow and therefore compromise the encapsulation and reusable aspects of a task flow. In addition, application scope may keep objects in memory longer than needed, causing unneeded overhead.

When you need to pass data values between activities within a task flow, you should use page flow scope. View scope should be used for variables that are needed only within the current view activity, not across view activities.

13.3 Creating EL Expressions

You use EL expressions in MAF applications to bind attributes to object values determined at runtime. For example, `#{UserList.selectedUsers}` might reference a set of selected users, `#{user.name}` might reference a particular user's name, while `#{user.role == 'manager'}` would evaluate whether a user is a manager or not. At runtime, a generic expression evaluator returns the `List`, `String`, and `boolean` values of these respective expressions, automating access to the individual objects and their properties without requiring code.

Expressions are not evaluated until they are needed for rendering a value. Because MAF AMX supports only deferred evaluation, an expression using the immediate construction expression (`"${}"`) still parses, but behaves the same as a deferred expression (`"#{}"`). At runtime, the value of certain UI components (such as an `inputText` component or an `outputText` component) is determined by its `value`

attribute. While a component can have static text as its value, typically the `value` attribute will contain an EL expression that the runtime infrastructure evaluates to determine what data to display. For example, an `outputText` component that displays the name of the currently logged-in user might have its `value` attribute set to the expression `#{UserInfo.name}`. Since any attribute of a component (and not just the `value` attribute) can be assigned a value using an EL expression, it's easy to build dynamic, data-driven user interfaces. For example, you could hide a component when a set of objects you need to display is empty by using a boolean-valued expression like `#{not empty UserList.selectedUsers}` in the UI component's `rendered` attribute. If the list of selected users in the object named `UserList` is empty, the `rendered` attribute evaluates to `false` and the component disappears from the page.

In a typical application, you would create objects like `UserList` as a managed bean. The runtime manages instantiating these beans on demand when any EL expression references them for the first time. When displaying a value, the runtime evaluates the EL expression and pulls the value from the managed bean to populate the component with data when the page is displayed. If the user updates data in the UI component, the runtime pushes the value back into the corresponding managed bean based on the same EL expression. For more information about creating and using managed beans, see Section 13.4, "Creating and Using Managed Beans." For more information about EL expressions, see the Java EE tutorial at `http://www.oracle.com/technetwork/java/index.html`.

Note: When using an EL expression for the `value` attribute of an editable component, you must have a corresponding `set` method for that component, or else the EL expression will evaluate to read-only, and no updates to the value will be allowed.

For example, say you have an `inputText` component (whose ID is `it1`) on a page, and you have its value set to `#{myBean.inputValue}`. The `myBean` managed bean would have to have `get` and `set` methods as follows, in order for the `inputText` value to be updated:

```
public void setIt1(RichInputText it1) {
    this.it1 = it1;
}

public RichInputText getIt1() {
    return it1;
}
```

13.3.1 About Data Binding EL Expressions

When you use the Palette to create a component, the MAF data binding expressions are created for you. The expressions are added to every component attribute that will either display data from or reference properties of a binding object. Each prebuilt expression references the appropriate binding objects defined in the page definition file. You can edit these binding expressions or create your own, as long as you adhere to the basic MAF binding expression syntax. MAF data binding expressions can be added to any component attribute that you want to populate with data from a binding object, if the attribute supports EL.

A typical MAF data binding EL expression uses the following syntax to reference any of the different types of binding objects in the binding container:

`#{bindings.`*`BindingObject.propertyName`*`}`

where:

- `bindings` is a variable that identifies that the binding object being referenced by the expression is located in the binding container of the current page. All MAF data binding EL expressions must start with the `bindings` variable.
- *`BindingObject`* is the ID, or for attributes the name, of the binding object as it is defined in the page definition file. The binding `objectID` or name is unique to that page definition file. An EL expression can reference any binding object in the page definition file, including parameters, executables, or value bindings.
- *`propertyName`* is a variable that determines the default display characteristics of each databound UI component and sets properties for the binding object at runtime. There are different binding properties for each type of binding object. For more information about binding properties, see Section 13.3.3, "What You May Need to Know About MAF Binding Properties."

For example, in the following expression:

```
#{bindings.ProductName.inputValue}
```

the `bindings` variable references a bound value in the current page's binding container. The binding object being referenced is `ProductName`, which is an attribute binding object. The binding property is `inputValue`, which returns the value of the first `ProductName` attribute.

> **Tip:** While the binding expressions in the page definition file can use either a dollar sign (`$`) or hash sign (`#`) prefix, the EL expressions in MAF pages can only use the hash sign (`#`) prefix.

As stated previously, when you use the Palette to create UI components, these expressions are built for you. However, you can also manually create them if you need to. The OEPE Expression Builder is a dialog that helps you build EL expressions by providing lists of binding objects defined in the page definition files, as well as other valid objects to which a UI component may be bound. It is particularly useful when creating or editing MAF databound expressions because it provides a hierarchical list of MAF binding objects and their most commonly used properties. For information about binding properties, see Section 13.3.3, "What You May Need to Know About MAF Binding Properties."

13.3.2 How to Create an EL Expression

You can create EL expressions declaratively using the OEPE Expression Builder. You can access the Expression Builder wherever you see in an editor, dialog, or in the page editor properties.

Before you begin

It may be helpful to have an understanding of EL expressions. For more information, see Section 13.3, "Creating EL Expressions."

To use the Expression Builder:

1. In the MAF Feature Editor, expand the feature you wish to modify, and right-click **Constraints**. Choose **New** > **Constraint Expression**.
2. Under Constraint Expression, click to open the Expression Builder dialog.
3. Create expressions using the following features:

- Use the **Variables** filter to select items that you want to include in the expression. These items are displayed in a tree that is a hierarchical representation of the binding objects. Each icon in the tree represents various types of binding objects that you can use in an expression.

  To narrow down the tree, you can enter search criteria in **Variables**. The EL accessible objects exposed by MAF are located under the **MAF Objects** node and the **Managed Beans** node.

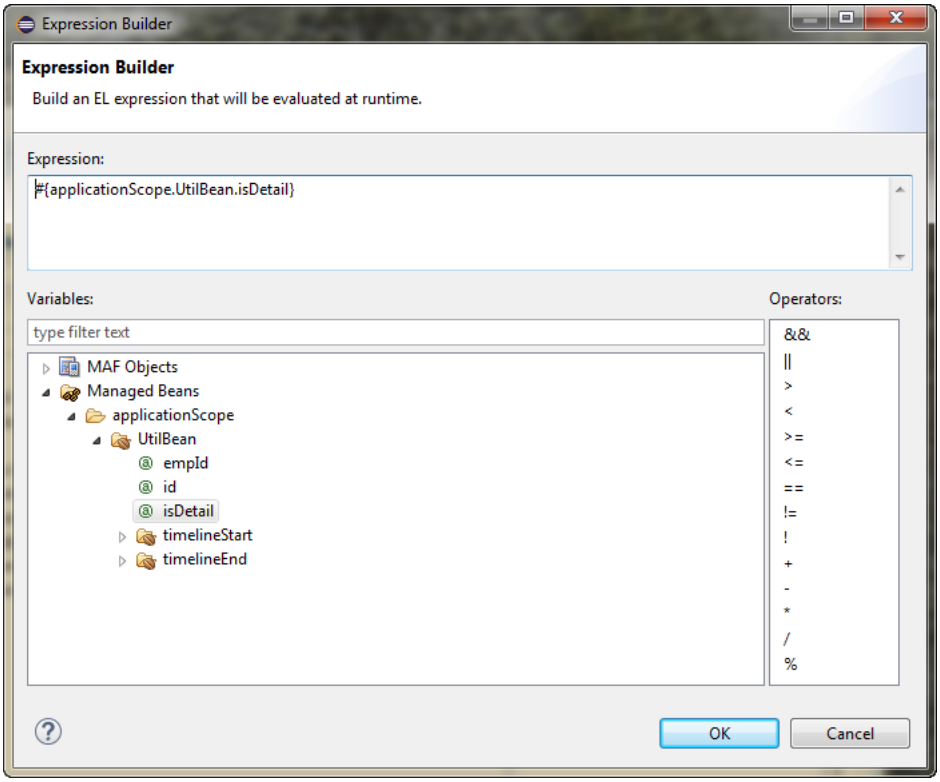
  Tip: For more information about these objects, see the *Java API Reference for Oracle Mobile Application Framework*. See also Section 13.3.5, "About the Categories in the Expression Builder."

  Double-click an item in the tree to move it to the **Expression** box within an EL expression. You can also type the expression directly in the **Expression** box.

- Use the operator buttons to add logical or mathematical operators to the expression.
- To see more information about folders and variables in the tree, hover your mouse over them.

Figure 13–1 shows an example of how to create an EL expression from the Managed Beans category. However, you can create EL expressions from any of the categories described in Section 13.3.5, "About the Categories in the Expression Builder."

Figure 13–1 The Expression Builder Dialog



Tip: For information about using proper syntax to create EL expressions, see the Java EE tutorial at `http://download.oracle.com/javaee/index.html`.

13.3.2.1 About the Method Expression Builder

Table 13–1 shows properties that have the Method Expression Builder option available in the Properties window instead of the Expression Builder option. The only difference between them is that the Method Expression Builder filters out the managed beans depending on the selected property.

Table 13–1 Properties for the Method Expression Builder

Property	Element
action	amx:commandButton
action	amx:commandLink
action	amx:listItem
action	amx:navigationDragBehavior
action	dvtm:chartDataItem
action	dvtm:ieDataItem
action	dvtm:timelineItem
action	dvtm:area
action	dvtm:marker
actionListener	amx:listItem
actionListener	amx:commandButton
actionListener	amx:commandLink
binding	amx:actionListener
mapBoundsChangeListener	dvtm:geographicMap
mapInputListener	dvtm:geographicMap
moveListener	amx:listView
rangeChangeListener	amx:listView
selectionListener	amx:listView
selectionListener	amx:filmStrip
selectionListener	dvtm:areaDataLayer
selectionListener	dvtm:pointDataLayer
selectionListener	dvtm:treemap
selectionListener	dvtm:sunburst
selectionListener	dvtm:timelineSeries
selectionListener	dvtm:nBox
selectionListener	dvtm:areaChart
selectionListener	dvtm:barChart
selectionListener	dvtm:bubbleChart
selectionListener	dvtm:comboChart
selectionListener	dvtm:horizontalBarChart
selectionListener	dvtm:lineChart
selectionListener	dvtm:funnelChart
selectionListener	dvtm:pieChart

Table 13–1 (Cont.) Properties for the Method Expression Builder

Property	Element
selectionListener	dvtm:scatterChart
valueChangeListener	amx:inputDate
valueChangeListener	amx:inputNumberSlider
valueChangeListener	amx:inputText
valueChangeListener	amx:selectBooleanCheckbox
valueChangeListener	amx:selectBooleanSwitch
valueChangeListener	amx:selectManyCheckbox
valueChangeListener	amx:selectManyChoice
valueChangeListener	amx:selectOneButton
valueChangeListener	amx:selectOneChoice
valueChangeListener	amx:selectOneRadio
valueChangeListener	dvtm:statusMeterGauge
valueChangeListener	dvtm:dialGauge
valueChangeListener	dvtm:ratingGauge
viewportChangeListener	dvtm:areaChart
viewportChangeListener	dvtm:barChart
viewportChangeListener	dvtm:comboChart
viewportChangeListener	dvtm:horizontalBarChart
viewportChangeListener	dvtm:lineChart

13.3.2.2 About Non EL-Properties

Table 13–2 shows the properties that do not have the EL Expression Builder option available in the Properties window, because they are not EL-enabled.

Table 13–2 Non EL-Properties

Property	Element
id	all elements
facetName	amx:facetRef
failSafeClientHandler	amx:loadingIndicatorBehavior
failSafeDuration	amx:loadingIndicatorBehavior
group	amx:validationBehavior
name	amx:attribute
name	amx:attributeList
name	amx:attributeListIterator
name	amx:facet
ref	amx:attributeList
type	dvtm:attributeGroups
var	amx:carousel

Table 13–2 (Cont.) Non EL-Properties

Property	Element
var	amx:filmStrip
var	amx:iterator
var	amx:listView
var	amx:loadBundle
var	dvtm:areaChart
var	dvtm:barChart
var	dvtm:bubbleChart
var	dvtm:comboChart
var	dvtm:funnelChart
var	dvtm:horizontalBarChart
var	dvtm:lineChart
var	dvtm:pieChart
var	dvtm:scatterChart
var	dvtm:sparkChart
var	dvtm:geographicMap
varStatus	amx:attributeListIterator

13.3.3 What You May Need to Know About MAF Binding Properties

When you create a databound component using the Expression Builder, the EL expression might reference specific MAF binding properties. At runtime, these binding properties can define such things as the default display characteristics of a databound UI component or specific parameters for iterator bindings. The ADF binding properties are defined by Oracle APIs. For a full list of the available properties for each binding type, see Table 13–3, " Runtime EL Properties of MAF Bindings"

Values assigned to certain properties are defined in the page definition file. For example, iterator bindings have a property called `RangeSize`, which specifies the number of rows the iterator should display at one time. The value assigned to `RangeSize` is specified in the page definition file, as shown below.

```
<iterator Binds="ItemsForOrder" RangeSize="25"
              DataControl="BackOfficeAppModuleDataControl"
              id="ItemsForOrderIterator" ChangeEventPolicy="ppr"/>
```

13.3.4 How to Reference Binding Containers

You can reference the active screen's binding container by the root EL expression `"#{bindings}"` and you can reference another screen's binding container through the expression `"#{data.PageDefName}"`. The Mobile Application Framework AMX binding objects are referenced by name from the binding container `"#{bindings.Name}"`.

Table 13–3 shows a partial list of the properties that you can use in EL expressions to access values of the Mobile Application Framework AMX binding objects at runtime. The properties appear in alphabetical order.

Table 13–3 Runtime EL Properties of MAF Bindings

Runtime Property	Description	Iterator	Action	attributeValues	Tree
`class`	Returns the Java class object for the runtime binding.	Yes	Yes	Yes	Yes
`collectionModel`	Exposes a collection of data. EL expressions used within a component that is bound to a `collectionModel` can be referenced with a `row` variable [1], which will resolve the expression for each element in the collection.	No	No	No	Yes
`collectionModel.makeCurrent`	Causes the selected row to become the current row in the iterator for this binding.	No	No	No	Yes
`collectionModel.selectedRow`	Returns a reference to the selected row.	No	No	No	Yes
`currentRow`	Returns a reference to the current row or data object pointed to by the iterator (for example, built-in navigation actions).	Yes	No	No	No
`currentRow.dataprovider`	Returns a reference to the current row or data object pointed to by the iterator. (This is the same object returned by `currentRow`, just with a different syntax).	Yes	No	No	No
`enabled`	Returns `true` or `false`, depending on the state of the action binding. For example, the action binding may be enabled (`true`) or disabled (`false`) based on the currency (as determined, for example, when the user clicks the First, Next, Previous, or Last navigation buttons).	No	Yes	No	No
`execute`	Invokes the named action or `methodAction` binding when resolved.	No	Yes	No	No
`format`	This is a shortcut for `hints.format`.	No	No	Yes	Yes
`hints`	Returns a list of name-value pairs for UI hints for all display attributes to which the binding is associated.	No	No	Yes	Yes
`inputValue`	Returns the value of the first attribute to which the binding is associated.	No	No	Yes	No
`items`	Returns the list of values associated with the current list-enabled attribute.	No	No	Yes	No
`label`	Available as a child of `hints` or direct child of an attribute. Returns the label (if supplied by control `hints`) for the first attribute of the binding.	No	No	Yes	Yes
`name`	Returns the `id` of the binding as declared in the `PageDef.xml` file.	Yes	Yes	Yes	Yes

Table 13–3 (Cont.) Runtime EL Properties of MAF Bindings

Runtime Property	Description	Iterator	Action	attributeValues	Tree
`rangeSize`	Returns the range size of the iterator binding's row set. This allows you to determine the number of data objects to bind from the data source.	Yes	No	No	Yes
`result`	Returns the result of a method that is bound and invoked by a method action binding.	No	Yes	No	No
`updateable`	Available as a child of `hints` or direct child of an attribute. Returns `true` if the first attribute to which the binding is associated is updateable. Otherwise, returns `false`.	No	No	Yes	Yes
`viewable`	Available as a child of `Tree`. Resolves at runtime whether this binding and the associated component should be rendered or not.	No	No	No	Yes

[1] The EL term *row* is used within the context of a collection component; `row` simply acts as an iteration variable over each element in the collection whose attributes can be accessed by a MAF AMX binding object when the collection is rendered. Attribute and list bindings can be accessed through the `row` variable. The syntax for such expressions will be the same as those used for accessing binding objects outside of a collection, with the `row` variable prepended as the first term: `#{row.bindings.Name.property}`.

13.3.5 About the Categories in the Expression Builder

The following categories are available in the Expression Builder for MAF AMX pages:

- About the Bindings Category
- About the Managed Beans Category
- About the Mobile Application Framework Objects Category

13.3.5.1 About the Bindings Category

This section lists the options available under the Bindings category. The `bindings` and `data` nodes display the same set of supported bindings and properties. Table 13–4 lists available binding types along with the properties that are supported for each binding type. The `securityContext` node supports the following properties:

- `authenticated`
- `userGrantedPrivilege`
- `userInRole`
- `userName`

For example:

```
#{securityContext.authenticated}
#{securityContext.userGrantedPrivilege['submit_privilege']}
#{securityContext.userInRole['manager_role']}
#{securityContext.userName}
```

Table 13–4 Supported Binding Types

Binding Type	Properties
accessorIterator	class
	currentRow: dataProvider
	name
	rangeSize
action	class
	enabled
	execute
	name
attributeValues	autoSubmit
	category
	class
	controlType
	displayHeight
	displayHint
	displayWidth
	filedorder
	format
	hints: allows.read, allows.update, autoSubmit, category, controlType, displayHeight, displayHint, displayWidth, filedorder, format, label, mandatory, precision, tooltip, updateable
	inputValue
	items
	iteratorBinding
	label
	mandatory
	name
	precision
	tooltip
	updateable

Table 13–4 (Cont.) Supported Binding Types

Binding Type	Properties
button	autoSubmit
	category
	class
	controlType
	displayHeight
	displayHint
	displayWidth
	filedorder
	format
	hints: allows.read, allows.update, autoSubmit, category, controlType, displayHeight, displayHint, displayWidth, filedorder, format, label, mandatory, precision, tooltip, updateable
	inputValue
	items
	iteratorBinding
	label
	mandatory
	name
	precision
	tooltip
	updateable
invokeAction	always
	deferred
iterator	class
	currentRow: dataProvider
	name
	rangeSize

Table 13–4 (Cont.) Supported Binding Types

Binding Type	Properties
list	autoSubmit
	category
	class
	controlType
	displayHeight
	displayHint
	displayWidth
	filedorder
	format
	hints: format, allows.read, allows.update, autoSubmit, category, controlType, displayHeight, displayHint, displayWidth, filedorder, format, label, mandatory, precision, tooltip, updateable
	inputValue
	items
	iteratorBinding
	label
	mandatory
	name
	precision
	tooltip
	updateable
methodAction	class
	enabled
	execute
	name
	operationEnabled
	operationInfo
	paramsMap
	result
methodIterator	class
	currentRow: dataProvider
	name
	rangeSize
searchAction	class
	enabled
	execute
	name
	operationEnabled
	operationInfo
	paramsMap
	result

Table 13–4 (Cont.) Supported Binding Types

Binding Type	Properties
tree	category
	class
	collectionModel: bindings, makeCurrent, selectedRow, <AttrName>
	displayHeight
	displayHint
	displayWidth
	filedorder
	format
	hints: category, displayHeight, displayHint, displayWidth, filedorder, format, label, mandatory, precision, tooltip, updateable, <AttrName>
	iteratorBinding
	label
	mandatory
	name
	precision
	rangeSize
	tooltip
	updateable
	viewable
variable	class
	currentRow: dataProvider
	name
variableIterator	class
	currentRow: dataProvider
	name

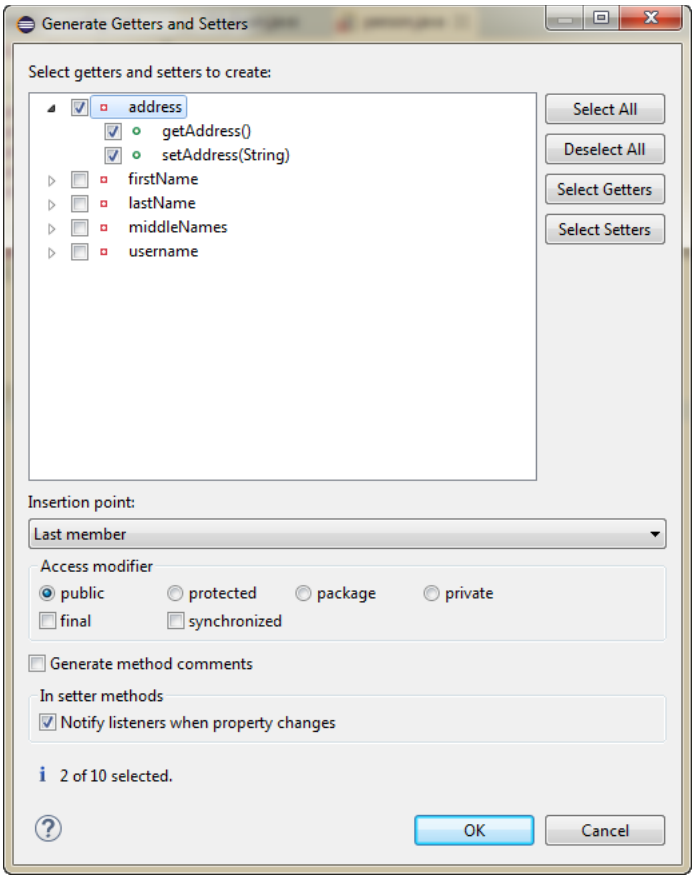
13.3.5.2 About the Managed Beans Category

This section lists the options available under the Managed Beans category.

- `applicationScope`: Managed Beans > `applicationScope` node contains everything that is defined at the application level (for example, application-scoped managed beans).
- `pageFlowScope`: Managed Beans > `pageFlowScope` node contains everything that is defined at the page flow level (for example, page flow-scoped managed beans).
- `viewScope`: Managed Beans > `viewScope` node contains everything that is defined at the view level (for example, view-scoped managed beans).

The MAF runtime will register itself as a listener on managed bean property change notifications so that EL expressions bound to UI components that reference bean properties will update automatically if the value of the property changes. Sourcing these notifications requires some additional code in the beans' property accessors. To automatically generate the necessary code to source notifications from your beans' property accessors, select the **Notify listeners when property changes** checkbox in the Generate Getters and Setters dialog (see Figure 13–2).

Figure 13–2 Notify Listeners When Property Changes



It is not necessary to add this code to simply reference bean methods or properties through EL, but it is necessary to keep the rendering of any EL expressions in the active form that depend on values stored in the bean current if those values change, especially if the change is indirect, such as a side effect of executing a bean method that changes one or more property values. For information about property changes and the `PropertyChangeSupport` class, see Section 13.12, "About Data Change Events."

This example illustrates how to retrieve a value bound to another managed bean attribute programmatically.

```
public void someMethod()
{
  Object value =
AdfmfJavaUtilities.evaluateELExpression("#{applicationScope.MyManagedBean.someProp
erty}");
```

This example illustrates how to execute bindings programmatically from a managed bean.

```
public void someMethod()
{
  Object value =
AdfmfJavaUtilities.evaluateELExpression("#{bindings.someDataControlMethod.execute}
");
```

Note: If you declare a managed bean within the `applicationScope` of a feature but then try to reference that bean through EL in another feature at design time, you will see a warning in the design time about invalid EL. This warning is due to the fact that the design time cannot find a reference in the current project for that bean. You can reference that bean at runtime only if you first visit the initial feature where you declared the bean and the bean is instantiated before you access it through EL in another feature. This is not the case for the `PreferenceValue` element as it uses the `Name` attribute value as the node label.

13.3.5.3 About the Mobile Application Framework Objects Category

The Mobile Application Framework Objects category lists various objects defined in the Mobile Application Framework that can be referenced using EL, such as object scopes.

MAF variables and managed bean references are defined within different object scopes that determine the variable's lifetime and visibility. In order of decreasing visibility, they are application scope, page flow scope, and view scope. For more information about the different object scopes, see Section 13.2, "About Object Scope Lifecycles.".

In addition to these variable-containing scopes, MAF defines scopes that can expose information about device properties and application preferences. These scopes have application-level lifetime and visibility.

The following are available under the Mobile Application Framework Objects category:

- `applicationScope`: The `applicationScope` node contains everything that is defined at the application level (for example, application-scoped managed beans). EL variables defined in the application scope are available for the life of the application, across feature boundaries.
- `pageFlowScope`: The `pageFlowScope` node contains everything that is defined at the page flow level (for example, page flow-scoped managed beans). EL variables defined in the page flow scope namespace are available for the duration of a feature, within the bounds of a single feature.
- `preferenceScope`: The `preferenceScope` node contains all the application and feature preferences.

  Preference elements use the `Id` attribute value as the node label in the Expression Builder, except for the `PreferenceValue` element. The `PreferenceValue` element uses the `Name` attribute value as the node label in the Expression Builder.

  Note: Where string tokens in EL expressions contain a dot (".") or any special character, or a reserved word like `default`, the Expression Builder surrounds such string tokens with a single quote and bracket. When the feature ID or preference component ID contains a dot, the Expression Builder displays each part of the ID that is separated by a dot as a separate property in the `preferenceScope` hierarchy. The expression generated also takes each part of the ID separated by a dot as a separate property.

  Following are some sample `preferenceScope` EL expressions:

```
"#{preferenceScope.application.OracleMobileApp.Edition['default']}"
```

- `viewScope`: This node contains everything that is defined at the view level (for example, view-scoped managed beans). EL variables defined in the view scope namespace are available for the duration of the view, within the bounds of a single page of a feature.
- `row`: The `row` object is an intermediate variable that is a shortcut to a single provider in the `collectionModel`. Its name is the value of the `var` attribute of the parent component (such as List View or Carousel).

  Note: It is not possible to evaluate `#{row}` or properties of `row` using `AdfmfJavaUtilities.evaluateELExpression`. These expressions will return a null value.

- `viewControllerBundle`

  This is the name of the resource bundle variable that points to a resource bundle defined at the project level. This node is shown only after the `amx:loadBundle` element has been dropped and a resource bundle has been created. The name of this node will vary as it depends on the variable name of `amx:loadBundle`. This node will display all strings declared in the bundle.

  This example shows an example of AMX code for `viewControllerBundle`.

```
<amx:loadBundle basename="mobile.ViewControllerBundle"
var="viewcontrollerBundle"/>
```

13.3.6 About EL Events

EL events play a significant role in the functioning of the MAF AMX UI, enabling expressions with common terms to update in sync with each other.

EL expressions can refer to values in various contexts. The example below shows the creation of two Input Number Slider components, with each component tied to an `applicationScope` value. The output text then uses EL to display a simple addition equation along with the calculated results. When the framework parses the EL expression in the output text labels, it determines that the expression contains references to two values and creates event listeners (see Section 12.10, "Using Event Listeners") for the output text on those two values. When the value of the underlying expression changes, an event is generated to all listeners for that value.

Note: If you are referencing properties on a managed bean (as opposed to scope objects) you have to add the listeners. For more information, see Section 13.3.5.2, "About the Managed Beans Category."

```
<amx:inputNumberSlider id="slider1" label="X" value="#{applicationScope.X}"/>
<amx:inputNumberSlider id="slider2" label="Y" value="#{applicationScope.Y}"/>
<amx:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
```

In the example above, two components are updating one value each, and one component is consuming both values. The next example shows that the behavior would be identical if a third Input Number Slider component is added that references one of the existing values.

```
<amx:inputNumberSlider id="slider1" label="X" value="#{applicationScope.X}"/>
<amx:inputNumberSlider id="slider2" label="Y" value="#{applicationScope.Y}"/>
<amx:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
<amx:inputNumberSlider id="slider3" label="X" value="#{applicationScope.X}"/>
```

In the example above, when either Input Number Slider component updates `#{applicationScope.X}`, the other is automatically updated along with the Output Text.

13.3.7 How to Use EL Expressions Within Managed Beans

While OEPE creates many needed EL expressions for you, and you can use the Expression Builder to create those not built for you, there may be times when you need to access, set, or invoke EL expressions within a managed bean.

This example shows how you can get a reference to an EL expression and return (or create) the matching object.

```
public static Object resolveExpression(String expression) {
  return AdfmfJavaUtilities.evaluateELExpression(expression);
}
```

This example shows how you can resolve a method expression.

```
public static Object resloveMethodExpression(String expression,
                                              Class returnType,
                                              Class[] argTypes,
                                              Object[] argValues) {
  MethodExpression methodExpression =
AdfmfJavaUtilities.getMethodExpression(expression,     returnType, argTypes);
  return methodExpression.invoke(AdfmfJavaUtilities.getAdfELContext(), argValues);
}
```

This example shows how you can set a new object on a managed bean.

```
public static void setObject(String expression, Object newValue) {
  AdfmfJavaUtilities.setELValue(expression, newValue);
}
```

13.4 Creating and Using Managed Beans

Managed beans are Java classes that you register with the application using various configuration files. When the MAF application starts up, it parses these configuration files and the beans are made available and can be referenced in an EL expression, allowing access to the beans' properties and methods. Whenever a managed bean is referenced for the first time and it does not already exist, the Managed Bean Creation Facility instantiates the bean by calling the default constructor method on the bean. If any properties are also declared, they are populated with the declared default values.

Often, managed beans handle events or some manipulation of data that is best handled at the front end. For a more complete description of how to use managed beans, see the Java EE tutorial at http://www.oracle.com/technetwork/java/index.html.

> **Best Practice:** Use managed beans to store only bookkeeping information, for example the current user. All application data and processing should be handled by logic in the business layer of the application.

Note: EL expressions must explicitly include the scope to reference the bean. For example, to reference the `MyBean` managed bean from the `pageFlowScope` scope, your expression would be `#{pageFlowScope.MyBean}`.

13.4.1 How to Create a Managed Bean in OEPE

You can create a managed bean and register it with the MAF application at the same time using the editor.

Before you begin

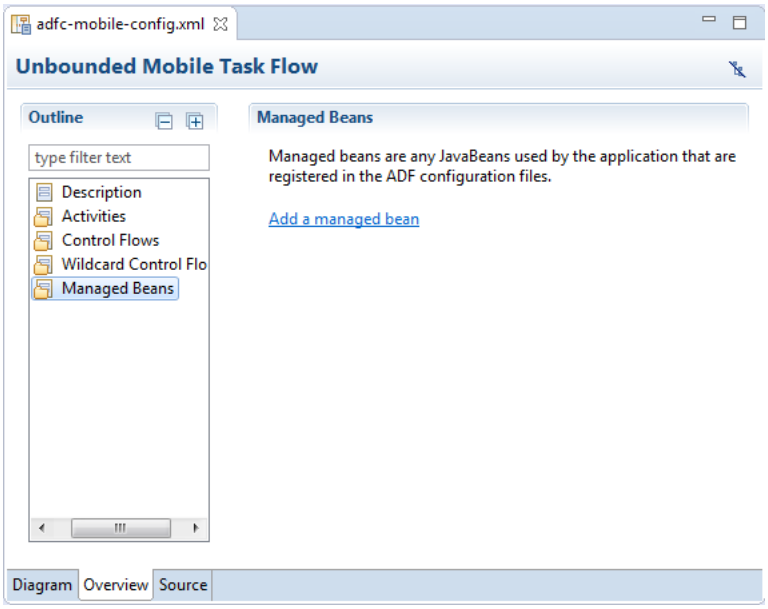
It may be helpful to have an understanding of managed beans. For more information, see Section 13.4, "Creating and Using Managed Beans."

To create and register a managed bean:

1. In the Project Explorer, expand the view project, then expand **ViewContent** and double-click **adfc-mobile-config.xml**.
2. In the editor window, click the **Overview** tab.
3. In the overview editor, click the **Managed Beans** navigation tab.

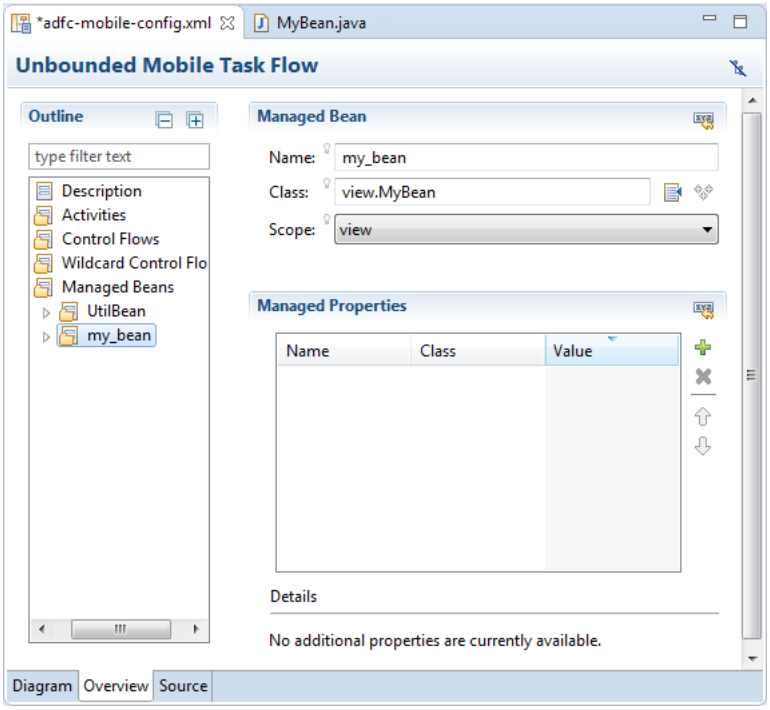
   Figure 13–3 shows the editor for the `adfc-mobile-config.xml` file.

Figure 13–3 Managed Beans in the adfc-mobile-config.xml File



4. Select Managed Beans and click **Add a managed bean**, see Figure 13–4.

Figure 13–4 Creating a Managed Bean



5. To define the managed bean, do the following:
   - Enter a name for the bean.
   - Either browse to an existing class, or enter the name of a new class and click . A new Java class is created and opened in the editor.
   - Choose the scope of the bean. The options are:
     - application
     - page flow
     - view

     Note: When determining what scope to register a managed bean with or to store a value in, always try to use the narrowest scope possible. For more information about the different object scopes, see Section 13.2, "About Object Scope Lifecycles."

6. You can optionally add managed properties for the bean. When the bean is instantiated, any managed properties will be set with the provided value. With the bean selected in the Managed Bean table, click to add a row to the Managed Properties table. In the Properties window, enter a property name (other fields are optional).

Note: While you can declare managed properties using this editor, the corresponding code is not generated on the Java class. You must add that code by creating private member fields of the appropriate type, and then by choosing the **Source** > **Generate Getters and Setters for MAF** menu item on the context menu of the code editor to generate the corresponding `get` and `set` methods for these bean properties.

13.4.2 What Happens When You Use OEPE to Create a Managed Bean

When you create a managed bean and elect to generate the Java file, OEPE creates a stub class with the given name and a default constructor. This example shows the code added to the `MyBean` class stored in the view package.

```
package view;

public class MyBean {
    public MyBean() {
    }
}
```

You now must add the logic required by your page. You can then refer to that logic using an EL expression that refers to the `managed-bean-name` given to the managed bean. For example, to access the `myInfo` property on the `my_bean` managed bean, the EL expression would be:

```
#{my_bean.myInfo}
```

OEPE also adds a `managed-bean` element to the `adfc-mobile-config.xml` file (or to the task flow file that is being edited). The next example shows the `managed-bean` element created for the `MyBean` class.

```
<managed-bean>
  <managed-bean-name>my_bean</managed-bean-name>
  <managed-bean-class>view.MyBean</managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

13.5 Exposing Business Services with Data Controls

Once you have your application's services in place, you can use OEPE to create data controls that provide the information needed to declaratively bind UI components to those services.

You generate data controls from the New dialog, available from **File** > **New** > **Other**. Data controls consist of one or more XML metadata files that define the capabilities of the services that the bindings can work with at runtime. The data controls work in conjunction with the underlying services.

13.5.1 How to Create Data Controls

You create adapter-based data controls from within the Project Explorer of OEPE.

Before you begin:

It may be helpful to have a general understanding of using data controls. For more information, see Section 13.5, "Exposing Business Services with Data Controls."

You will need to complete this task:

Create an application workspace and add the business services on which you want to base your data control. For information on creating an application workspace, see Section 2.3, "Creating a MAF Application."

To create a data control:

1. Right-click the top-level node for the view project in the Project Explorer and choose **New** > **Data Control**.
2. In the New dialog, expand **Oracle**, then **Mobile Application Framework** and select **Data Controls**. Click **Next**.
3. On the Data Control Source page:
   - Choose the project.
   - Choose the type of data control.
   - Click , and in the Data Control Source dialog choose the class.

   Click **Next**.
4. Complete the remaining steps of the wizard.

Note: In some cases, you can create a data control by right-clicking the class or object on which the data control will be based and choosing Create Data Control.

If the object is a bean class, or a WSDL file, right-click and choose **Model Components** > **Create Data Control**.

13.5.2 What Happens in Your Project When You Create a Data Control

When you create a data control, OEPE creates the data control definition file (`DataControls.dcx`), opens the file in the Data Control Manager, and displays the file's hierarchy in the Palette. This file enables the data control to work directly with the services and the bindings.

You can see the code from the corresponding XML file by clicking the Source tab in the editor window.

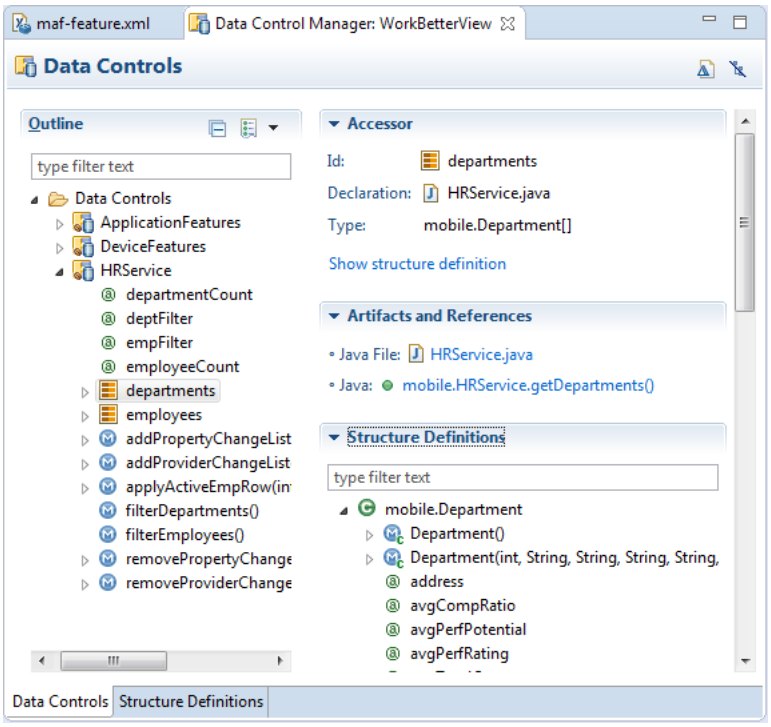
13.5.2.1 The Data Control Manager

The Data Control Manager provides a view of the hierarchies of data control objects and exposed methods of your data model in the `DataControls.dcx` file. You open the Data Control Manager from the Project Explorer by expanding the application project node, then expanding **MAF** and double-clicking **Data Control Manager**.

See Table 13–5 for a description of the icons that are used in the Data Control Manager and Palette.

You can change the labels and tooltips for a data control object by selecting the object and clicking the **Edit structure definition** link, as shown in Figure 13–5.

Figure 13–5 Data Control Manager



13.5.2.2 The Palette

The Palette appears, by default, at the bottom left corner of the IDE. You can create databound UI components by dragging nodes from the Palette to the design editor for a page. You can create tags and tag bindings by double-clicking in the Palette, and if you drag a tag onto a page, you can bind using the Expression Builder.

13.5.3 Data Control Built-in Operations

The data control framework defines a standard set of operations for data controls. These operations are implemented using functionality of the underlying business service. At runtime, when one of these data collection operations is invoked by name by the data binding layer, the data control delegates the call to an appropriate service method to handle the built-in functionality. For example, in bean data controls, the `Next` operation relies on the bean collection's iterator.

Most of the built-in operations affect the current row. However, the `execute` operation refreshes the data control itself.

The operations available vary by data control type and the functionality of the underlying business service. Here is the full list of built-in operations:

- `Create`: Creates a new row that becomes the current row. This new row is also added to the row set.
- `CreateInsert`: Creates a new row that becomes the current row and inserts it into the row set.
- `Create With Parameters`: Uses named parameters to create a new row that becomes the current row and inserts it into the row set.
- `Delete`: Deletes the current row.

- `Execute`: Refreshes the data collection by executing or reexecuting the accessor method.

  `ExecuteWithParams`: Refreshes the data collection by first assigning new values to variables that passed as parameters, then executing or reexecuting the associated query. This operation is only available for data control collection objects that are based on parameterized queries.
- `First`: Sets the first row in the row set to be the current row.
- `Last`: Sets the last row in the row set to be the current row.
- `Next`: Sets the next row in the row set to be the current row.
- `Next Set`: Navigates forward one full set of rows.
- `Previous`: Sets the previous row in the row set to be the current row.
- `Previous Set`: Navigates backward one full set of rows.
- `removeRowWithKey`: Tries to find a row using the serialized string representation of the row key passed as a parameter. If found, the row is removed.
- `setCurrentRowWithKey`: Tries to find a row using the serialized string representation of the row key passed as a parameter. If found, that row becomes the current row.
- `setCurrentRowWithKeyValue`: Tries to find a row using the primary key attribute value passed as a parameter. If found, that row becomes the current row.

13.6 Creating Databound UI Components from the Data Controls Palette

You can design a databound user interface by dragging an item from the Palette and dropping it on a page as a specific UI component. When you use data controls to create a UI component, OEPE automatically creates the various code and objects needed to bind the component to the data control you selected.

In the Palette, each data control object is represented by a specific icon. Table 13–5 describes what each icon represents, where it appears in the Palette hierarchy, and what components it can be used to create.

Table 13–5 Palette Icons and Object Hierarchy

Icon	Name	Description	Used to Create...
	Data Control	Represents a data control.	Serves as a container for the other objects and is not used to create anything.
	Collection	Represents a named data collection returned by an accessor method or operation.	Forms, tables, graphs, trees, range navigation components, master-detail components, and selection list components
	Structured Attribute	Represents a returned object that is neither a Java primitive type (represented as an attribute) nor a collection of any type.	Forms, label, text field, date, list of values, and selection list components.
	Attribute	Represents a discrete data element in an object (for example, an attribute in a row).	Label, text field, date, list of values, and selection list components.

Table 13–5 (Cont.) Palette Icons and Object Hierarchy

Icon	Name	Description	Used to Create...
	Method	Represents a method or operation in the data control or one of its exposed structures that may accept parameters, perform some business logic and optionally return single value, a structure, or a collection.	Command components. For methods that accept parameters: command components and parameterized forms.
	Method Return	Represents an object that is returned by a method or other operation. The returned object can be a single value or a collection. A method return appears as a child under the method that returns it. The objects that appear as children under a method return can be attributes of the collection, other methods that perform actions related to the parent collection, or operations that can be performed on the parent collection.	For single values: text fields and selection lists. For collections: forms, tables, trees, and range navigation components. When a single-value method return is dropped, the method is not invoked automatically by the framework. To invoke the method, you can drop the corresponding method as a button. If the form is part of a task flow, you can create a method activity to invoke the method.
	Operation	Represents a built-in data control operation that performs actions on the parent object.	UI command components, such as buttons and links.
	Parameter	Represents a parameter value that is declared by the method or operation under which it appears.	Label, text, and selection list components.

13.6.1 How to Use the Data Controls Palette

OEPE provides you with a predefined set of UI components from which to choose for each data control item you can drop.

Before you begin:

It may be helpful to have an understanding of the different objects in the Palette. For more information, see Section 13.6, "Creating Databound UI Components from the Data Controls Palette."

You will need to complete these tasks:

- Create a data control as described in Section 13.5.1, "How to Create Data Controls."
- Create a a MAF AMX page as described in Section 11.3.1.2, "Creating MAF AMX Pages."

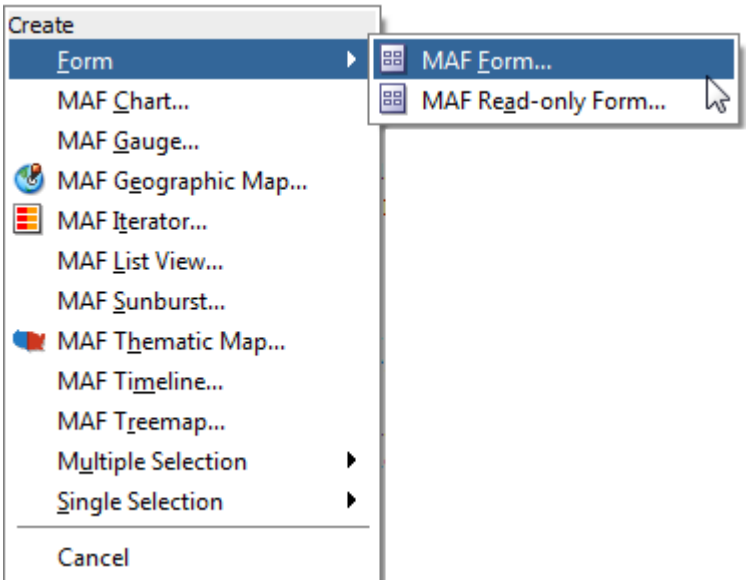
To use the Palette to create UI components:

1. Select an item in the Palette and drag it onto the visual editor for your page. For a definition of each item in the panel, see Table 13–5, " Palette Icons and Object Hierarchy".
2. From the ensuing context menu, choose a UI component.

   When you drag an item from the Palette and drop it on a page, OEPE displays a context menu of all the default UI components available for the item you dropped. The components displayed are based on the libraries in your project.

   Figure 13–6 shows the context menu displayed when a data collection from the Palette is dropped on a page.

Figure 13–6 Dropping Component From the Palette



Depending on the component you select from the context menu, OEPE may display a dialog that enables you to define how you want the component to look. For example, if you select **Form** from the context menu, the Edit Form Fields dialog opens. Once you select a component, OEPE inserts the UI component on the page in the visual editor.

The UI components selected by default are determined first by any UI hints set on the corresponding business object. If no UI hints have been set, then OEPE uses input components for standard forms and tables, and output components for read-only forms and tables. Components for lists are determined based on the type of list you chose when dropping the data control object.

By default, the UI components created when you use the Data Controls are bound to attributes in the MAF data control and may have built-in features, such as:

- Databound labels
- Tooltips
- Formatting
- Basic navigation buttons
- Validation, if validation rules are attached to a particular attribute.

The default components are fully functional without any further modifications. However, you can modify them to suit your particular needs.

Tip: If you want to change the type of MAF databound component used on a page, the easiest method is to use either the visual editor or the structure window to delete the component, and then drag and drop a new one from the Palette. When you use the visual editor or the structure window to delete a databound component from a page, if the related binding objects in the page definition file are not referenced by any other component, OEPE automatically deletes those binding objects for you (automatic deletion of binding objects will not happen if you use the source editor).

13.6.2 What Happens When You Use the Data Controls Palette

When an application is built using the Palette, OEPE does the following:

- Creates a `DataBindings.cpx` file in the `adfmsrc/mobile` package for the project (if one does not already exist), and adds an entry for the page.

  A `DataBindings.cpx` files defines the *binding context* for the application. The binding context is a container object that holds a list of available data controls and data binding objects. The `DataBindings.cpx` file maps individual pages to the binding definitions in the page definition file and registers the data controls used by those pages. For more information, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts."

- Creates the `adfm.xml` file in the META-INF directory. This file creates a registry for the `DataBindings.cpx` file, which allows the application to locate it at runtime so that the binding context can be created.

- Adds a page definition file (if one does not already exist for the page) to the page definition subpackage. The default subpackage is `mobile.pageDefs` in the `adfmsrc` directory.

  The page definition file (*`pageName`*`PageDef.xml`) defines the binding container for each page in an application's view layer. The binding container provides runtime access to all the binding objects for a page. For more information about the page definition file, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts."

  Tip: The current binding container is also available from `AdfContext` for programmatic access.

- Configures the page definition file, which includes adding definitions of the binding objects referenced by the page.

- Adds the given component to the page.

  These prebuilt components include the data binding expression language (EL) expressions that reference the binding objects in the page definition file. For more information, see Section 13.3.1, "About Data Binding EL Expressions."

- Adds all the libraries, files, and configuration elements required by the UI components. For more information on the artifacts required for databound components, see Section 2.3.2, "What Happens When You Create an MAF Application."

13.7 What Happens at Runtime: How the Binding Context Works

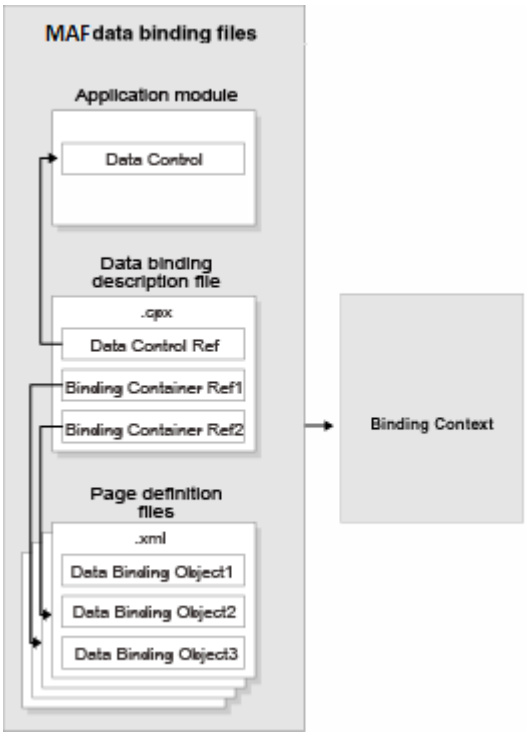
When a page contains MAF bindings, at runtime the interaction with the business services initiated from the client or controller is managed by the application through a

single object known as the **binding context**. The binding context is a runtime map (named *data* and accessible through the EL expression `#{data}`) of all data controls and page definitions within the application.

The MAF creates the binding context from the application, `DataBindings.cpx`, and page definition files, as shown in Figure 13–7. The union of all the `DataControls.dcx` files and any application modules in the workspace define the available data controls at design time, but the `DataBindings.cpx` file defines what data controls are available to the application at runtime. The `DataBindings.cpx` file lists all the data controls that are being used by pages in the application and maps the binding containers, which contain the binding objects defined in the page definition files, to web page URLs. The page definition files define the binding objects used by the application pages. There is one page definition file for each page.

The binding context does not contain live instances of these objects. Instead, it is a map that contains references that become data control or binding container objects on demand. When the object (such as a page definition) is released from the application (for example when a task flow ends or when the binding container or data control is released at the end of the request), data controls and binding containers turn back into reference objects. For more information about the `DataBindings.cpx` file, see Section 11.3.2.3.5, "What You May Need to Know About Generated Drag and Drop Artifacts."

Figure 13–7 File Binding Context Runtime Usage



Note: Carefully consider the binding styles you use when configuring components. More specifically, combining standard bindings with managed bean bindings will frequently result in misunderstood behaviors because the class instances are unlikely to be the same between the binding infrastructure and the managed bean infrastructure. If you mix bindings, you may end up calling behavior on an instance that isn't directly linked to the UI.

For more information on working with bindings in MAF, see the following:

- Section 11.3.2.3.4, "What You May Need to Know About Generated Bindings"
- Section 11.3.2.3.6, "Using the MAF AMX Editor Bindings Tab"
- Section 11.3.2.3.7, "What You May Need to Know About Removal of Unused Bindings"

13.8 Working with Data Control Attributes

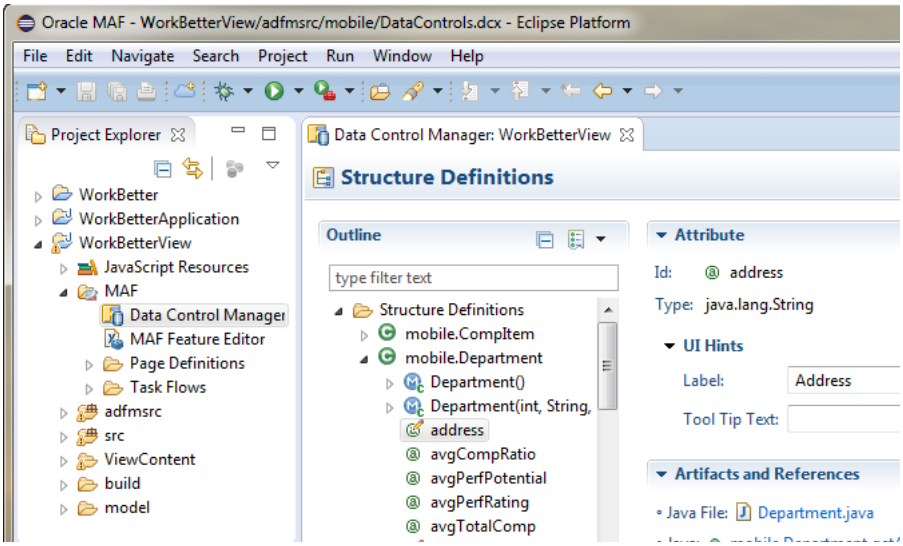
When you create a data control for your business services, you create a data control structure file for an individual data object in which you can set UI hints for the data object's persistent attributes.

13.8.1 Setting UI Hints on Attributes

You can set UI hints on attributes so that those attributes are displayed and labeled in a consistent and localizable way by any UI components that use those attributes. UI hints determine things such as the type of UI component used to display the attribute, the label, the tooltip, and whether the field should be automatically submitted. You can also determine whether a given attribute is displayed or hidden. To create UI hints for attributes, use the Data Control Manager for the data object's data control structure file, which is accessible from the Project Explorer, as shown in Figure 13–8.

Attributes where a value has been entered for one of the UI hints use the icon .

Figure 13–8 Editing Data Control Attribute UI Hints



To set a UI hint:

1. In the Project Explorer, expand the assembly or view project node, and then the MAF node.
2. Double-click **Data Control Manager**.

   > 3. Under Outline, expand mobile.Department. You can edit the attributes by selecting them. Attributes with a pencil on the icon already have an entry in one of the two editable fields, see the attached image.
3. In the editor, under Outline, select the attribute you want to edit.
4. If necessary, expand Attribute then expand UI Hints, and enter the values you want for **Label** and **Tool Tip Text**.

13.9 Creating and Using Bean Data Controls

A bean data control serves as a metadata wrapper for a bean class and exposes the bean's code elements as data control objects, which can then be used to bind those code elements to UI components. Java bean data controls obtain their data structure from POJOs (plain old Java objects).

Before you begin:

It may be helpful to have a general understanding of data controls. For more information, see Section 13.5.1, "How to Create Data Controls."

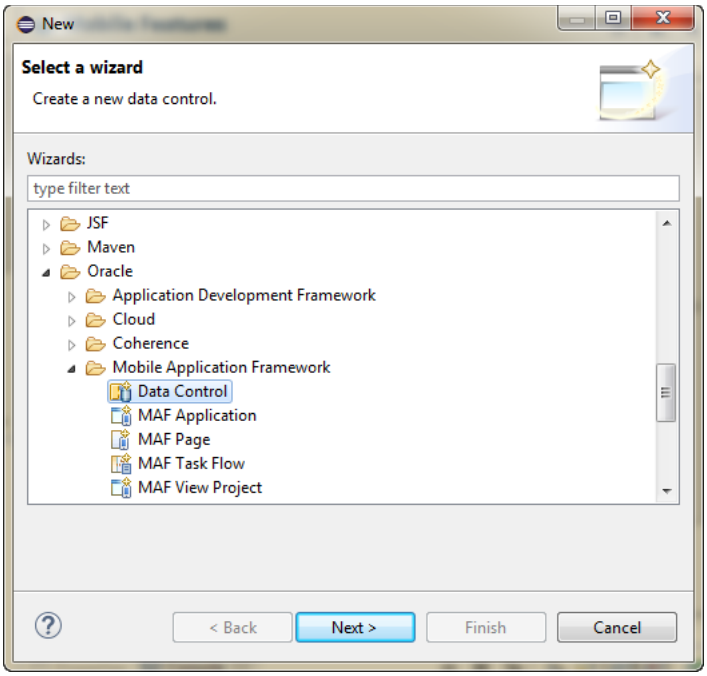
You will need to complete this task:

Create an application workspace and add the business services on which you want to base your data control. For information on creating an application workspace, see Section 2.3, "Creating a MAF Application."

To create a bean data control:

1. In the Project Explorer, right-click the assembly project, and then select **File > New > Other** from the main OEPE menu.
2. In the **New** dialog, expand the **Oracle**, then expand **Mobile Application Framework** and choose **Data Control**. Click **Next**.

Figure 13–9 Creating a New Data Control



3. On the **Data Control Source** page of the wizard, choose the project and select the data source **Java Bean**.
4. Click and in the Data Control Source dialog choose the class that to use as the source for the data control. Click **OK** in the dialog, then click **Next** in the wizard.
5. In the Data Control Details page, enter a Data control id and click **Next**.
6. Check details of the data control in the Summary page and click **Finish**.

Note: If the JavaBean is using a background thread to update data in the UI, you need to manually call `oracle.adfmf.framework.api.AdfmfJavaUtilities.flushDataChangeEvent`. For information about the `flushDataChangeEvent` method, see Section 13.12, "About Data Change Events."

13.9.1 What You May Need to Know About Serialization of Bean Class Variables

MAF does not serialize to JavaScript Object Notation (JSON) data bean class variables that are declared as transient. To avoid serialization of a chain of nested objects, you should define them as transient. This strategy also helps to prevent the creation of cyclic objects due to object nesting.

Consider the following scenario: you have an `Employee` object that has a child `Employee` object representing the employee's manager. If you do not declare the child object transient, a chain of serialized nested objects will be created when you attempt to calculate the child `Employee` object at runtime.

To serialize and deserialize Java objects into JSON objects, use the `JSONBeanSerializationHelper` class. The `JSONBeanSerializationHelper` class enables you to implement your own custom JSON serialization and deserialization, and it provides a hook to alter the JSON object after the JSON serialization (and deserialization) process. The `JSONBeanSerializationHelper` class is similar to the

`GenericTypeSerializationHelper` class, which you can use to serialize and deserialize `GenericType` objects in REST web services. For details, see the `oracle.adfmf.framework.api.JSONBeanSerializationHelper` class in the *Java API Reference for Oracle Mobile Application Framework*.

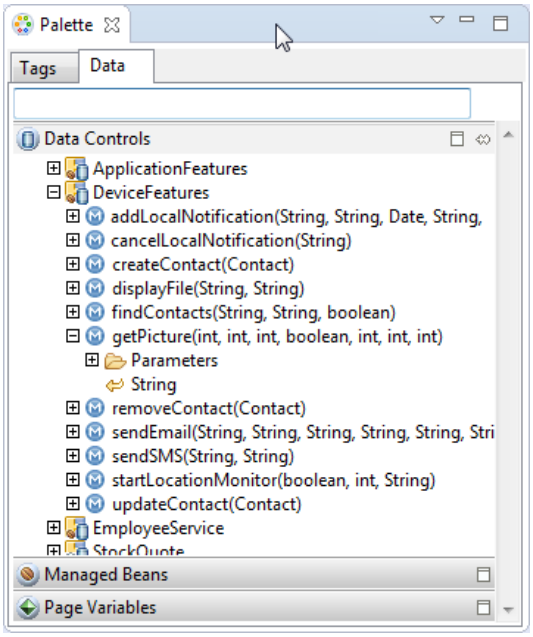
MAF does not support serializing objects of the `GregorianCalendar` class. The `JSONBeanSerializationHelper` class cannot serialize objects of the `GregorianCalendar` class because the `GregorianCalendar` class has cyclical references in it. Instead, use `java.util.Date or java.sql.Date` for date manipulation. The following example shows how to convert a `GregorianCalendar` object using `java.util.Date`:

```
Calendar calDate = new GregorianCalendar();
calDate.set(1985, 12, 1); // "January 1, 1986"
Date date = calDate.getTime();
```

13.10 Using the DeviceFeatures Data Control

MAF exposes device-specific features that you can use in your application through the DeviceFeatures data control, a component that appears in the Data Controls Manager when you create a new MAF application. The Cordova Java API is abstracted through this data control, enabling the application features implemented as MAF AMX to access various services embedded on a device. By dragging and dropping the operations provided by the DeviceFeatures data control into a MAF AMX page, you can add functions to manage the user contacts stored on a device, create and send both email and SMS text messages, ascertain the location of a device, use a device's camera, and retrieve images stored in a device's file system. The following sections describe each of these operations in detail, including how to use them declaratively and how to implement them with Java code and JavaScript.

Figure 13–10 MAF DeviceFeatures Data Control in the Editor



The DeviceFeatures data control appears in the Data Controls Manager automatically when you create an application using the MAF application template. Figure 13–10

shows the DeviceFeatures data control in the Data Control Manager. The following methods are available:

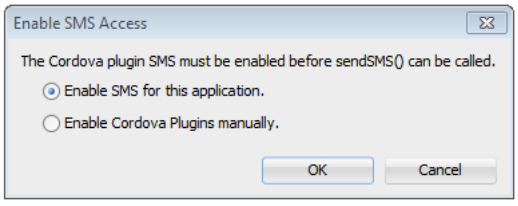
- `addLocalNotification`
- `cancelLocalNotification`
- `createContact`
- `displayFile`
- `findContacts`
- `getPicture`
- `removeContact`
- `sendEmail`
- `sendSMS`
- `startLocationMonitor`
- `updateContact`

After you create a page, you can drag DeviceFeatures data control methods (or other objects nested within those methods) from the Palette to a MAF AMX view to create command buttons and other components that are bound to the associated functionality. You can accept the default bindings or modify the bindings using EL. You can also use JavaScript or Java to implement or configure functionality. For information on how to include data controls in your MAF application, see Section 11.3.2, "How to Add UI Components and Data Controls to an MAF AMX Page."

The `DeviceManager` is the object that enables you to access device functionality. You can get a handle on this object by calling `DeviceManagerFactory.getDeviceManager`. The following sections describe how you can invoke methods like `getPicture` or `createContact` using the `DeviceManager` object.

With the exception of network access, access to all of the Apache Cordova-enabled device capabilities is not enabled by default for MAF applications. The operations that the DeviceFeatures data control expose require that the associated plugin be enabled in the MAF application for the operation to function correctly at runtime. If, for example, you want to use the `sendSMS` operation from the DeviceFeatures data control, you must enable the SMS plugin in the MAF application. You can enable plugins manually or you can choose the appropriate option in the dialog that OEPE displays when you drag and drop an operation that does not have the associated plugin enabled in the MAF application. For example, OEPE displays the dialog in Figure 13–11 when you drag and drop the `sendSMS` operation to a MAF AMX page in a MAF application that has yet to enable the SMS plugin.

Figure 13–11 Enabling Plugin for a DeviceFeatures Data Control Operation



If the plugin that an operation requires is not enabled, a warning message appears in the source file of the MAF AMX page. Assume, for example, that the MAF application

does not enable the SMS plugin. The warning message shown in Figure 13–12 appears in MAF AMX pages where the application attempts to invoke the `sendSMS` operation. You resolve this issue by manually enabling the plugin, as described in Chapter 9, "Using Plugins in MAF Applications."

Figure 13–12 DeviceFeatures Data Control Operation Requires Plugin

13.10.1 How to Use the getPicture Method to Enable Taking Pictures

The DeviceFeatures data control includes the `getPicture` method, which enables MAF applications to leverage a device's camera and photo library so end users can take a photo or retrieve an existing image. There are three examples near the end of this section. The first shows JavaScript code that enables an end user to take a picture with a device's camera. The second and third show Java code that will enable an end user to take a picture or retrieve a saved image. For information about the `getPicture` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

The following parameters control where the image is taken from and how it is returned:

> **Note:** If you do not specify a `targetWidth`, `targetHeight`, and `quality` for the picture being taken, the default values used are maximum values, and this can cause memory failures.

- `quality`: Set the quality of the saved image. Range is 0 to 100, inclusive. A higher number indicates higher quality, but also increases the file size. Only applicable to JPEG images (specified by `encodingType`).
- `destinationType`: Choose the format of the return value:
  - `DeviceManager.CAMERA_DESTINATIONTYPE_DATA_URL (0)`—Returns the image as a Base64-encoded string. This value is also specified as an `enum` using `DeviceManager.CAMERA_DESTINATION_DATA_URL` when used programmatically. You need to prefix the value returned with `""` in order to see the image in an image component.
  - `DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URI (1)`—Returns the image file path. This value is also specified as an enum using DeviceManager.`CAMERA_DESTINATION_FILE_URI` when used programmatically.

Note: If a file URI is returned by the `getPicture` method, it should be stripped of any query parameters before being used to determine the size of the file. For example:

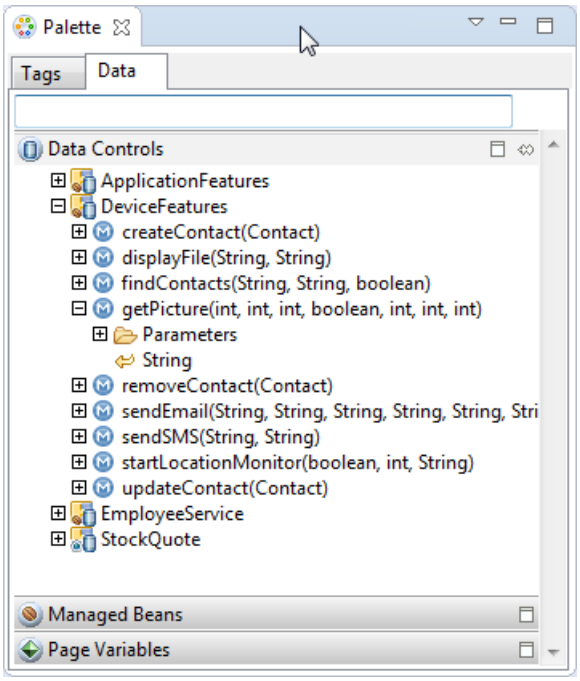
```
String fileURI = ...getPicture(...);

fileURI = fileURI.substring(0, result.lastIndexOf("?"));
```

- `sourceType`: Set the source of the picture:
  - `DeviceManager.CAMERA_SOURCETYPE_PHOTOLIBRARY (0)`—Enables the user to choose from a previously saved image. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_PHOTOLIBRARY` when used programmatically.
  - `DeviceManager.CAMERA_SOURCETYPE_CAMERA (1)`—Enables the user to take a picture with device's camera. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_CAMERA` when used programmatically.
  - `DeviceManager.CAMERA_SOURCETYPE_SAVEDPHOTOALBUM (2)`—Allows the user to choose from an existing photo album. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_SAVEDPHOTOALBUM` when used programmatically.
- `allowEdit`: Choose whether to allow simple editing of the image before selection (boolean).
- `encodingType`: Choose the encoding of the returned image file:
  - `DeviceManager.CAMERA_ENCODINGTYPE_JPEG` (0)—Encodes the returned image as a JPEG file. This value is also specified as an enum using `DeviceManager.CAMERA_ENCODINGTYPE_JPEG` when used programmatically.
  - `DeviceManager.CAMERA_ENCODINGTYPE_PNG` (1)—Encodes the returned image as a PNG file. This value is also specified as an enum using `DeviceManager.CAMERA_ENCODINGTYPE_PNG` when used programmatically.
- `targetWidth`: Set the width in pixels to scale the image. Aspect ratio is maintained. A negative or zero value indicates that the original dimensions of the image will be used.
- `targetHeight`: Set the height in pixels to scale the image. Aspect ratio is maintained. A negative or zero value indicates that the original dimensions of the image will be used.

To customize a getPicture operation using the DeviceFeatures data control:

1. Drag the **getPicture** operation from the **Palette > Data Tab > Data Controls** and drop it on the page as a **Button**, as shown in Figure 13–13.

Figure 13–13 Customizing the getPicture operation

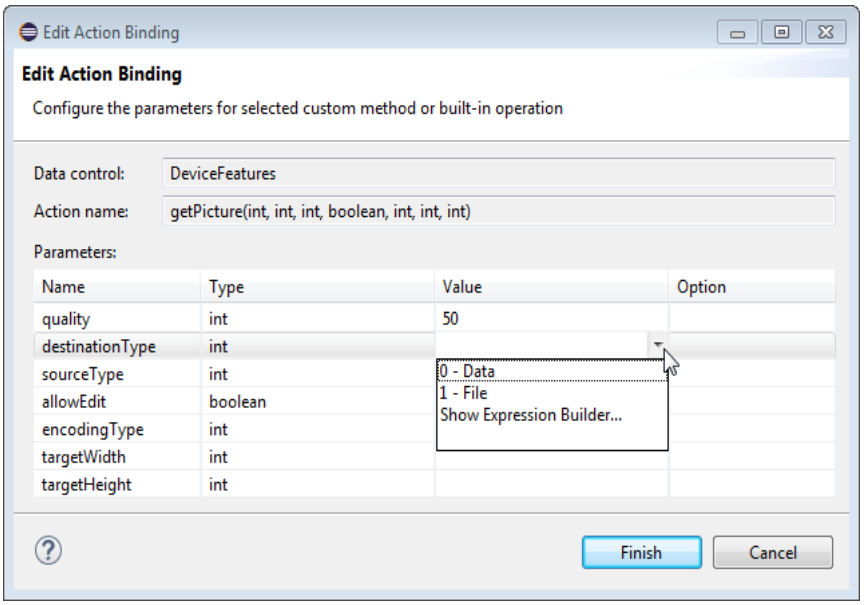


If you want to provide more control to the user, drop the **getPicture** operation as a **Parameter Form**. This allows the end user to specify settings before taking a picture or choosing an existing image.

2. In the Edit Action dialog, set the values for all parameters described above. Be sure to specify `destinationType = 1` so that the image is returned as a filename.
3. Drag the return value of **getPicture** and drop it on the page as an **Image**.
4. From the Common Components panel, drag an **Image** from the Component Palette and drop it on the page.

Figure 13–14 shows the bindings for displaying an image from the end user's photo library:

Figure 13–14 Bindings for Displaying an Image from the Photo Library at Design Time



When this application is run, the image chooser will automatically be displayed and the end user can select an image to display. The image chooser is displayed automatically because the Image control is bound to the return value of the `getPicture` operation, which in turn causes the `getPicture` operation to be invoked.

Note: The timeout value for the `getPicture` method is set to 5 minutes. If the device operation takes longer than the timeout allowed, a timeout error is displayed.

Keep in mind the following platform-specific issues:

- iOS
  - Set quality below 50 to avoid memory error on some devices.
  - When `destinationType` `FILE_URI` is used, photos are saved in the application's temporary directory.
  - The contents of the application's temporary directory are deleted when the application ends. You may also delete the contents of this directory using the `navigator.fileMgr` APIs if storage space is a concern.
  - `targetWidth` and `targetHeight` must both be specified to be used. If one or both parameters have a negative or zero value, the original dimensions of the image will be used.
- Android
  - Ignores the `allowEdit` parameter.
  - `Camera.PictureSourceType.PHOTOLIBRARY` and `Camera.PictureSourceType.SAVEDPHOTOALBUM` both display the same photo album.
  - `Camera.EncodingType` is not supported. The parameter is ignored, and will always produce JPEG images.

- `targetWidth` and `targetHeight` can be specified independently. If one parameter has a positive value and the other uses a negative or zero value to represent the original size, the positive value will be used for that dimension, and the other dimension will be scaled to maintain the original aspect ratio.
- When `destinationType` `DATA_URL` is used, large images can exhaust available memory, producing an out-of-memory error, and will typically do so if the default image size is used. Set the `targetWidth` and `targetHeight` to constrain the image size.

This example shows JavaScript code that allows the user to take a picture with a device's camera. The result will be the full path to the saved image.

```
// The camera, like many other device-specific features, is accessed
// from the global 'navigator' object in JavaScript.
// Note that in the Cordova JavaScript APIs, the parameters are passed
// in as a dictionary, so it is only necessary to provide key-value pairs
// for the parameters you want to specify.

navigator.camera.getPicture(onSuccess, onFail, { quality: 50 });

function onSuccess(imageURI) {
    var image = document.getElementById('myImage');
    image.src = imageURI;
}
function onFail(message) {
    alert('Failed because: ' + message);
}
```

This example shows Java code that allows the user to take a picture with a device's camera. The result will be the full path to the saved image.

```
import oracle.adf.model.datacontrols.device;

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Take a picture with the device's camera.
// The result will be the full path to the saved PNG image.
String imageFilename = DeviceManagerFactory.getDeviceManager().getPicture(100,
    DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URI,
    DeviceManager.CAMERA_SOURCETYPE_CAMERA, false,
    DeviceManager.CAMERA_ENCODINGTYPE_PNG, 0, 0);
```

This example shows Java code that allows the user to retrieve a previously-saved image. The result will be a base64-encoded JPEG.

```
import oracle.adf.model.datacontrols.device;

// Retrieve a previously-saved image. The result will be a base64-encoded JPEG.
String imageData = DeviceManagerFactory.getDeviceManager().getPicture(100,
  DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URL,
  DeviceManager.CAMERA_SOURCETYPE__PHOTOLIBRARY, false,
  DeviceManager.CAMERA_ENCODINGTYPE_JPEG, 0, 0);
```

13.10.2 How to Use the SendSMS Method to Enable Text Messaging

The DeviceFeatures data control includes the `sendSMS` method, which enables MAF applications to leverage a device's Short Message Service (SMS) text messaging

interface so end users can send and receive SMS messages. MAF enables you to display a device's SMS interface and optionally pre-populate the following fields:

- `to`: List recipients (comma-separated).
- `body`: Add message body.

After the SMS text messaging interface is displayed, the end user can choose to either send the SMS or discard it. It is not possible to automatically send the SMS due to device and carrier restrictions; only the end user can actually send the SMS.

Note: The timeout value for the `sendSMS` method is set to 5 minutes. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

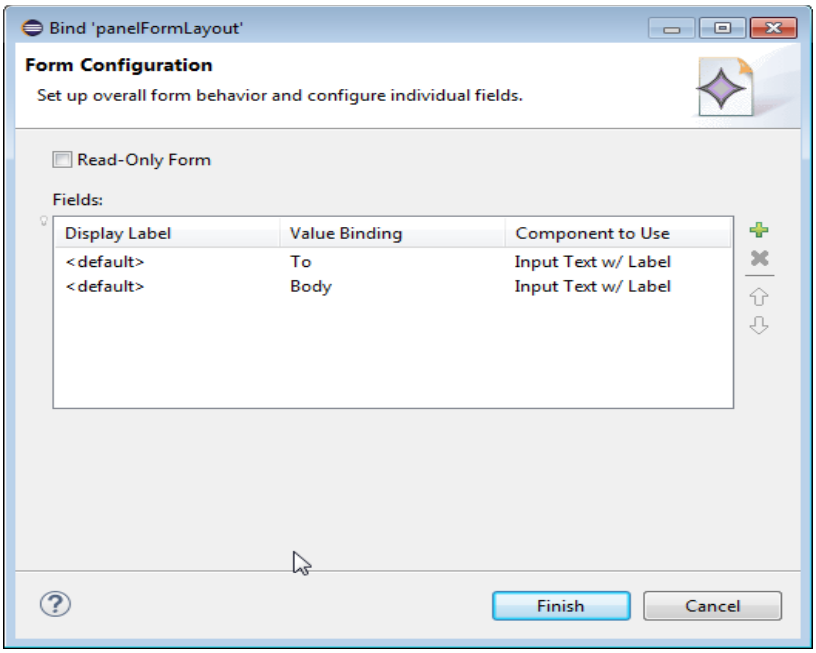
Note: In Android, if an end user switches away from their application while editing an SMS message and then subsequently returns to it, they will no longer be in the SMS editing screen. Instead, that message will have been saved as a draft that can then manually be selected for continued editing.

To customize a sendSMS operation using the DeviceFeatures data control:

To display an interactive form on the page for sending SMS, drag the `sendSMS` operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Parameter Form**. You can then customize the form in the **Edit Form Fields** dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various fields described above. Below this form will be a button to display the device's SMS interface, which will display an SMS that is ready to send with all of the specified fields pre-populated.

Figure 13–15 shows the bindings for sending an SMS using an editable form on the page.

Figure 13–15 Bindings for Sending an SMS Using an Editable Form at Design Time



The examples below show code examples that allow the end user to send an SMS message with a device's text messaging interface.

For information about the `sendSMS` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

Javascript code sample for sendSMS.

```
adf.mf.api.sendSMS({to: "5551234567", body: "This is a test message"});
```

Java code sample for sendSMS.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Send an SMS to the phone number "5551234567"
DeviceManagerFactory.getDeviceManager().sendSMS("5551234567", "This is a test
message");
```

13.10.3 How to Use the sendEmail Method to Enable Email

The DeviceFeatures data control includes the `sendEmail` method, which enables MAF applications to leverage a device's email messaging interface so end users can send and receive email messages. MAF enables you to display a device's email interface and optionally pre-populate the following fields:

- `to`: List recipients (comma-separated).
- `cc`: List CC recipients (comma-separated).
- `subject`: Add message subject.
- `body`: Add message body.
- `bcc`: List BCC recipients (comma-separated).

- `attachments`: List file names to attach to the email (comma-separated).
- `mimeTypes`: List MIME types to use for the attachments (comma-separated). Specify null to let MAF automatically determine the MIME types. It is also possible to specify only the MIME types for selected attachments as shown in the two examples at the end of this section.

After the device's email interface is displayed, the user can choose to either send the email or discard it. It is not possible to automatically send the email due to device and carrier restrictions; only the end user can actually send the email. The device must also have at least one email account configured to send email or an error will be displayed indicating that no email accounts could be found.

Note: The timeout value for the `sendEmail` method is set to 5 minutes. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

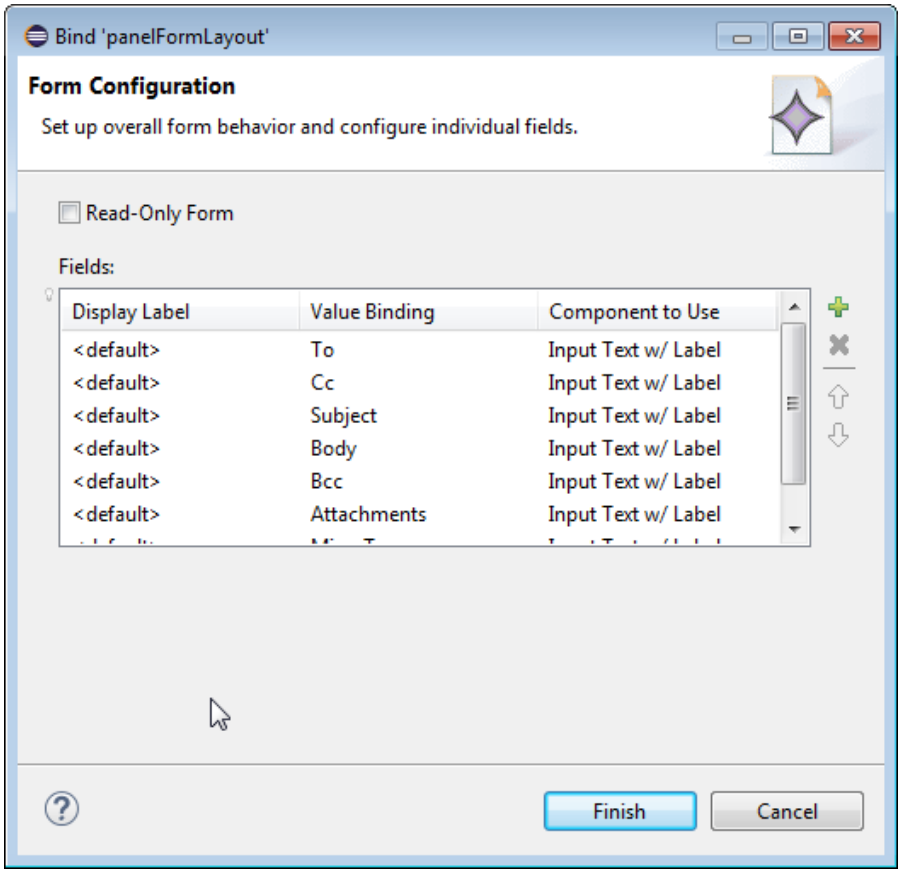
Note: In Android, if an end user switches away from their application while editing an email and then subsequently returns to it, they will no longer be in the email editing screen. Instead, the message will be saved as a draft that can then be manually selected for continued editing.

To customize a sendEmail operation using the DeviceFeatures data control:

In OEPE, drag the `sendEmail` operation from the DeviceFeatures data control in the Paletteto the page designer and drop it as a **Parameter Form**. You can then customize the form in the **Edit Form Fields** dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various fields described above. Below this form will be a button to display the device's email interface, which will display an email ready to send with all of the specified fields pre-populated.

Figure 13–16 shows the bindings for sending an email using an editable form on the page.

Figure 13–16 Bindings for Sending an Email Using an Editable Form at Design Time



These examples show code examples that allow the end user to send an email message with the device's email interface.

For information about the `sendEmail` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

Javascript code sample for sendEmail:

```
// Populate an email to 'ann.li@example.com',
// copy 'joe.jones@example.com', with the
// subject 'Test message', and the body 'This is a test message'
// No BCC recipients or attachments
adf.mf.api.sendEmail({to: "ann.li@example.com",
                          cc: "joe.jones@example.com",
                          subject: "Test message",
                          body: "This is a test message"});

// Populate the same email as before, but this time, also BCC
// 'john.smith@example.com' & 'jane.smith@example.com' and attach two files.
// By not specifying a value for the mimeTypes parameter, you are telling
// ADFMobile to automatically determine the MIME type for each of the attachments.
adf.mf.api.sendEmail({to: "ann.li@example.com",
                          cc: "joe.jones@example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@example.com,jane.smith@example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
```

```
// For iOS only: Same as previous email, but this time, explicitly specify
// all the MIME types.
adf.mf.api.sendEmail({to: "ann.li@example.com",
                          cc: "joe.jones@example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@example.com,jane.smith@example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: "text/plain,image/png"});

// For iOS only: Same as previous email, but this time, only specify
// the MIME type for the second attachment and let the system determine
// the MIME type for the first one.
adf.mf.api.sendEmail({to: "ann.li@example.com",
                          cc: "joe.jones@example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@example.com,jane.smith@example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: ",image/png"});

// For Android only: Same as previous e-mail, but this time, explicitly specify
// the MIME type.
adf.mf.api.sendEmail({to: "ann.li@example.com",
                          cc: "joe.jones@example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@example.com,jane.smith@example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: "image/*"});
// You can also use "plain/text" as the MIME type as it just determines the type
// of applications to be filtered in the application chooser dialog.
```

Java code sample for sendEmail:

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Populate an email to 'ann.li@example.com', copy 'joe.jones@example.com', with
the
// subject 'Test message', and the body 'This is a test message'.
// No BCC recipients or attachments.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@example.com",
                                        "joe.jones@example.com",
                                        "Test message",
                                        "This is a test message",
                                        null,
                                        null,
                                        null);

// Populate the same email as before, but this time, also BCC
// 'john.smith@example.com' & 'jane.smith@example.com' and attach two files.
// By specifying null for the mimeTypes parameter, you are telling
// ADFMobile to automatically determine the MIME type for each of the attachments.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@example.com",
                                        "joe.jones@example.com",
                                        "Test message",
                                        "This is a test message",
```

```
                                        "john.smith@example.com,jane.smith@example
.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        null);

// Same as previous email, but this time, explicitly specify all the MIME types.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@example.com",
                                        "joe.jones@example.com",
                                        "Test message",
                                        "This is a test message",
                                        "john.smith@example.com,jane.smith@example
.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        "text/plain,image/png");


// Same as previous email, but this time, only specify the MIME type for the
// second attachment and let the system determine the MIME type for the first one.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@example.com",
                                        "joe.jones@example.com",
                                        "Test message",
                                        "This is a test message",
                                        "john.smith@example.com,jane.smith@example
.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        ",image/png");
```

13.10.4 How to Use the createContact Method to Enable Creating Contacts

The DeviceFeatures data control includes the `createContact` method, which enables MAF applications to leverage a device's interface and file system for managing contacts so end users can create new contacts to save in the device's address book. MAF enables you to display the device's interface and optionally pre-populate the `Contact` fields. The `createContact` method takes in a `Contact` object as a parameter and returns the created `Contact` object, as shown in the second of the three examples at the end of this section.

For more information about the `createContact` method and the `Contact` object, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`). Also see Section 13.10.5, "How to Use the findContacts Method to Enable Finding Contacts" for a description of `Contact` properties.

Note: The timeout value for the `createContact` method is set to 1 minute. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

Note: If a null `Contact` object is passed in to the method, an exception is thrown.

To customize a createContact operation using the DeviceFeatures data control:

1. In OEPE, drag the `createContact` operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Link** or **Button**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter the `Contact` object parameter to the `createContact` operation. This parameter must be an EL expression that refers to the property of a managed bean that is used to return the `Contact` from a JavaBean class. Assuming a managed bean already exists with a getter for a `Contact` object, you can use the EL Expression Builder to set the value of the parameter. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `createContact` operation when pressed. The next example shows an example of managed bean code for creating a `Contact` object.

2. You can also drag a `Contact` return object from under the `createContact` operation in the Palette and drop it on to the page as a **Form**. You can then customize the form in the **Edit Form Fields** dialog. When the `createContact` operation is performed, the results will be displayed in this form.

This example shows managed bean code for creating a contact object.

```
private Contact contactToBeCreated;

public void setContactToBeCreated(Contact contactToBeCreated)
{
   this.contactToBeCreated = contactToBeCreated;
}

public Contact getContactToBeCreated()
{
   String givenName = "Mary";
   String familyName = "Jones";
   String note = "Just a Note";
   String phoneNumberType = "mobile";
   String phoneNumberValue = "650-555-0111";
   String phoneNumberNewValue = "650-555-0199";
   String emailType = "home";
   String emailTypeNew = "work";
   String emailValue = "Mary.Jones@example.com";
   String addressType = "home";
   String addressStreet = "500 Barnacle Pkwy";
   String addressLocality = "Redwood Shores";
   String addressCountry = "USA";
   String addressPostalCode = "94065";
   ContactField[] phoneNumbers = null;
   ContactField[] emails = null;
   ContactAddresses[] addresses = null;

   /*
    * Create contact
    */
   this.contactToBeCreated = new Contact();

   ContactName name = new ContactName();
   name.setFamilyName(familyName);
   name.setGivenName(givenName);
   this.contactToBeCreated.setName(name);

   ContactField phoneNumber = new ContactField();
   phoneNumber.setType(phoneNumberType);
   phoneNumber.setValue(phoneNumberValue);
```

```
   phoneNumbers = new ContactField[] { phoneNumber };

   ContactField email = new ContactField();
   email.setType(emailType);
   email.setValue(emailValue);

   emails = new ContactField[] { email };

   ContactAddresses address = new ContactAddresses();
   address.setType(addressType);
   address.setStreetAddress(addressStreet);
   address.setLocality(addressLocality);
   address.setCountry(addressCountry);

   addresses = new ContactAddresses[] { address };

   this.contactToBeCreated.setNote(note);
   this.contactToBeCreated.setPhoneNumbers(phoneNumbers);
   this.contactToBeCreated.setEmails(emails);
   this.contactToBeCreated.setAddresses(addresses);

   return this.contactToBeCreated;
}
```

This example shows JavaScript code for createContact.

```
// Contacts, like many other device-specific features, are accessed from the
global 'navigator' object in JavaScript.
var contact = navigator.contacts.create();

var name = new ContactName();
name.givenName = "Mary";
name.familyName = "Jones";

contact.name = name;

// Store contact phone numbers in ContactField[]
var phoneNumbers = [1];
phoneNumbers[0] = new ContactField('home', '650-555-0123', true);

contact.phoneNumbers = phoneNumbers;

// Store contact email addresses in ContactField[]
var emails = [1];
emails[0] = new ContactField('work', 'Mary.Jones@example.com');

contact.emails = emails;

// Save
contact.save(onSuccess, onFailure);

function onSuccess()
{
  alert("Create Contact successful.");
}

function onFailure(Error)
{
  alert("Create Contact failed: " + Error.code);
}
```

This example shows Java code for createContact.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

import oracle.adf.model.datacontrols.device.ContactAddresses;
import oracle.adf.model.datacontrols.device.ContactField;
import oracle.adf.model.datacontrols.device.ContactName;

String givenName = "Mary";
String familyName = "Jones";
String note = "Just a Note";
String phoneNumberType = "mobile";
String phoneNumberValue = "650-555-0111";
String phoneNumberNewValue = "650-555-0199";
String emailType = "home";
String emailTypeNew = "work";
String emailValue = "Mary.Jones@example.com";
String addressType = "home";
String addressStreet = "500 Barnacle Pkwy";
String addressLocality = "Redwood Shores";
String addressCountry = "USA";
String addressPostalCode = "91234";
ContactField[] phoneNumbers = null;
ContactField[] emails = null;
ContactAddresses[] addresses = null;
ContactField[] emails = null;

/*
* Create contact
*/
Contact aContact = new Contact();

ContactName name = new ContactName();
name.setFamilyName(familyName);
name.setGivenName(givenName);
aContact.setName(name);

ContactField phoneNumber = new ContactField();
phoneNumber.setType(phoneNumberType);
phoneNumber.setValue(phoneNumberValue);

phoneNumbers = new ContactField[] { phoneNumber };

ContactField email = new ContactField();
email.setType(emailType);
email.setValue(emailValue);

emails = new ContactField[] { email };

ContactAddresses address = new ContactAddresses();
address.setType(addressType);
address.setStreetAddress(addressStreet);
address.setLocality(addressLocality);
address.setCountry(addressCountry);

addresses = new ContactAddresses[] { address };

aContact.setNote(note);
aContact.setPhoneNumbers(phoneNumbers);
aContact.setEmails(emails);
aContact.setAddresses(addresses);
```

```
// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Invoking the createContact method, using the DeviceDataControl object.
Contact createdContact = DeviceManagerFactory.getDeviceManager()
    .findContacts.createContact(aContact);
```

13.10.5 How to Use the findContacts Method to Enable Finding Contacts

The DeviceFeatures data control includes the `findContacts` method, which enables MAF applications to leverage a device's interface and file system for managing contacts so end users can find one or more contacts from the device's address book. MAF enables you to display the device's interface and optionally pre-populate the `findContacts` fields. The `findContacts` method takes in a filter string and a list of field names to look through (and return as part of the found contacts). The filter string can be anything to look for in the contacts. For more information about the `findContacts` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

The `findContacts` operation takes the following arguments:

- `contactFields`: *Required parameter*. Use this parameter to specify which fields should be included in the `Contact` objects resulting from a `findContacts` operation. Separate fields with a comma (spacing does not matter).
- `filter`: The search string used to filter contacts. (String) (Default: `""`)
- `multiple`: Determines if the `findContacts` operation should return multiple contacts. (Boolean) (Default: `false`)

> **Note:** Passing in a field name that is not in the following list may result in a `null` return value for the `findContacts` operation. Also, only the fields specified in the `Contact` fields argument will be returned as part of the `Contact` object.

The following list shows the possible `Contact` properties that can be passed in to look through and be returned as part of the found contacts:

- `id`: A globally unique identifier
- `displayName`: The name of this contact, suitable for display to end-users
- `name`: An object containing all components of a person's name
- `nickname`: A casual name for the contact. If you set this field to null, it will be stored as an empty string.
- `phoneNumbers`: An array of all the contact's phone numbers
- `emails`: An array of all the contact's email addresses
- `addresses`: An array of all the contact's addresses
- `ims`: An array of all the contact's instant messaging (IM) addresses (The `ims` property is not supported in this release.)

> **Note:** MAF does not support the `Contact` property `ims` in this release. If you create a contact with the `ims` property, MAF will save the contact without the `ims` property. As a result, if a user tries to perform a search based on `ims`, the user will not be able to find the contact. Also, if a user tries to enter `ims` in a search field, the `ims` will be returned as `null`.

- `organizations`: An array of all the contact's organizations
- `birthday`: The birthday of the contact. Although you cannot programmatically set a contact's birthday field and persist it to the address book, you can still use the operating system's address book application to manually set this field.
- `note`: A note about the contact. If you set this field to null, it will be stored as an empty string.
- `photos`: An array of the contact's photos
- `categories`: An array of all the contact's user-defined categories.
- `urls`: An array of web pages associated to the contact

> **Note:** The timeout value for the `findContacts` method is set to 1 minute. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

To customize a findContacts operation using the DeviceFeatures data control:

1. In OEPE, drag the `findContacts` operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `findContacts` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `findContacts` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields described above. Below this form will be a button, which will use the entered values to perform a `findContacts` operation when pressed.

2. You can also drag a `Contact` return object from under the `findContacts` operation in the Palette and drop it on to the page as a **Form**. You can then customize the form in the Edit Form Fields dialog. When the `findContacts` operation is performed, the results will be displayed in this form.

The first example shows possible argument values for the `findContacts` method. The second and third examples show how to find a contact by family name and get the contact's name, phone numbers, email, addresses, and note.

This example shows possible argument values for findContacts.

```
// This will return just one contact with only the ID field:
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("",
"", false);

// This will return all contacts with only ID fields:
```

```
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("",
"", true);

// This will return just one contact with all fields:
Contact[] foundContacts =
DeviceManagerFactory.getDeviceManager().findContacts("*", "", false);

// This will return all contacts with all fields:
Contact[] foundContacts =
DeviceManagerFactory.getDeviceManager().findContacts("*", "", true);

// These will throw an exception as contactFields is a required argument and
cannot be null:
DeviceManagerFactory.getDeviceManager().findContacts(null, "", false);
DeviceManagerFactory.getDeviceManager().findContacts(null, "", true);

// These will throw an exception as the filter argument cannot be null:
DeviceManagerFactory.getDeviceManager().findContacts("", null, false);
DeviceManagerFactory.getDeviceManager().findContacts("", null, true);
```

Note: The `Contact` fields passed are strings (containing the comma-delimited fields). If any arguments are passed as `null` to the method, an exception is thrown.

This example shows JavaScript code for findContacts.

```
var filter = ["name", "phoneNumbers", "emails", "addresses", "note"];

var options = new ContactFindOptions();
options.filter="FamilyName";

// Contacts, like many other device-specific features, are accessed from
// the global 'navigator' object in JavaScript.
navigator.contacts.find(filter, onSuccess, onFail, options);

function onSuccess(contacts)
{
  alert ("Find Contact call succeeded! Number of contacts found = " +
contacts.length);
}

function onFail(Error)
{
  alert("Find Contact failed: " + Error.code);
}
```

This example shows Java code for findContacts.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
 * Find Contact - Find contact by family name.
 *
 * See if we can find the contact that we just created.
 */

String familyName = "FamilyName"

// Access device features in Java code by acquiring an instance of the
```

```
// DeviceManager from the DeviceManagerFactory.
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses,note", familyName, true);
```

13.10.6 How to Use the updateContact Method to Enable Updating Contacts

The DeviceFeatures data control includes the `updateContact` method, which enables MAF applications to leverage a device's interface and file system for managing contacts so end users can update contacts in the device's address book. MAF enables you to display the device's interface and optionally pre-populate the `updateContact` fields. The `updateContact` method takes in a `Contact` object as a parameter and returns the updated `Contact` object, as shown in the second of the three examples at the end of this section

For more information about the `updateContact` method and the `Contact` object, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`). Also see Section 13.10.5, "How to Use the findContacts Method to Enable Finding Contacts" for a description of `Contact` properties.

Note: The `Contact` object that is needed as the input parameter can be found using the `findContacts` method as described in Section 13.10.5, "How to Use the findContacts Method to Enable Finding Contacts." If a null `Contact` object is passed in to the method, an exception is thrown.

To customize an updateContact operation using the DeviceFeatures data control:

1. In OEPE, drag the **updateContact** operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Link** or **Button**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter the `Contact` object parameter to the `updateContact` operation. This parameter must be an EL expression that refers to the property of a managed bean that is used to return the `Contact` from a JavaBean class. Assuming a managed bean already exists with a getter for a `Contact` object, you can use the EL Expression Builder to set the value of the parameter. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `updateContact` operation when pressed. The first example in Section 13.10.4, "How to Use the createContact Method to Enable Creating Contacts" shows an example of managed bean code for creating a `Contact` object.

2. You can also drag a **Contact** return object from under the `updateContact` operation in the Palette and drop it on to the page as a **Form**. You can then customize the form in the Edit Form Fields dialog. When the `updateContact` operation is performed, the results will be displayed in this form.

The first and third examples below show how to update a contact's phone number. The second and fourth examples below show how to add another phone number to a contact.

This example shows JavaScript for updateContact.

```
function updateContact(contact)
{
  try
  {
    if (null != contact.phoneNumbers)
```

```
    {
      alert("Number of phone numbers = " + contact.phoneNumbers.length);
      var numPhoneNumbers = contact.phoneNumbers.length;
      for (var j = 0; j < numPhoneNumbers; j++)
      {
        alert("Type: " + contact.phoneNumbers[j].type + "\n" +
              "Value: "  + contact.phoneNumbers[j].value + "\n" +
              "Preferred: "  + contact.phoneNumbers[j].pref);

        contact.phoneNumbers[j].type = "mobile";
        contact.phoneNumbers[j].value = "408-555-0100";
      }

      // save
      contact.save(onSuccess, onFailure);
    }
    else
    {
      //alert ("No phone numbers found in the contact.");
    }
  }
  catch(e)
  {
    alert("updateContact - ERROR: " + e.description);
  }
}

function onSuccess()
{
  alert("Update Contact successful.");
}

function onFailure(Error)
{
  alert("Update Contact failed: " + Error.code);
```

This example, JavaScript code for adding a phone number with updateContact, shows you how to add another phone number to the already existing phone numbers.

```
function updateContact(contact)
{
  try
  {
    var phoneNumbers = [1];
    phoneNumbers[0] = new ContactField('home', '650-555-0123', true);
    contact.phoneNumbers = phoneNumbers;

    // save
    contact.save(onSuccess, onFailure);
  }
  catch(e)
  {
    alert("updateContact - ERROR: " + e.description);
  }
}

function onSuccess()
{
  alert("Update Contact successful.");
}
```

```
function onFailure(Error)
{
  alert("Update Contact failed: " + Error.code);
}
```

This example, Java code for updateContact, shows how to update a contact's phone number, email type, and postal code.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
 * Update Contact - Updating phone number, email type, and adding address postal
code
 */
String familyName = "FamilyName";
String phoneNumberNewValue = "650-555-0123";
String emailTypeNew = "work";
String addressPostalCode = "91234";

Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses,note", familyName, true);

// Assuming there was only one contact returned, we can use the first contact in
the array.
// If more than one contact is returned then we have to filter more to find the
exact contact
// we need to update.

foundContacts[0].getPhoneNumbers()[0].setValue(phoneNumberNewValue);
foundContacts[0].getEmails()[0].setType(emailTypeNew);
foundContacts[0].getAddresses()[0].setPostalCode(addressPostalCode);

Contact updatedContact =
DeviceManagerFactory.getDeviceManager().updateContact(foundContacts[0]);
```

This example, Java code for adding a phone number with updateContact, shows you how to add another phone number to the already existing phone numbers.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

String additionalPhoneNumberValue = "408-555-0123";
String additionalPhoneNumberType = "mobile";
// Create a new phoneNumber that will be appended to the previous one.
ContactField additionalPhoneNumber = new ContactField();
additionalPhoneNumber.setType(additionalPhoneNumberType);
additionalPhoneNumber.setValue(additionalPhoneNumberValue);

foundContacts[0].setPhoneNumbers(new ContactField[] { additionalPhoneNumber });

// Access device features in Java code by acquiring an instance of the
DeviceManager
// from the DeviceManagerFactory.
Contact updatedContact =
DeviceManagerFactory.getDeviceManager().updateContact(foundContacts[0]);
```

Note: The timeout value for the `updateContact` method is set to 1 minute. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

13.10.7 How to Use the removeContact Method to Enable Removing Contacts

The DeviceFeatures data control includes the `removeContact` method, which enables MAF applications to leverage a device's interface and file system for managing contacts so end users can remove contacts from the device's address book. MAF enables you to display the device's interface and optionally pre-populate the `removeContact` fields. The `removeContact` method takes in a `Contact` object as a parameter, as shown in the first example in this section.

Note: The `Contact` object that is needed as the input parameter can be found using the `findContacts` method as described in Section 13.10.5, "How to Use the findContacts Method to Enable Finding Contacts."

To customize a removeContact operation using the DeviceFeatures data control:

1. In OEPE, drag the **removeContact** operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `removeContact` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `removeContact` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields. Below this form will be a button, which will use the entered values to perform a `removeContact` operation when pressed.

2. You can also drag a `Contact` return object from under the `removeContact` operation in the Palette and drop it on to the page as a **Form**. You can then customize the form in the Edit Form Fields dialog. When the `removeContact` operation is performed, the results will be displayed in this form.

These examples show you how to delete a contact that you found using `findContacts`. For information about the `removeContact` method and the Contact object, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

Note: In Android, the `removeContact` operation does not remove the contact fully. After a contact is removed by calling the `removeContact` method, a contact with the "(Unknown)" display name shows in the contacts list in the application.

This example shows JavaScript for removeContact.

```
// Remove the contact from the device
contact.remove(onSuccess,onError);
```

```
function onSuccess()
{
  alert("Removal Success");
}

function onError(contactError)'
{
  alert("Error = " + contactError.code);
}
```

This example shows Java code for removeContact.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
 * Remove the contact from the device
 */
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses", familyName, true);

// Assuming there is only one contact returned, we can use the first contact in
the array.
// If more than one contact is returned we will have to filter more to find the
// exact contact we want to remove.

// Access device features in Java code by acquiring an instance of the
DeviceManager
// from the DeviceManagerFactory.
DeviceManagerFactory.getDeviceManager().removeContact(foundContacts[0]);
```

Note: The timeout value for the `removeContact` method is set to 1 minute. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

13.10.8 How to Use the startLocationMonitor Method to Enable Geolocation

The DeviceFeatures data control includes the `startLocationMonitor` method, which enables MAF applications to leverage a device's geolocation services in order to obtain and track the device's location. MAF enables you to display a device's interface and optionally pre-populate the `startLocationMonitor` fields.

MAF exposes APIs that enable you to acquire a device's current position, allowing you to retrieve the device's current location for one instant in time or to subscribe to it on a periodic basis. The examples at the end of this section show code examples that will allow your application to obtain the device's location. For information about the `startLocationMonitor` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

Note: The `altitudeAccuracy` property is not supported by Android devices.

Updates do not occur as frequently on the Android platform as on iOS.

To listen for changes in a device's location using the DeviceFeatures data control:

In OEPE, drag the `startLocationMonitor` operation from the DeviceFeatures data control in the Palette to the page designer and drop it as a **Link** or **Button**. When prompted by the **Edit Action Dialog**, populate the fields with values for the parameters that the operation supports, as described in the following list or see the `DeviceDataControl` class's `startLocationMonitor` method in *Java API Reference for Oracle Mobile Application Framework*.

- `enableHighAccuracy`: If `true`, use the most accurate possible method of obtaining a location fix. This is just a hint; the operating system may not respect it. Devices often have several different mechanisms for obtaining a location fix, including cell tower triangulation, Wi-Fi hotspot lookup, and true GPS. Specifying `false` indicates that you are willing to accept a less accurate location, which may result in a faster response or consume less power.
- `updateInterval`: Defines how often, in milliseconds, to receive updates. Location updates may not be delivered as frequently as specified; the operating system may wait until a significant change in the device's position has been detected before triggering another location update.
- `locationListener`: EL expression that resolves to a bean method with the following signature:

  ```
  void methodName(Location newLocation)
  ```

  This EL expression will be evaluated every time a location update is received. For example, enter `viewScope.LocationListenerBean.locationUpdated` (without the surrounding`#{}`), then define a bean named `LocationListenerBean` in `viewScope` and implement a method with the following signature:

  ```
  public void locationUpdated(Location currentLocation)
  {
    System.out.println(currentLocation);
    // To stop subscribing to location updates, invoke the following:
    // DeviceManagerFactory.getDeviceManager().clearWatchPosition(
    //     currentLocation.getWatchId());
  }
  ```

  Note: Do not use the EL syntax `#{LocationListenerBean.locationUpdate}` to specify the `locationListener`, unless you truly want the result of evaluating that expression to be the name of the `locationListener`.

The example at the end of this section shows how to subscribe to changes in the device's location using the `DeviceManager.startUpdatingPosition` method. For more information about the parameters that this method takes, see *Java API Reference for Oracle Mobile Application Framework*.

For an example of how to subscribe to changes in the device's position using JavaScript, refer to the Cordova documentation (`http://cordova.apache.org/`).

Parameters returned in the callback function specified by the `locationListener` are as follows:

Parameters returned in the callback function specified by the `locationListener` are as follows:

- `double getAccuracy`—Accuracy level of the latitude and longitude coordinates in meters
- `double getAltitude`—Height of the position in meters above the ellipsoid
- `double getLatitude`—Latitude in decimal degrees
- `double getLongitude`—Longitude in decimal degrees
- `double getAltitudeAccuracy`—Accuracy level of the altitude coordinate in meters
- `double getHeading`—Direction of travel, specified in degrees counting clockwise relative to the true north
- `double getSpeed`—Current ground speed of the device, specified in meters per second
- `long getTimestamp`—Creation of a timestamp in milliseconds since the Unix epoch
- `String getWatchId`—Only used when subscribing to periodic location updates. A unique ID that can be subsequently used to stop subscribing to location updates

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;
import oracle.adf.model.datacontrols.device.GeolocationCallback;
import oracle.adf.model.datacontrols.device.Location;

// Subscribe to location updates that will be delivered every 20 seconds, with
high accuracy.
// As you can have multiple subscribers, let's identify this one as
'MyGPSSubscriptionID'.
// Notice that this call returns the watchID, which is usually the same as the
watchID passed in.
// However, it may be different if the specified watchID conflicts with an
existing watchID,
// so be sure to always use the returned watchID.
String watchID =
DeviceManagerFactory.getDeviceManager().startUpdatingPosition(20000, true, "
       "MyGPSSubscriptionID", new GeolocationCallback() {
    public void locationUpdated(Location position) {
        System.out.println("Location updated to: " + position);
    }
});

// The previous call returns immediately so that you can continue processing.
// When the device's location changes, the locationUpdated() method specified in
// the previous call will be invoked in the context of the current feature.

// When you wish to stop being notified of location changes, call the following
method:
DeviceManagerFactory().getDeviceManager().clearWatchPosition(watchID);
```

For more information about the `startLocationMonitor` and `startHeadingMonitor` methods, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework* and refer to the Cordova documentation (`http://cordova.apache.org`).

The following example shows how to get a device's current location (one time) using the DeviceManager.getCurrentPosition. For information about the parameters that this method accepts, see *Java API Reference for Oracle Mobile Application Framework*.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;
```

```
import oracle.adf.model.datacontrols.device.Location;

// Get the device's current position, with highest accuracy, and accept a cached
location that is
// no older than 60 seconds.
Location currentPosition =
DeviceManagerFactory.getDeviceManager().getCurrentPosition(60000, true);
System.out.println("The device's current location is: latitude=" +
currentPosition.getLatitude() +
    ", longitude=" + currentPosition.getLongitude());
```

13.10.9 How to Use the displayFile Method to Enable Displaying Files

The DeviceFeatures data control includes the `displayFile` method, which enables MAF applications to display files that are local to the device. Depending on the platform, application users can view PDFs, image files, Microsoft Office documents, and various other file types. On iOS, the application user has the option to preview supported files within the MAF application. Users can also open those files with third-party applications, email them, or send them to a printer. On Android, all files are opened in third-party applications. In other words, the application user leaves the MAF application while viewing the file. The user may return to the MAF application by pressing the Android Back button. If the device does not have an application capable of opening the given file, an error is displayed. For an example of how the `displayFile` method opens files on both iOS- and Android-powered devices, see the DeviceDemo sample application. This application is available from **File** > **New** > **MAF Examples**.

The `displayFile` method is only able to display files that are local to the device. This means that remote files first have to be downloaded. Use the call `AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.`*`DownloadDirectory`*) to return the directory root where downloaded files should be stored. Note that on iOS, this location is specific to the application, but on Android this location refers to the external storage directory. The external storage directory is publicly accessible and allows third-party applications to read files stored there.

Table 13–6 Supported File Types

iOS	Android
For more information about supported file types, see the Quick Look preview controller documentation at the Apple iOS development site (http://developer.apple.com/library/ios/navigation/).	The framework will start the viewer associated with the given MIME type if it is installed on the device. There is no built-in framework for viewing specific file types. If the device does not have an application installed that handles the file type, the MAF application displays an error.
iWork documents	
Microsoft Office documents (Office '97 and newer)	
Rich Text Format (RTF) documents	
PDF files	
Images	
Text files whose uniform type identifier (UTI) conforms to the public.text type	
Comma-separated value (csv) files	

To customize a displayFile operation using the DeviceFeatures data control:

1. In OEPE, drag the **displayFile** operation from the DeviceFeatures data control in the Palette and drop it on the page designer as a **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `displayFile` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `displayFile` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various fields. Below this form will be a button, which will use the entered values to perform a `displayFile` operation when pressed.

The two examples below show you how to view files using `the displayFile` method. For information about the `displayFile` method, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework*).

Example showing Java code for displayFile.

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

 URL remoteFileUrl;
        InputStream is;
        BufferedOutputStream fos;
        try {

            // Open connection to remote file; fileUrl here is a String containing
the URL to the remote file.
            remoteFileUrl = new URL(fileUrl);
            URLConnection connection = remoteFileUrl.openConnection();
            is = new BufferedInputStream(connection.getInputStream());
            // Saving the file locally as 'previewTempFile.<extension>'
            String fileExt = fileUrl.substring(fileUrl.lastIndexOf('.'),
fileUrl.length());
            String tempFile = "/previewTempFile" + fileExt;
            File localFile = null;
            // Save the file in the DownloadDirectory location
            localFile = new
File(AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DownloadDirectory)
+ tempFile);
            if (localFile.exists()) {
                localFile.delete();
            }
            // Use buffered streams to download the file.
            fos = new BufferedOutputStream(new FileOutputStream(localFile));
            byte[] data = new byte[1024];
            int read = 0;
            while ((read = is.read(data)) != -1) {
                fos.write(data, 0, read);
            }
            is.close();
            fos.close();

            // displayFile takes a URL string which has to be encoded on iOS.
            // iOS does not handle "+" as an encoding for space (" ") but
            // expects "%20" instead.  Also, the leading slash MUST NOT be
            // encoded to "%2F".  We will revert it to a slash after the
            // URLEncoder converts it to "%2F".
            StringBuffer buffer = new StringBuffer();
            String path = URLEncoder.encode(localFile.getPath(), "UTF-8");
```

```
            // replace "+" with "%20"
            String replacedString = "+";
            String replacement = "%20";
            int index = 0, previousIndex = 0;
            index = path.indexOf(replacedString, index);
            while (index != -1) {
                buffer.append(path.substring(previousIndex,
index)).append(replacement);
                previousIndex = index + 1;
                index = path.indexOf(replacedString, index +
replacedString.length());
            }
            buffer.append(path.substring(previousIndex, path.length()));
            // Revert the leading encoded slash ("%2F") to a literal slash ("/").
            if (buffer.indexOf("%2F") == 0) {
                buffer.replace(0, 3, "/");
            }

            // Create URL and invoke displayFile with its String representation.
            URL localURL = null;
            if (Utility.getOSFamily() == Utility.OSFAMILY_ANDROID) {
                localURL = new URL("file", "localhost",
localFile.getAbsolutePath());
            }
            else if (Utility.getOSFamily() == Utility.OSFAMILY_IOS)
            {
                localURL = new URL("file", "localhost", buffer.toString());
            }

DeviceManagerFactory.getDeviceManager().displayFile(localURL.toString(), "remote
file");
        } catch (Throwable t) {
            System.out.println("Exception caught: " + t.toString());
        }
```

13.10.10 How to Use the addLocalNotification and cancelLocalNotification Methods to Manage Local Notifications

The DeviceFeatures data control includes the `addLocalNotification` and `cancelLocalNotification` methods, which enable MAF applications to leverage a device's interface for managing notifications so end users can schedule or cancel local notifications.

For information about using the `addLocalNotification` and `cancelLocalNotification` methods, see the `DeviceDataControl` class in the *Java API Reference for Oracle Mobile Application Framework*). For more information about managing local notifications, including code examples, see Section 24.3, "Managing Local Notifications." For general information about notifications, see Section 24.1, "Introduction to Notifications."

The LocalNotificationDemo sample application provides an example of how to schedule and receive local notifications within a MAF application. This sample application is available from File > New > MAF Examples.

For more information about sample applications, see Appendix E, "MAF Sample Applications."

13.10.11 What You May Need to Know About Device Properties

There may be features of your application that rely on specific device characteristics or capabilities. For example, you may want to present a different user interface depending on the device's screen orientation, or there may be a mapping feature that you want to enable only if the device supports geolocation. MAF provides a number of properties that you can access from Java, JavaScript, and EL in order to support this type of dynamic behavior. Table 13–7 lists these properties, along with information about how to query them, what values to expect in return, and whether the property can change during the application's lifecycle. The example at the end of this section shows an example of how you can access these properties using JavaScript.

Note: The timeout value for device properties is set to 1 minute. If the device's operation takes longer than the timeout allowed, a timeout error is displayed.

Table 13–7 Device Properties and Corresponding EL Expressions

Property	Static/ Dynamic	EL Expression	Sample Value	Java API
device.name	Static	#{deviceScope.device.name}	"iPhone Simulator", "Joe Smith's iPhone"	DeviceManager.getName()
device.platform	Static	#{deviceScope.device.platform}	"iPhone Simulator", "iPhone"	DeviceManager.getPlatform()
device.version	Static	#{deviceScope.device.version}	"4.3.2", "5.0.1"	DeviceManager.getVersion()
device.os	Static	#{deviceScope.device.os}	"iOS"	DeviceManager.getOs()
device.model	Static	#{deviceScope.device.model}	"x86_64", "i386", "iPhone3,1"	DeviceManager.getModel()
device.phonegap	Static	#{deviceScope.device.phonegap}	"1.0.0"	DeviceManager.getPhonegap()
hardware.hasCamera	Static	#{deviceScope.hardware.hasCamera}	"true", "false"	DeviceManager.hasCamera()
hardware.hasContacts	Static	#{deviceScope.hardware.hasContacts}	"true", "false"	DeviceManager.hasContacts()
hardware.hasTouchScreen	Static	#{deviceScope.hardware.hasTouchScreen}	"true", "false"	DeviceManager.hasTouchScreen()
hardware.hasGeolocation	Static	#{deviceScope.hardware.hasGeolocation}	"true", "false"	DeviceManager.hasGeolocation()
hardware.hasAccelerometer	Static	#{deviceScope.hardware.hasAccelerometer}	"true", "false"	DeviceManager.hasAccelerometer()
hardware.hasCompass	Static	#{deviceScope.hardware.hasCompass}	"true", "false"	DeviceManager.hasCompass()
hardware.hasFileAccess	Static	#{deviceScope.hardware.hasFileAccess}	"true", "false"	DeviceManager.hasFileAccess()
hardware.hasLocalStorage	Static	#{deviceScope.hardware.hasLocalStorage}	"true", "false"	DeviceManager.hasLocalStorage()

Table 13–7 (Cont.) Device Properties and Corresponding EL Expressions

Property	Static/ Dynamic	EL Expression	Sample Value	Java API
hardware.hasMediaPlayer	Static	#{deviceScope.hardware.hasMediaPlayer}	"true", "false"	DeviceManager.hasMediaPlayer()
hardware.hasMediaRecorder	Static	#{deviceScope.hardware.hasMediaRecorder}	"true", "false"	DeviceManager.hasMediaRecorder()
hardware.networkStatus	Dynamic	#{deviceScope.hardware.networkStatus}	"wifi", "2g", "unknown", "none"[1]	DeviceManager.getNetworkStatus()
hardware.screen.width	Dynamic	#{deviceScope.hardware.screen.width}	320, 480	DeviceManager.getScreenWidth()
hardware.screen.height	Dynamic	#{deviceScope.hardware.screen.height}	480, 320	DeviceManager.getScreenHeight()
hardware.availableWidth	Dynamic	#{deviceScope.hardware.screen.availableWidth}	<= 320, <= 480	DeviceManager.getAvailableScreenWidth()
hardware.availableHeight	Dynamic	#{deviceScope.hardware.screen.availableHeight}	<= 480, <= 320	DeviceManager.getAvailableScreenHeight()
hardware.screen.dpi	Static	#{deviceScope.hardware.screen.dpi}	160, 326	DeviceManager.getScreenDpi()
hardware.screen.diagonalSize	Static	#{deviceScope.hardware.screen.diagonalSize}	9.7, 6.78	DeviceManager.getScreenDiagonalSize()
hardware.screen.scaleFactor	Static	#{deviceScope.hardware.screen.scaleFactor}	1.0, 2.0	DeviceManager.getScreenScaleFactor()

[1] If both `wifi` and `2G` are turned on, network status will be `wifi`, as `wifi` takes precedence over `2G`.

This example illustrates how you can access device properties using JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Device Properties Example</title>

    <script type="text/javascript" charset="utf-8"
src="cordova-2.2.0.js"></script>
    <script type="text/javascript" charset="utf-8">

    // Wait for Cordova to load
    //
    //document.addEventListener("deviceready", onDeviceReady, false);
    document.addEventListener("showpagecomplete",onDeviceReady,false);

    // Cordova is ready
    //
    function onDeviceReady() {
        adf.mf.api.getDeviceProperties(properties_success, properties_fail);
    }

    function properties_success(response) {
      try {
        var element = document.getElementById('deviceProperties');
        var device = response.device;
```

```
        var hardware = response.hardware;
        element.innerHTML = 'Device Name:              ' + device.name
       + '<br />' +
                           'Device Platform:          ' + device.platform
       + '<br />' +
                           'Device Version:           ' + device.version
       + '<br />' +
                           'Device OS:                ' + device.os
       + '<br />' +
                           'Device Model:             ' + device.model
       + '<br />' +
                           'Hardware Screen Width:    ' + hardware.screen.width
+ '<br />' +
                           'Hardware Screen Height:   ' + hardware.screen.height
+ '<br />' +
      } catch (e) {alert("Exception: " + e);}
    }

    function properties_fail(error) {
        alert("getDeviceProperties failed");
    }

    </script>
  </head>
  <body>
    <p id="deviceProperties">Loading device properties...</p>
  </body>
</html>
```

13.11 Validating Attributes

In the Mobile Application Framework, validation occurs in the data control layer, with validation rules set on binding attributes. Attribute validation takes place at a single point in the system, during the `setValue` operation on the bindings.

You can define the following validators for attributes exposed by the data controls:

- Compare validator
- Length validator
- List validator
- Range validator

All validators for a given attribute are executed, and nested exceptions are thrown for every validator that does not pass. You can define a validation message for attributes, which is displayed when a validation rule is fired at runtime. For more information, see Section 12.9, "Validating Input."

> **Note:** Due to a JSON limitation, the value that a `BigDecimal` can hold is within the range of a `Double`, and the value that a `BigInteger` can hold is within the range of a `Long`. If you want to use numbers greater than those allowed, you can call `toString` on `BigDecimal` or `BigInteger` to (de)serialize values as `String`.

Table 13–8 lists supported validation combinations for the length validator.

Table 13–8 Length Validation

Compare type	Byte	Character
Equals	Supported	Supported
Not Equals	Supported	Supported
Less Than	Supported	Supported
Greater Than	Supported	Supported
Less Than Equal To	Supported	Supported
Greater Than Equal To	Supported	Supported
Between	Supported	Supported

Table 13–9 and Table 13–10 list supported validation combinations for the range validator.

Table 13–9 Range Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short
Between	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Not Between	Supported	Supported	Supported	Supported	Supported	Supported	Supported

Table 13–10 Range Validation - math, sql, and util Packages

Compare type	java.math.BigDecimal	java.math. BigInteger	java.sql.Date	java.sql.Time	java.sql.Time stamp	java.util.Date
Between	Supported	Supported	Not supported	Not supported	Not supported	Not supported
Not Between	Supported	Supported	Not supported	Not supported	Not supported	Not supported

Table 13–11 lists supported validation combinations for the list validator.

Table 13–11 List Validation

Compare type	String
In	Supported
Not In	Supported

Table 13–12 and Table 13–13 lists supported validation combinations for the compare validator.

Table 13–12 Compare Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short	String
Equals	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Not Equals	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Less Than	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported

Table 13–12 (Cont.) Compare Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short	String
`Greater Than`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
`Less Than Equal To`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
`Greater Than Equal To`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported

Table 13–13 Compare Validation - java.math, java.sql, and java.util Packages

Compare type	java.math. BigDecimal	java.math. BigInteger	java.sql.Date	java.sql.Time	java.sql.Timestamp	java.util.Date
`Equals`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Not Equals`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Less Than`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Greater Than`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Less Than Equal To`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Greater Than Equal To`	Supported	Supported	Not supported	Not supported	Not supported	Not supported

13.12 About Data Change Events

To simplify data change events, OEPE uses the property change listener pattern. In most cases you can use OEPE to generate the necessary code to source notifications from your beans' property accessors by selecting the **Notify listeners when property changes** checkbox in the Generate Accessors dialog (see Section 13.3.5.2, "About the Managed Beans Category" for details). The `PropertyChangeSupport` object is generated automatically, with the calls to `firePropertyChange` in the newly-generated setter method. Additionally, the `addPropertyChangeListener` and `removePropertyChangeListener` methods are added so property change listeners can register and unregister themselves with this object. This is what the framework uses to capture changes to be pushed to the client cache and to notify the user interface layer that data has been changed.

> **Note:** If you are manually adding a `PropertyChangeSupport` object to a class, you must also include the `addPropertyChangeListener` and `removePropertyChangeListener` methods (using these explicit method names).

Property changes alone will not solve all the data change notifications, as in the case where you have a bean wrapped by a data control and you want to expose a collection of items. While a property change is sufficient when individual items of the list change, it is not sufficient for *cardinality* changes. In this case, rather than fire a

property change for the entire collection, which would cause a degradation of performance, you can instead refresh just the collection delta. To do this you need to expose more data than is required for a simple property change, which you can do using the `ProviderChangeSupport` class.

> **Note:** The `ProviderChangeSupport` object is *not* generated automatically—you must manually add it to your class—along with the `addProviderChangeListener` and `removeProviderChangeListener` methods (using these explicit method names).

Since the provider change is required only when you have a dynamic collection exposed by a data control wrapped bean, there are only a few types of provider change events to fire:

- `fireProviderCreate`—when a new element is added to the collection
- `fireProviderDelete`—when an element is removed from the collection
- `fireProviderRefresh`—when multiple changes are done to the collection at one time and you decide it is better to simply ask for the client to refresh the entire collection (this should only be used in bulk operations)

The `ProviderChangeSupport` class is used for sending notifications relating to collection elements, so that components update properly when a change occurs in a JavaBean data control. It follows a similar pattern to the automatically-generated `PropertyChangeSupport` class, but the event objects used with `ProviderChangeSupport` send more information, including the type of operation as well as the key and position of the element that changed. `ProviderChangeSupport` captures structural changes to a collection, such as adding or removing an element (or provider) from a collection. `PropertyChangeSupport` captures changes to the individual items in the collection.

The example below shows how to use `ProviderChangeSupport` for sending notifications relating to structural changes to collection elements (such as when adding or removing a child). For more information on the `ProviderChangeListener` interface and the `ProviderChangeEvent` class, see the *Java API Reference for Oracle Mobile Application Framework*.

```
public class NotePad {
   private   static    List
                       s_notes              = null;

/* manually adding property change listener as well as provider change listener.
*/
   protected transient PropertyChangeSupport
                       propertyChangeSupport = new PropertyChangeSupport(this);
   protected transient ProviderChangeSupport
                       providerChangeSupport = new ProviderChangeSupport(this);

    public NotePad() {
        …
    }

    public  mobile.Note[] getNotes() {
        mobile.Note n[] = null;

        synchronized (this)
        {
            if(s_notes.size() > 0) {
```

```
            n = (mobile.Note[])
                s_notes.toArray(new mobile.Note[s_notes.size()]);
        }
        else {
            n = new mobile.Note[0];
        }
    }

    return n;
}

public void addNote() {
    System.out.println("Adding a note ....");
    Note  n = new Note();
    int   s = 0;

    synchronized (this)
    {
        s_notes.add(n);
        s = s_notes.size();
    }

    System.out.println("firing the events");
    providerChangeSupport.fireProviderCreate("notes", n.getUid(), n);
}

public void removeNote() {
    System.out.println("Removng a note ....");
    if(s_notes.size() > 0) {
        int     end = -1;
        Note    n   = null;

        synchronized (this)
        {
            end   = s_notes.size() - 1;
            n     = (Note)s_notes.remove(end);
        }

        System.out.println("firing the events");
    providerChangeSupport.fireProviderDelete("notes", n.getUid());
    }
}

public void RefreshNotes() {
    System.out.println("Refreshing the notes ....");

    providerChangeSupport.fireProviderRefresh("notes");
}

public void addProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.addProviderChangeListener(l);
}

public void removeProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.removeProviderChangeListener(l);
}

protected String   status;

/* --- OEPE generated accessors --- */
```

```
    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public void setStatus(String status) {
        String oldStatus = this.status;
        this.status = status;
        propertyChangeSupport.firePropertyChange("status", oldStatus, status);
    }

    public String getStatus() {
        return status;
    }
}
```

Data changes are passed back to the client (to be cached) with any response message or return value from the JVM layer. This allows OEPE

to compress and reduce the number of events and updates to refresh to the user interface, allowing the framework to be as efficient as possible.

However, there are times where you may need to have a background thread handle a long-running process (such as web-service interactions, database interactions, or expensive computations) and notify the user interface independent of a user action. To update data on an AMX page to reflect the current values of data fields whose values have changed, you can avoid the performance hit associated with reloading the whole AMX page by calling `AdfmfJavaUtilities.flushDataChangeEvent` to force the currently queued data changes to the client.

> **Note:** The `flushDataChangeEvent` method can only be executed from a background thread.

The next example shows how the `flushDataChangeEvent` method can be used to force pending data changes to the client. For more information about `oracle.adfmf.framework.api.AdfmfJavaUtilities.flushDataChangeEvent`, see *Java API Reference for Oracle Mobile Application Framework*.

```
/* Note - Simple POJO used by the NotePad managed bean or data control wrapped
bean */

package mobile;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;


/**
 * Simple note object
 * uid   - unique id - generated and not mutable
 * title - title for the note - mutable
 * note  - note comment - mutable
```

```
 */
public class Note {
    /* standard OEPE generated property change support */
    protected transient PropertyChangeSupport
                      propertyChangeSupport = new PropertyChangeSupport(this);


    private static boolean s_backgroundFlushTestRunning = false;


    public Note() {
        this("" + (System.currentTimeMillis() % 10000));
    }

    public Note(String id) {
        this("UID-"+id, "Title-"+id, "");
    }

    public Note(String uid, String title, String note) {
        this.uid     = uid;
        this.title   = title;
        this.note    = note;
    }


    /* update the current note with the values passed in */
    public void updateNote(Note n) {
        if (this.getUid().compareTo(n.getUid()) == 0) {
            this.setTitle(n.getTitle());
            this.setNote(n.getNote());
        } else {
            throw new IllegalArgumentException("note");
        }
    }


    /* background thread to simulate some background process that make changes */
    public void startNodeBackgroundThread(ActionEvent actionEvent) {
        Thread backgroundThread   = new Thread() {
            public void run() {
                System.out.println("startBackgroundThread enter - " +
                                                        s_
backgroundFlushTestRunning);

                s_backgroundFlushTestRunning = true;
                for(int i = 0; i <= iterations; ++i) {
                    try
                    {
                        System.out.println("executing " + i + " of " + iterations
+ "
                                " iterations.");

                        /* update a property value */
                        if(i == 0) {
                            setNote("thread starting");
                        }
                        else if( i == iterations) {
                            setNote("thread complete");
                         s_backgroundFlushTestRunning = false;
                        }
```

```
                        else {
                            setNote("executing " + i + " of " + iterations + "
iterations.");
                        }
                        setVersion(getVersion() + 1);
                        setTitle("Thread Test v" + getVersion());
                        AdfmfJavaUtilities.flushDataChangeEvent();  /* key line */
                    }
                    catch(Throwable t)
                    {
                        System.err.println("Error in the background thread: " +
t);
                    }

                    try {
                        Thread.sleep(delay);  /* sleep for 6 seconds */
                    } catch (InterruptedException ex) {
                        ex.printStackTrace();
                    }
                }
            }
        };

        backgroundThread.start();
    }

    protected String uid;
    protected String title;
    protected String note;
    protected int    version;

    protected int    iterations =  10;
    protected int    delay      = 500;


    /* --- OEPE generated accessors --- */

    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public String getUid() {
        return uid;
    }

    public void setTitle(String title) {
        String oldTitle = this.title;
        this.title = title;
        propertyChangeSupport.firePropertyChange("title", oldTitle, title);
    }

    public String getTitle() {
        return title;
    }

    public void setNote(String note) {
```

```
        String oldNote = this.note;
        this.note = note;
        propertyChangeSupport.firePropertyChange("note", oldNote, note);
    }

    public String getNote() {
        return note;
    }

    public void setVersion(int version) {
        int oldVersion = this.version;
        this.version = version;
        propertyChangeSupport.firePropertyChange("version", oldVersion, version);
    }

    public int getVersion() {
        return version;
    }

    public void setIterations(int iterations) {
        int oldIterations = this.iterations;
        this.iterations = iterations;
        propertyChangeSupport.
                 firePropertyChange("iterations", oldIterations, iterations);
    }

    public int getIterations() {
        return iterations;
    }

    public void setDelay(int delay) {
        int oldDelay = this.delay;
        this.delay = delay;
        propertyChangeSupport.
                firePropertyChange("delay", oldDelay, delay);
    }

    public int getDelay() {
        return delay;
    }
}
```

```
/* NotePad - Can be used as a managed bean or wrapped as a data control */

package mobile;

import java.util.ArrayList;
import java.util.List;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;
import oracle.adfmf.java.beans.ProviderChangeListener;
import oracle.adfmf.java.beans.ProviderChangeSupport;
```

```
public class NotePad {
    private static List     s_notes                       = null;
    private static boolean  s_backgroundFlushTestRunning = false;

    protected transient     PropertyChangeSupport
        propertyChangeSupport = new PropertyChangeSupport(this);

    protected transient     ProviderChangeSupport
        providerChangeSupport = new ProviderChangeSupport(this);

    public NotePad() {
        if (s_notes == null) {
            s_notes = new ArrayList();

            for(int i = 1000; i < 1003; ++i) {
                s_notes.add(new Note(""+i));
            }
        }
    }

    public  mobile.Note[] getNotes() {
        mobile.Note n[] = null;

        synchronized (this)
        {
            if(s_notes.size() > 0) {
                n = (mobile.Note[])s_notes.
                     toArray(new mobile.Note[s_notes.size()]);
            }
            else {
                n = new mobile.Note[0];
            }
        }

        return n;
    }

    public void addNote() {
        System.out.println("Adding a note ....");
        Note  n = new Note();
        int   s = 0;

        synchronized (this)
        {
            s_notes.add(n);
            s = s_notes.size();
        }

        System.out.println("firing the events");

        /* update the note count property on the screen */
        propertyChangeSupport.
             firePropertyChange("noteCount", s-1, s);

        /* update the notes collection model with the new note */
        providerChangeSupport.
             fireProviderCreate("notes", n.getUid(), n);

        /* to update the client side model layer */
        AdfmfJavaUtilities.flushDataChangeEvent();
```

```
}

public void removeNote() {
    System.out.println("Removing a note ....");
    if(s_notes.size() > 0) {
        int     end = -1;
        Note    n   = null;

        synchronized (this)
        {
            end   = s_notes.size() - 1;
            n     = (Note)s_notes.remove(end);
        }

        System.out.println("firing the events");

        /* update the client side model layer */
        providerChangeSupport.
            fireProviderDelete("notes", n.getUid());

       /* update the note count property on the screen */
       propertyChangeSupport.
            firePropertyChange("noteCount", -1, end);
    }
}

public void RefreshNotes() {
    System.out.println("Refreshing the notes ....");

    /* update the entire notes collection on the client */
    providerChangeSupport.fireProviderRefresh("notes");
}

public int getNoteCount() {
    int size = 0;

    synchronized (this)
    {
        size = s_notes.size();
    }
   return size;
}

public void
addProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.addProviderChangeListener(l);
}

public void
removeProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.removeProviderChangeListener(l);
}

public void
startListBackgroundThread(ActionEvent actionEvent) {
    for(int i = 0; i < 10; ++i) {
        _startListBackgroundThread(actionEvent);
        try {
            Thread.currentThread().sleep(i * 1234);
        } catch (InterruptedException e) {
```

```
        }
    }
}

public void
_startListBackgroundThread(ActionEvent actionEvent) {
    Thread backgroundThread   = new Thread() {
        public void run() {
            s_backgroundFlushTestRunning = true;

            for(int i = 0; i <= iterations; ++i) {
                System.out.println("executing " + i +
                      " of " + iterations + " iterations.");

                try
                {
                    /* update a property value */
                    if(i == 0) {
                        setStatus("thread starting");
                        addNote();  // add a note
                    }
                    else if( i == iterations) {
                        setStatus("thread complete");
                        removeNote();  // remove a note
                     s_backgroundFlushTestRunning = false;
                    }
                    else {
                        setStatus("executing " + i + " of " +
                                iterations + " iterations.");

                        synchronized (this)
                        {
                            if(s_notes.size() > 0) {
                                Note n =(Note)s_notes.get(0);

                                n.setTitle("Updated-" +
                                      n.getUid() + " v" + i);
                            }
                        }
                    }
                    AdfmfJavaUtilities.
                             flushDataChangeEvent();
                }
                catch(Throwable t)
                {
                    System.err.
                    println("Error in bg thread - " + t);
                }

                try {
                     Thread.sleep(delay);
                } catch (InterruptedException ex) {
                    setStatus("inturrpted " + ex);
                    ex.printStackTrace();
                }
            }
        }
    };

    backgroundThread.start();
```

```
    }


    protected int iterations = 100;
    protected int delay      = 750;

    protected String   status;

    /* --- OEPE generated accessors --- */

    public void
    addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void
    removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public void setStatus(String status) {
        String oldStatus = this.status;
        this.status = status;
        propertyChangeSupport.
                firePropertyChange("status", oldStatus, status);
    }

    public String getStatus() {
        return status;
    }

    public void setIterations(int iterations) {
        int oldIterations = this.iterations;
        this.iterations = iterations;
        propertyChangeSupport.
                firePropertyChange("iterations",
                                   oldIterations, iterations);
    }

    public int getIterations() {
        return iterations;
    }

    public void setDelay(int delay) {
        int oldDelay = this.delay;
        this.delay = delay;
        propertyChangeSupport.
                firePropertyChange("delay", oldDelay, delay);
    }

    public int getDelay() {
        return delay;
    }
}
```

The StockTracker sample application provides an example of how data change events use Java to enable data changes to be reflected in the user interface. This sample application is available from **File** > **New** > **MAF Examples**.

For more information about sample applications, see Appendix E, "MAF Sample Applications."

14

Configuring End Points Used in MAF Applications

This chapter describes how to use the Configuration Service to configure end points used in a MAF application.

This chapter includes the following sections:

- Section 14.1, "Introduction to Configuring End Points"
- Section 14.2, "Defining the Configuration Service End Point"
- Section 14.3, "Creating the User Interface for the Configuration Service"
- Section 14.4, "About the URL Construction"
- Section 14.5, "Setting Up the Configuration Service on the Server"

14.1 Introduction to Configuring End Points

The Configuration Service is a tool that allows you to configure end points used by web services, login utilities, and any other parts of a MAF application.

Note: MAF applications on the Universal Windows Platform do not support the use of the Configuration Service.

14.2 Defining the Configuration Service End Point

The end point (or seed) URL is defined in the `connections.xml` file and a new connection entry must be added to that file. This new connection should be of type `HttpURLConnection`, with its `url` value pointing to the configuration server end point (or seed) URL and its `name` set to an arbitrary value which will eventually be referenced in a JavaBean code.

This example shows how to define the Configuration Service end point in the `connections.xml` file.

```
<RefAddresses>
     <XmlRefAddr addrType="ConfigServiceConnection">
       <Contents>
         <urlconnection name="ConfigServiceConnection" url="http://127.0.0.1"/>
       </Contents>
     </XmlRefAddr>
   </RefAddresses>
 </Reference>
```

```
<!-- Login Server connection for secured configuration service -->
  <Reference name="ConfigServerLogin" className="oracle.adf.model.connection.adfmf.LoginConnection"
             adfCredentialStoreKey="ConfigServerLogin" partial="false"
             manageInOracleEnterpriseManager="true"
             deployable="true" xmlns="">
    <Factory className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
    <RefAddresses>
      <XmlRefAddr addrType="adfmfLogin">
        <Contents>
          <login url="http://127.0.0.1"/>
          <logout url="http://127.0.0.1"/>
          <authenticationMode value="remote"/>
          <idleTimeout value="300"/>
          <sessionTimeout value="28800"/>
          <maxFailuresBeforeCredentialCleared value="3"/>
          <rememberCredentials>
            <enableRememberUserName value="true"/>
            <rememberUserNameDefault value="true"/>
            <enableRememberPassword value="false"/>
            <enableStayLoggedIn value="false"/>
          </rememberCredentials>
          <accessControl/>
          <userObjectFilter/>
        </Contents>
      </XmlRefAddr>
    </RefAddresses>
```

If security is enabled for the configuration server, the connections.xml file has to include a login connection that points to the same end point URL as the URL connection. The login connection and `HttpURLConnection` should share the same `adfCredentialStoreKey`, as the previous example shows.

Most of the time, the end point URL needs to be retrieved from the end user. To address this use case, create a user interface to retrieve the value of the end point URL from the end user and set it in an application preference. The retrieved value can then be used in a Java bean method to override the connection URL value, as Example 14–1 shows.

Example 14–1 Overriding the Connection Definition

```
AdfmfJavaUtilities.clearSecurityConfigOverrides(<ConfigService_ConnectionName>);
AdfmfJavaUtilities.overrideConnectionProperty(<ConfigService_ConnectionName>,
"urlconnection",
                                                              "url",
<ConfigService_EndpointURL>);

// Required if Config Service is secured and the authentication endpoints are
input by the user
AdfmfJavaUtilities.clearSecurityConfigOverrides(<ConfigService_
AuthenticationConnectionName>);
AdfmfJavaUtilities.overrideConnectionProperty(<ConfigService_
AuthenticationConnectionName>,
                                                          "login", "url",
<Login_EndpointURL>);
AdfmfJavaUtilities.overrideConnectionProperty(<ConfigService_
AuthenticationConnectionName>,
                                                        "logout", "url",
<Logout_EndpointURL>);

// Final step is to apply the changes.
```

```
AdfmfJavaUtilities.updateApplicationInformation(false);
```

14.3 Creating the User Interface for the Configuration Service

If there is a requirement for the Configuration Service user interface, you should create it in a new or existing application feature.

MAF provides a set of APIs within the `oracle.adfmf.config.client.ConfigurationService` class that allow to check for new changes on the server and download the updates. You can use these APIs in a Java bean to activate the respective methods through the Configuration Service application feature.

In the following list of APIs and their sample usage, the `_configservice` variable represents an instance of the `oracle.adfmf.config.client.ConfigurationService` class:

- `setDeliveryMechanism` method sets the delivery mechanism for the Configuration Service. Since the communication with the previous release's configuration server is enabled through HTTP, `http` is passed in as an argument to this method:

  ```
  _configservice.setDeliveryMechanism("http");
  ```

  Note: The method argument refers to the web transport that is to be used for the Configuration Service and should not be confused with HTTP or HTTPS: if the end point is an HTTPS URL, setting the transport to http is still valid.

- `setDeliveryMechanismConfiguration` method sets additional attributes on the Configuration Service allowing to associate the configuration server connection and the end point URL:

  ```
  _configservice.setDeliveryMechanismConfiguration("connectionName",
                                                   "ConfigServerConnection");
  _configservice.setDeliveryMechanismConfiguration("root", <Endpoint_URL);
  ```

- `isThereAnyNewConfigurationChanges` method checks whether or not there are any changes on the server that are available for download, and if there are, this method returns `true`:

  ```
  _configservice.isThereAnyNewConfigurationChanges(<APPLICATION_ID>, <VERSION>);
  ```

- `stageAndActivateVersion` method triggers download of updates by the Configuration Service. The application version is passed in as an argument to this method, either as a hardcoded value or obtained through the `Application.getApplicationVersion` API:

  ```
  _configservice.stageAndActivateVersion("1.0");

  _configservice.stageAndActivateVersion(Application.getApplicationVersion);
  ```

- `addProgressListener` method registers an update progresslistener on the Configuration Service to receive update messages and progress status. The underlying class should implement the `ProgressListener` interface and the `updateProgress` method which is to be called from the Configuration Service. The `updateProgress` method receives the progress update message and the update percentage complete:

```
_configservice.addProgressListener(this);
```

- `removeProgressListener` method unregisters the update progress listener:

```
_configservice.removeProgressListener(this);
```

The ConfigServiceDemo sample application demonstrates how to use these APIs to communicate with the configuration server. For more information, see Appendix E, "MAF Sample Applications."

For more information about the `oracle.adfmf.config.client.ConfigurationService` class, see *Java API Reference for Oracle Mobile Application Framework*.

14.4 About the URL Construction

The Configuration Service takes the endpoint URL that the user provides or that is specified in the connections.xml file and uses it to construct the URL to download the connections.xml file.

For example, if a user provides the following endpoint URL for an application that has an application ID value of `com.mycompany.appname`:

```
http://my.server.com:port/SomeLocation
```

Then, the Configuration Service constructs the following URL to downloada the connections.xml file:

```
http://my.server.com:port/SomeLocation/com.mycompany.appname/connections.xml
```

14.5 Setting Up the Configuration Service on the Server

The Configuration Service can be implemented as a service that accepts HTTP `GET` request and returns the `connections.xml` file.

The URL used by the Configuration Service client is of the following format:

`url configured in` `/connections.xml`

The Configuration Service end point may be secured using basic authentication (`BASIC_AUTH`) over HTTP and HTTPS.

15

Using Web Services in a MAF Application

This chapter describes how to access REST web services from a MAF application.

This chapter includes the following sections:

- Section 15.1, "Introduction to Using Web Services in a MAF Application"
- Section 15.2, "Creating a Rest Service Adapter to Access Web Services"
- Section 15.3, "Accessing Secure Web Services"
- Section 15.4, "Configuring the Browser Proxy Information"

15.1 Introduction to Using Web Services in a MAF Application

MAF supports the consumption of REST web services with JSON objects (REST-JSON) in MAF applications. This type of web service is recommended by MAF over alternative web services as the smaller payloads that REST-JSON web services generate typically mean lower response times for communication between an application and the services that it accesses.

Using a REST-JSON web service in a MAF application requires you to configure a connection to the URL end point for the web service in your application. MAF stores this end point in the `connections.xml` file of your application. You also write an adapter (`RESTServiceAdapter`) that takes the value you configured in `connections.xml` and uses it to construct the request URI that you submit to the web service. You must also write Java classes to model the data that the web service returns. Use these classes to generate data controls that bind your application's AMX pages to the data that the web service accesses. If your application accesses secured web services, you must associate a security policy with the connection to the URL end point for the web service. If your application must access services hosted outside your corporate firewall, you may also need to configure entries in your application's `maf.properties` file (located at *`assembly_project`*`/META-INF`).

The WorkBetter sample application provides examples of programmatically consuming REST services using the `RESTServiceAdapter`. For more information about how to access the source code of the WorkBetter sample application, see Appendix E, "MAF Sample Applications.".

15.2 Creating a Rest Service Adapter to Access Web Services

Use a rest service adapter `RestServiceAdapter` to access data sent using REST calls and to trigger execution of web service operations. The `RestServiceAdapterFactory.createRestServiceAdapter()` API from the

`oracle.maf.api.dc.ws.rest` package creates adapters that implement `RestServiceAdapter`.

Ensure that the connection to the URL end point for the service exists in the `connections.xml` file, and then add your code to the bean class, as the following examples demonstrate.

Use the `RestServiceAdapterFactory.createMcsRestServiceAdapter()` API if you want to create an adapter that sends diagnostic information to Mobile Cloud Service, as described in Section 30.8, "Sending Diagnostic Information to Oracle Mobile Cloud Service.".

For more information about `RestServiceAdapterFactory` and `RestServiceAdapter`, see *Java API Reference for Oracle Mobile Application Framework*.

```
....
import oracle.maf.api.dc.ws.rest.RestServiceAdapterFactory;
import oracle.maf.api.dc.ws.rest.RestServiceAdapter;
....
RestServiceAdapterFactory factory = RestServiceAdapterFactory.newFactory();
RestServiceAdapter restServiceAdapter = factory.createRestServiceAdapter();

// Clear any previously set request properties, if any
restServiceAdapter.clearRequestProperties();

// Set the connection name
restServiceAdapter.setConnectionName("RestServerEndpoint");

// Specify the type of request
restServiceAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_GET);

// Specify the number of retries
restServiceAdapter.setRetryLimit(0);

// Set the URI which is defined after the endpoint in the connections.xml.
// The request is the endpoint + the URI being set
restServiceAdapter.setRequestURI("/WebService/Departments/100");

String response = "";

// Execute SEND and RECEIVE operation
try {
   // For GET request, there is no payload
   response = restServiceAdapter.send("");
}
catch (Exception e) {
   e.printStackTrace();
}
```

The following example demonstrates the use of the `RestServiceAdapter` for the `POST` request.

```
String id = "111";
String name = "TestName111";
String location = "TestLocation111";
....

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_POST);
restServiceAdapter.setRetryLimit(0);
```

```
restServiceAdapter.setRequestURI("/WebService/Departments");

String response = "";

// Execute SEND and RECEIVE operation
try {
   String postData = makeDepartmentPost("DEPT", id, name, location);
   response = restServiceAdapter.send(postData);
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + response);

private String makeDepartmentPost(String rootName, String id,
                                  String name, String location) {
   String ret = "<" + rootName + ">";
   ret += "<DEPTID>" + id + "</DEPTID>";
   ret += "<NAME>" + name + "</NAME>";
   ret += "<LOCATION>" + location + "</LOCATION>";
   ret += "</" + rootName + ">";
   return ret;
}
```

The following example demonstrates the use of the `RestServiceAdapter` for the `PUT` request.

```
String id = "111";
String name = "TestName111";
String location = "TestLocation111";

....

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_PUT);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/WebService/Departments");

String response = "";

// Execute SEND and RECEIVE operation
try {
   String putData = makeDepartmentPut("DEPT", id, name, location);
   response = restServiceAdapter.send(putData);
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + response);

private String makeDepartmentPut(String rootName, String id,
                                  String name, String location) {
   String ret = "<" + rootName + ">";
   ret += "<DEPTID>" + id + "</DEPTID>";
   ret += "<NAME>" + name + "</NAME>";
   ret += "<LOCATION>" + location + "</LOCATION>";
   ret += "</" + rootName + ">";
   return ret;
}
```

The following example demonstrates the use of the `RestServiceAdapter` for the `DELETE` request.

```
....

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_DELETE);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/WebService/Departments/44");

String response = "";

// Execute SEND and RECEIVE operation
try {
   // For DELETE request, there is no payload
   response = restServiceAdapter.send("");
}
catch (Exception e) {
   e.printStackTrace();
}

System.out.println("The response is:  " + response);
```

When you use the `RestServiceAdapter`, you should set the `Accept` and `Content-Type` headers to ensure that your request and response payloads are not deemed invalid due to mismatched MIME type.

> **Note:** The REST web service adapter only supports UTF-8 character set on MAF applications. UTF-8 is embedded in the adapter program.

15.2.1 Accessing Input and Output Streams

You can use the following `RestServiceAdapter` methods to obtain and customize the `javax.microedition.io.HttpConnection`, as well as access and interact with the connection's input and output streams which allows you to read data from the `HttpConnection` and write to it for further upload to the server:

- Get an `HttpConnection`:

  ```
  HttpConnection getHttpConnection(String requestMethod,
                                   String request,
                                   Object httpHeadersValue)
  ```

- Get the `HttpConnection`'s `OutputStream`:

  ```
  OutputStream getOutputStream(HttpConnection connection)
  ```

- Get the `HttpConnection`'s `InputStream`:

  ```
  InputStream getInputStream(HttpConnection connection)
  ```

- Close the `HttpConnection`:

  ```
  void close (HttpConnection connection)
  ```

- Look up a connection name in the `connections.xml` file and return the connection's end point:

  ```
  String getConnectionEndPoint(String connecionName)
  ```

These methods, while accomplishing the same functionality as the `RestServiceAdapter`'s send and `sendReceive` methods, provide opportunity for customization of the connection and the process of sending requests and receiving responses.

The following example initializes and returns an `HttpConnection` using the provided request method, request, and HTTP headers value. In addition, it injects basic authentication into the request headers from the credential store, obtains the input stream and closes the connection.

```
....
import oracle.maf.api.dc.ws.rest.RestServiceAdapterFactory;
import oracle.maf.api.dc.ws.rest.RestServiceAdapter;
....

RestServiceAdapterFactory factory = RestServiceAdapterFactory.newFactory();
RestServiceAdapter restServiceAdapter = factory.createRestServiceAdapter();


restServiceAdapter.clearRequestProperties();

// Specify the type of request
String requestMethod = RestServiceAdapter.REQUEST_TYPE_GET;

// Get the connection end point from connections.xml
String requestEndPoint = restServiceAdapter.getConnectionEndPoint("GeoIP");

// Get the URI which is defined after the end point
String requestURI = "/xml/" + someIpAddress;

// The request is the end point + the URI being set
String request = requestEndPoint + requestURI;

// Specify some custom request headers
HashMap httpHeadersValue = new HashMap();
httpHeadersValue.put("Accept-Language", "en-US");
httpHeadersValue.put("My-Custom-Header-Item", "CustomItem1");

// Get the connection
HttpConnection connection =
                  restServiceAdapter.getHttpConnection(requestMethod,
                                                       request,
                                                       httpHeadersValue);

// Get the input stream
InputStream inputStream = restServiceAdapter.getInputStream(connection);

// Define data
ByteArrayOutputStream byStream = new ByteArrayOutputStream();

int res = 0;
int bufsize = 0, bufread = 0;

byte[] data = (bufsize > 0) ? new byte[bufsize] : new byte[1024];

// Use the input stream to read data
while ((res = inputStream.read(data)) > 0) {
   byStream.write(data, 0, res);
   bufread = bufread + res;
}
```

```
data = byStream.toByteArray();

// Use data
...

restServiceAdapter.close(connection);
...
```

15.2.2 Support for Non-Text Responses

You can use the `RestServiceAdapter` to handle binary (non-text) responses received from web service calls. These responses can include any type of binary data, such as PDF or video files. The `RestServiceAdapter` method to use is `sendReceive`.

The following example shows how to send a request for a file to a REST server, and then save the file to a disk.

```
estServiceAdapterFactory factory = RestServiceAdapterFactory.newFactory();
RestServiceAdapter restServiceAdapter = factory.createRestServiceAdapter();

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("JagRestServerEndpoint");
restServiceAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_GET);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/ftaServer/v1/kpis/123/related/1");

// Set credentials needed to access the REST server
String theUsername = "hr_spec_all";
String thePassword = "Welcome1";
String userPassword = theUsername + ":" + thePassword;
String encoding = new sun.misc.BASE64Encoder().encode(userPassword.getBytes());

restServiceAdapter.addRequestProperty("Authorization", "Basic " + encoding);

// Execute the SEND and RECEIVE operation.
// Since it is a GET request, there is no payload.
try {
   this.responseRaw = restServiceAdapter.sendReceive("");
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + this.responseRaw);

// Write the response to a file
writeByteArrayToFile(this.responseRaw);
```

The following example demonstrates a method that is called by the code from the preceding example. This method saves a `byte[]` response to a file on disk:

```
public void writeByteArrayToFile(byte[] fileContent) {
   BufferedOutputStream bos = null;
   try {
      FileOutputStream fos = new FileOutputStream(new File(fileToSavePath));
      bos = new BufferedOutputStream(fos);
      // Write the byte [] to a file
      System.out.println("Writing byte array to file");
      bos.write(fileContent);
      System.out.println("File written");
   }
   catch(FileNotFoundException fnfe) {
```

```
            System.out.println("Specified file not found" + fnfe);
        }
        catch (IOException ioe) {
            System.out.println("Error while writing file" + ioe);
        }
        finally {
            if(bos != null) {
                try {
                    // Flush the BufferedOutputStream
                    bos.flush();
                    // Close the BufferedOutputStream
                    bos.close();
                }
                catch (Exception e) {
                }
            }
        }
    }
```

15.3 Accessing Secure Web Services

MAF supports both secured and unsecured web services. When a REST web service is secured, you must associate the REST connection with the predefined security policy that supports the REST web service, as described in Section 15.3.1, "How to Enable Access to Web Services."

Table 15–1 lists the predefined security policies that you can associate with connections to REST web services.

Table 15–1 Security Policies Supported for REST-Based Web Services

Authentication Type	REST Policy	Description
HTTP Basic	`oracle/http_basic_auth_over_ssl_client_policy`	This policy includes credentials in the HTTP header for outbound client requests and authenticates users against the Oracle Platform Security Services identity store. This policy also verifies that the transport protocol is HTTPS. Requests over a non-HTTPS transport protocol are refused. This policy can be enforced on any HTTP-based client.
HTTP Basic	`oracle/wss_http_token_client_policy`	This policy includes credentials in the HTTP header for outbound client requests. This policy can be enforced on any HTTP-based or HTTPS-based client.

Table 15–1 (Cont.) Security Policies Supported for REST-Based Web Services

Authentication Type	REST Policy	Description
HTTP Basic	`oracle/wss_http_token_over_ssl_client_policy`	This policy includes credentials in the HTTP header for outbound client requests and authenticates users against the Oracle Platform Security Services identity store. This policy also verifies that the transport protocol is HTTPS. Requests over a non-HTTPS transport protocol are refused. This policy can be enforced on any HTTP-based client.
HTTP Basic Web SSO	`oracle/http_cookie_client_policy`	This policy injects cookies obtained after authentication in the HTTP request header, e.g: accessing OAM Webgate resources. This policy also sets response cookies. This policy can be enforced on any REST-based client.
OAuth	`oracle/http_oauth2_token_mobile_client_policy`	This policy injects bearer token (OAuth access token) in the HTTP request header while communicating with the endpoint. This token can be obtained from any OAuth2 server. This policy can be enforced on any REST-based client.

For more information on these policies and their usage, see the Determining Which Predefined Policies to Use and Predefined Policies chapters in *Oracle Fusion Middleware Securing Web Services and Managing Policies with Oracle Web Services Manager*.

15.3.1 How to Enable Access to Web Services

When a web service is secured and expects an authentication token, you must associate the login connection with the predefined security policy that supports the web service. For a list of predefined security policies are supported for the authentication type available for REST-based web services, see Section 15.3, "Accessing Secure Web Services."

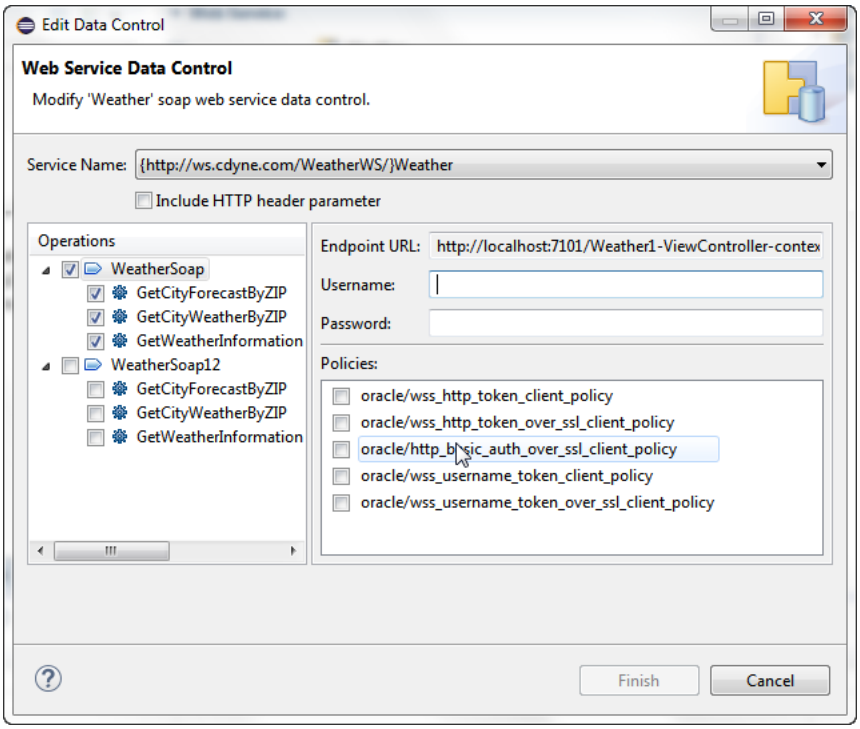
You create the login server connection using the Mobile Login Server Connection dialog in the MAF Application Editor. For details about creating the login server connection, see Section 29.5.1, "How to Create a MAF Login Connection."

To associate a security policy with a web service:

To create a persistent connection:

1. In the REST Client page of the REST Service Editor, click to open the Manage Connection wizard.
2. In the dialog, shown in Figure 15–1, click **Show Policies** and select the appropriate one.

Figure 15–1 Editing Web Service Data Control Policies



15.3.2 What Happens When You Enable Access to Web Services

OEPE stores the web service policy definitions in the `wsm-assembly.xml` file (located in `META-INF` directory of the application workspace).

You can view the security policy already associated with the REST web service using the Edit Data Control dialog shown in Figure 15–1.

15.3.3 What You May Need to Know About Accessing Web Services and Containerized MAF Applications

When the MAF application is containerized at deployment time, access to secured web services behind the corporate firewall will rely on the Mobile Security Access Server (MSAS), a component in the Oracle Mobile Security Suite (OMSS), to provide a central access point for securing traffic from mobile devices to corporate resources. In this case, the MSAS instance is configured to enforce an authentication endpoint for use in the initial authentication of the user.

Additionally, the backend service endpoints must be associated 1.) with a MSAS proxy application that ensures the URL of the resource is protected by an access policy that MSAS enforces and 2.) a client policy that MSAS adds to the proxied request, for example Single Sign-On (SSO).

In order to allow the MAF application to communicate with MSAS, the user installs and registers a Secure Workspace app for the type of authentication that has been configured for the MSAS instance. Then when the user attempts to access a protected resource, the MAF application and the Secure Workspace rely on a MSAS-generated Proxy Auto-Configuration file to determine which requests to proxy using the MSAS AppTunnel.

For more information about the role of MSAS AppTunnel in the authentication process of containerized MAF applications, see Section 29.4, "Overview of the Authentication

Process for Containerized MAF Applications."

For an overview of OMSS support for containerized MAF applications, see Section 2.8, "Containerizing a MAF Application for Enterprise Distribution."

In the OMSS documentation library, see the *Policy and Assertion Template Reference for Oracle Mobile Security Access Server* for reference information about MSAS predefined security and management policies.

15.3.4 What You May Need to Know About Credential Injection

For secured web services, the user credentials are dynamically injected into a web service request at the time when the request is invoked.

MAF uses Oracle Web Services Manager (OWSM) Mobile Agent to propagate user identity through web service requests.

Before web services are invoked, the user must respond to an authentication prompt triggered by the user trying to invoke a secured MAF application feature. The user credentials are stored in a credential store—a device-native and local repository used for storing credentials associated with the authentication provider's server URL and the user. At runtime, MAF assumes that all credentials have already been stored in the IDM Mobile credential store before the time of their usage.

In the `connections.xml` file, you have to specify the login server connection's `adfCredentialStoreKey` attribute value in the `adfCredentialStoreKey` attribute of the web service connection reference in order to associate the login server to the web service security (see the two examples below).

The following example shows the definition of the web service connection referenced as `adfCredentialStoreKey="MyAuth"`, where `MyAuth` is the name of the login connection reference.

```
<Reference name="URLConnection1"
           className="oracle.adf.model.connection.url.HttpURLConnection"
           adfCredentialStoreKey="MyAuth"
           xmlns="">
   <Factory className="oracle.adf.model.connection.url.URLConnectionFactory"/>
   <RefAddresses>
      <XmlRefAddr addrType="URLConnection1">
         <Contents>
            <urlconnection name="URLConnection1"
                           url="http://myhost.us.example.com:7777/
                                SecureRESTWebService1/Echo">
               <authentication style="challange">
                  <type>basic</type>
                  <realm>myrealm</realm>
               </authentication>
            </urlconnection>
         </Contents>
      </XmlRefAddr>
      <SecureRefAddr addrType="username"/>
      <SecureRefAddr addrType="password"/>
   </RefAddresses>
</Reference>
```

The following example shows the definition of the login connection, where `MyAuth` is used as the credential store key value in the login server connection.

```
<Reference name="MyAuthName"
           className="oracle.adf.model.connection.adfmf.LoginConnection"
```

```
            adfCredentialStoreKey="MyAuth"
            partial="false"
            manageInOracleEnterpriseManager="true"
            deployable="true"
            xmlns="">
      <Factory className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
      <RefAddresses>
         <XmlRefAddr addrType="adfmfLogin">
            <Contents>
               <login url="http://172.31.255.255:7777/
                           SecuredWeb1-ViewController-context-root/faces/view1.jsf"/>
               <logout url="http://172.31.255.255:7777/
                       /SecuredWeb1-ViewController-context-root/faces/view1.jsf"/>
               <accessControl url="http://myhost.us.example.com:7777/
                             UserObjects/jersey/getUserObjects" />
               <idleTimeout value="10"/>
               <sessionTimeout value="36000"/>
               <userObjectFilter>
                  <role name="testuser1_role1"/>
                  <role name="testuser2_role1"/>
                  <privilege name="testuser1_priv1"/>
                  <privilege name="testuser2_priv1"/>
                  <privilege name="testuser2_priv2"/>
               </userObjectFilter>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
</Reference>
```

If a web service request is rejected due to the authentication failure, MAF returns an appropriate exception and invokes an appropriate action (see Section 30.6, "Using and Configuring Logging in MAF Applications"). If none of the existing exceptions correctly represent the condition, a new exception is added.

The `connections.xml` file is deployed and managed under the Configuration Service. For more information, see Chapter 14, "Configuring End Points Used in MAF Applications."

`connections.xml` files in FARs are aggregated when the MAF application is deployed. The credentials represent deployment-specific data and are not expected to be stored in FARs.

15.3.5 What You May Need to Know About Cookie Injection

Each time a MAF application requests a REST web service for cookie-based authorization, MAF's security framework enables the transport layer of the REST web service to execute cookie injection for the login connection associated with the URL endpoint of the REST web service. This is handled at runtime without configuration of the MAF application by the MAF developer.

15.4 Configuring the Browser Proxy Information

If the web service you are to call resides outside your corporate firewall, you need to ensure that you have set the appropriate Java system properties to configure the use of an HTTP proxy server.

By default, MAF determines the proxy information using the system settings on the platform where you deploy the application.

Note: It is possible to define a different proxy for each MAF application.

If you do not want to obtain the proxy information from the device settings, first you need to add the `-Dcom.oracle.net.httpProxySource` system property. The default value of this property is `native`, which means that the proxy information is to be obtained from the device settings. You need to disable it by specifying a different value, such as `user`, for example: `-Dcom.oracle.net.httpProxySource=user`

JVM uses two different mechanisms for enabling the network connection:

1. The generic connection framework (GCF). If this mechanism is used, the proxy is defined through the system property `-Dcom.sun.cdc.io.http.proxy=<host>:<port>`
2. `java.net` API. If this mechanism is used, the proxy is defined through the standard `http.proxyHost` and `http.proxyPort`.

In either case, it is recommended to define all three properties in the `maf.properties` file (located at *`assembly_project`*`/META-INF`), which would look similar to the following:

```
java.commandline.argument=-Dcom.oracle.net.httpProxySource=user
java.commandline.argument=-Dcom.sun.cdc.io.http.proxy=www-proxy.us.mycompany.com:80
java.commandline.argument=-Dhttp.proxyHost=www-proxy.us.mycompany.com
java.commandline.argument=-Dhttp.proxyPort=80
```

Note: These properties affect only the JVM side of network calls.

16

Working with REST Services

This chapter describes the REST Service Editor, which allows you to create REST APIs and data types which can be used to generate code, and to access, use, and test REST services, and to use Oracle Mobile Cloud Service to develop mobile applications.

This chapter includes the following sections:

- Section 16.1, "Introduction to Working with REST Services"
- Section 16.2, "Using the REST Client"
- Section 16.3, "Modifying the Request Content"
- Section 16.4, "Importing and Modeling the REST API"
- Section 16.5, "Importing and Modeling Data Types"
- Section 16.6, "Testing Modeled Requests Against the REST Service"
- Section 16.7, "Creating REST Service Artifacts"
- Section 16.8, "Using the Connections View"

16.1 Introduction to Working with REST Services

The REST Service Editor allows you to create REST APIs and data types from which you can generate code to use in applications. It helps you to develop applications which interact with REST APIs, and test how a REST API that you are developing works. You can use the editor when you are developing an application that will be used with a REST service to see how the application handles REST requests.

You can:

- Connect to a live REST service and send requests to examine the responses returned by the server.
- Create a REST API and data types either from scratch, or by importing the REST API and data types of a live service and modeling your own REST API based on it.

The editor works with REST service descriptions, which are XMI files which store the description of a web service.

The editor has three pages, which you navigate between using the tabs on the bottom left of the editor:

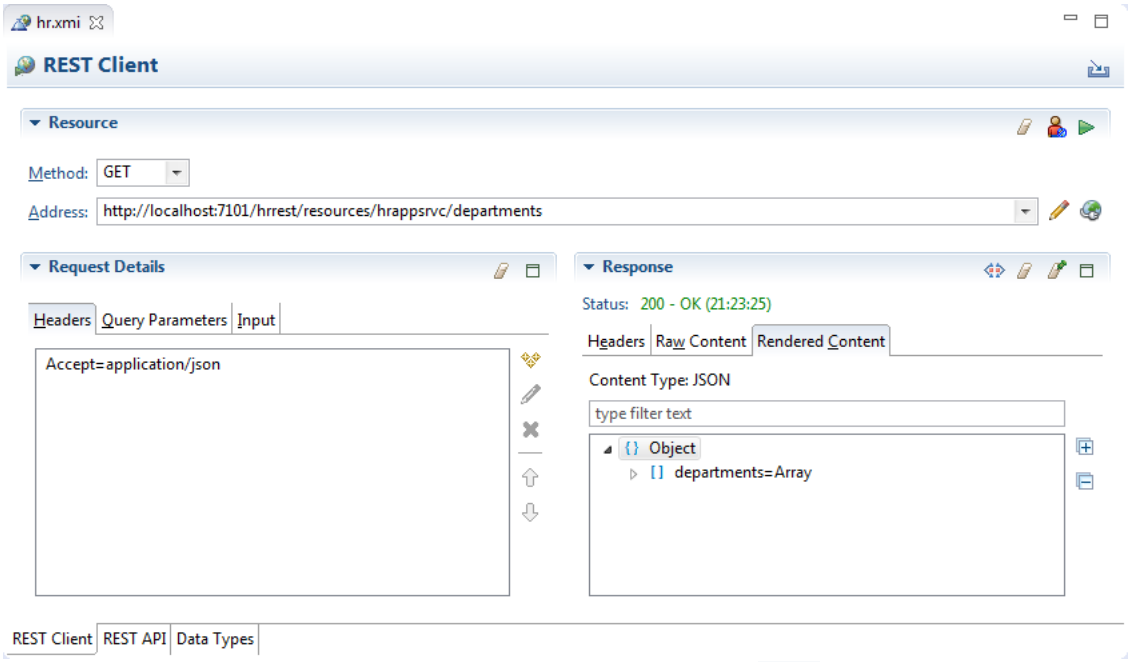
- REST Client
- REST API
- Data Types

The REST Service Editor has undo/redo support. The pages work together, but only the REST Client interacts with a live REST service. The other two pages allow you to model a REST API and its data types. When you save the editor, only the REST API and Data Types pages are saved.

16.1.1 REST Client Page

REST Client page, shown in Figure 16–1, allows you to experiment with existing applications that expose REST APIs. You can, for example, find out how your application will interact with a REST API, and send requests to the service and see the responses from the server. This page is not part of the persistent state of the REST Service Description, so changes neither dirty the editor nor are they persisted when you save the editor.

Figure 16–1 REST Client Page



You can import REST client information using which opens the Import REST Client Information dialog, where you choose to import.

You can modify the requests by:

- Choosing the method. Those available are GET, POST, PUT, and DELETE.
- Only addressing part of the service by using fragments in the URI.
- Sending queries as part of the request to return just the responses which match.
- Using headers to define what is returned in the response, for example, to specify the content type.
- Using query parameters to configure the response, for example `verbose=false`.
- Inputting information as part of the request when using POST or PUT, or output information when using GET.

In addition, from the REST Client page you can import REST Client information from the REST service.

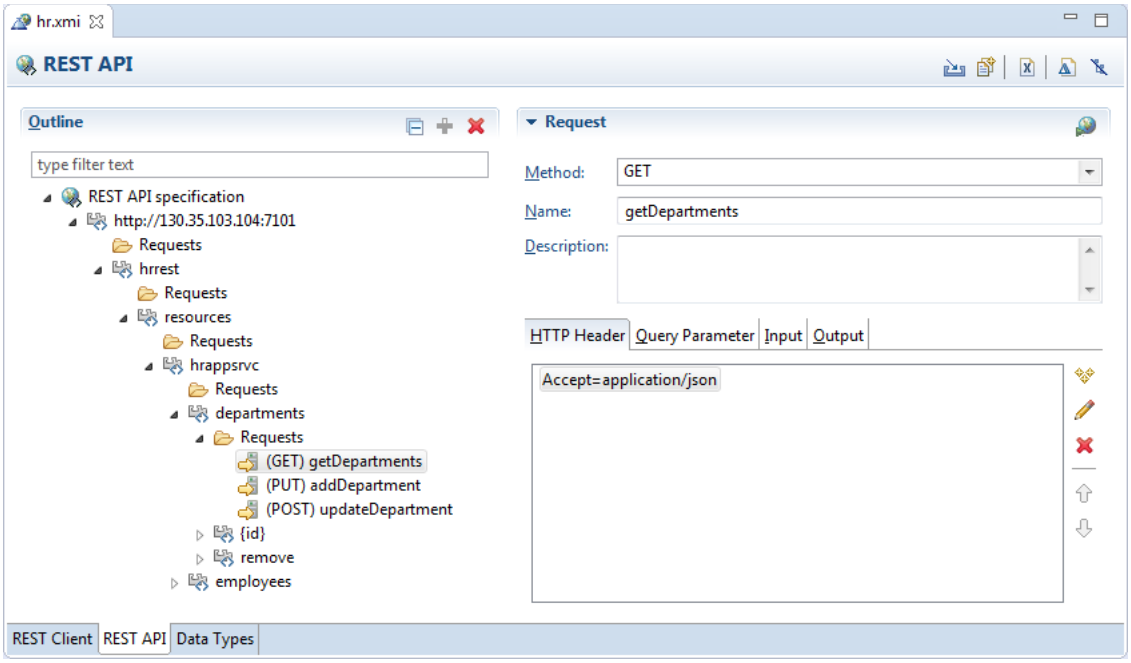
16.1.2 REST API Page

The REST API page, shown in Figure 16–2, allows you to model a REST API, or to import a RAML definition from an MCS backend.

If you are modelling a REST API and you have connected to a REST Service using the REST Client you can import REST client information from the server. The REST API is displayed on this page, and you can model changes to it.

If you are working with MCS APIs you can import a RAML definition from a file on the local file system, or from an API in a mobile backend, and model it as a REST API.

Figure 16–2 REST API Page



Paths are made up of path segments and they may contain variables of the form `{variable-name}`. In Figure 16–2, the path to the variable {id} is made up of the path segments shown in the outline to form `http://130.35.103.104:7101/hrrest/resources/happsrvc/departments/{id}`.

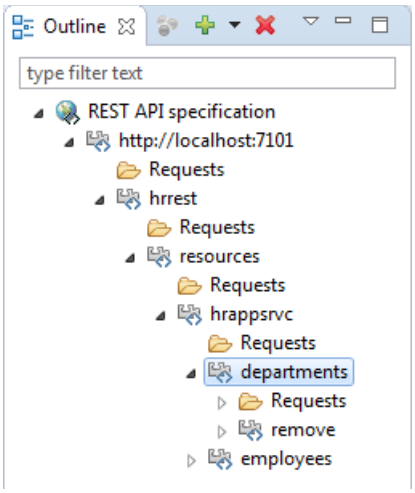
For the Java domain, the default value of the variable is `java.lang.Object`.

You can hover the mouse over a path segment or variable for more information.

You can rt a REST API from a RAML file or a Mobile Cloud Service backend by clicking . For more information, see Section 17.2.1.4.2, "Importing a REST API Using the REST Service Editor."

The Outline area displays the REST API specification for the service. The content of the area on the right depends on what is selected in the specification. The outline is reflected in the Outline window of the OEPE IDE, shown in Figure 16–3. You can perform operations in the REST API page of the REST Service Editor, or in the Outline window when the REST API page is selected in the editor.

Figure 16–3 Outline Window



In the REST API page, you can modify the REST API by:

- Specifying new paths to services.
- Creating new requests, and modify them using HTTP headers, query parameters, and specifying input and output parameters.

Once you are happy with the REST API you can copy your changes to the REST Client page to try the request against a live REST service.

If you are developing MAF applications to work with the REST service, you can generate JAVA artifacts based on the REST API to use in your application.

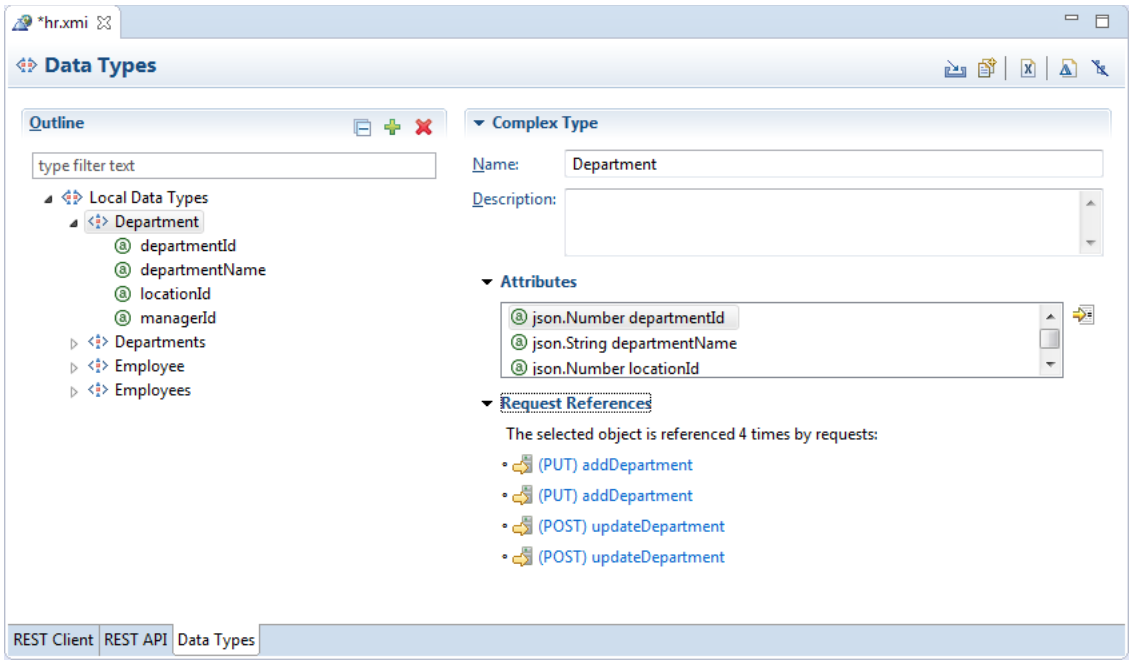
16.1.3 Data Types Page

The Data Types page, shown in Figure 16–4, allows you to model the data types used by a REST API. The imported data types are displayed on this page, and you can model changes to them.

If you have connected to a REST Service using the REST Client you can import REST client information from the server.

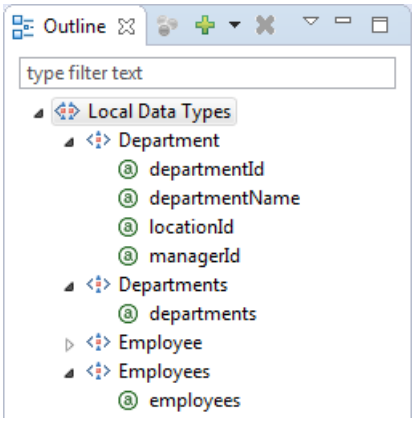
If you are working with MCS APIs, you can import the data types associated with a mobile backend or a RAML file on the local file system. You can also import a JSON Example or JSON Schema defined in a file from local file system.

Figure 16–4 Data Types Page



The Outline area displays the data types used by the modeled REST API. The content of the area on the right depends on what is selected in the specification. The outline is reflected in the Outline window of the OEPE IDE, shown in Figure 16–5. You can perform operations in the Data Types page of the REST Service Editor, or in the Outline window when the Data Types page is selected in the editor.

Figure 16–5 Outline Window



In the Data Types page, you can modify the local data types by:

- Creating new attributes.
- Specifying new complex or simple data types.

You can import data types from a file.

If you are developing MAF applications to work with the REST service, you can generate JAVA artifacts based on the data types to use in your application.

16.1.4 Using Authentication

The REST Service Editor support connecting to REST services that use authentication. The types of authentication that you can use are:

- HTTP (basic, digest, universal)
- OAuth2 ('code' grant type)
- OAuth2 ('resource owner password credentials' type)

You set authentication details when you create connections to the REST service. For more information, see Section 16.2.1.8, "How to Use Authentication."

16.1.5 How to Open the REST Service Editor

The REST Service Editor opens when:

- You open an existing REST service description in an OEPE application.
- You create a new REST service description in OEPE. For more information, see Section 16.1.6, "How to Create a REST Service Description.".

16.1.6 How to Create a REST Service Description

You can create a REST service description to contain a modeled REST API for MAF applications created or migrated to MAF 2.1 or later. The description is stored in the view project in an appropriate folder.

You can also create a REST service description by importing a RAML definition from an MCS API. For more information, see Section 17.2.1.4.1, "Importing a REST API from Oracle Cloud View."

Before you begin:

- Choose the Oracle MAF perspective.
- Open or create a MAF application.

To create a REST service description:

1. Either, from the main OEPE menu or from the context menu of one of the projects of the MAF application, choose **File > New > REST Service Description**.

   Or:

   a. In the Project Explorer, right-click the assembly project, and then choose **File > New > Other** from the main OEPE menu.

   b. In the **New** dialog, expand **Oracle** and choose **REST Service Description**. In the New dialog, click **Next**.

2. On the **REST Service Description** page, select the parent folder in the view project for the application to store the description. For example, a folder in the view project called **src.rest**.

   Give the description a name and click **Finish**. The REST Service Description opens in the REST Service Editor.

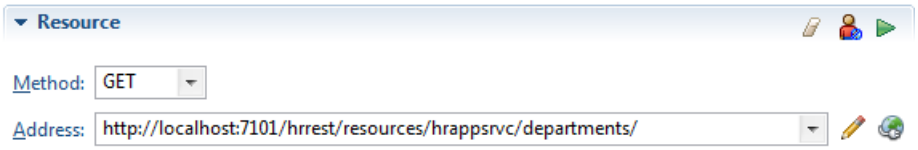
16.2 Using the REST Client

The REST Client, shown in Figure 16–1, allows you to connect to and work with a live REST service.

16.2.1 Specifying REST Service Connections

You specify the method to be used for the connection and the URI of the REST service in the Resource area at the top of the REST Client, as shown in Figure 16–6. Connections you define here are available in the Connections view, where you can edit them or reuse them elsewhere in the mobile application. For more information, see Section 16.8, "Using the Connections View."

Figure 16–6 REST Service Connection



The REST Client can use the following HTTP methods:

- GET, to retrieve a representation of a resource.
- POST, used to create new resources. You POST to the parent of the new resource you want to create, and the service takes care of creation and assigning a new id.
- PUT, used to update a representation of a resource. The input specifying the update is sent in the body of the request.
- DELETE, used to delete a resource.

The connection to the REST service is specified by a URI. Usually URIs are available for just the current session, although you can define persistent URIs which are available in future sessions. If you have used multiple URIs, all those that made valid connections are available from the address dropdown.

You can define the connection in a number of ways:

- You can simply enter a valid address for a REST API in **Address**. For more information, see Section 16.2.1.1, "How to Enter a Simple URI."
- The Compose Address functionality allows you to separately enter segments, a query, and a fragment for an address, and it handles the encoding for you. For more information, see Section 16.2.1.2, "How to Compose an Address to a REST Service" and Section 16.2.1.4, "How to Include Fragments in the Request."
- You can use connections which are defined in the application, that is a connection in the application's `connections.xml` file. For more information, see Section 16.2.1.5, "How to Use Connection Names from the Application."
- You can define persistent URIs which are available in future sessions. For more information, see

16.2.1.1 How to Enter a Simple URI

You connect to a REST service from the REST Service Editor from the REST Client page.

To enter a simple URI to a REST service:

- Enter the URI in the Address field. As you start to type the editor suggests URIs based on previous entries during the same session.

16.2.1.2 How to Compose an Address to a REST Service

You connect to a REST service from the REST Service Editor from the REST Client page. The dialog will do any encoding for any characters that cannot be directly represented in the URI.

To compose the address:

1. Click to open the Compose Address dialog.
2. In **Segments**, enter the URI.

   Alternatively, you can use connection names from the application to create the URI. For more information, see Section 16.2.1.5, "How to Use Connection Names from the Application."
3. To use a query so that the server returns just the result of the query, see Section 16.2.1.3, "How to Include Queries in the Request."
4. To use a fragment which locates a specific location at the destination, see Section 16.2.1.4, "How to Include Fragments in the Request."
5. If authentication is required, in the Resource section click to open the Authentication Information wizard. For more information, see Section 16.2.1.8, "How to Use Authentication."

16.2.1.3 How to Include Queries in the Request

You can send a query as part of the URI so that the server returns just the result of the query. You can enter the query value, or use variable that you specify when you make the connection.

To enter a query:

1. Click to open the Compose Address dialog.
2. To enter a query, enter the value in **Query**.

   For example, if you are sending the request to `http://server:port/hrrest/resources/hrappsrvc/departments` and you specifically want details about a department with the id=10, you can use the address `http://server:port/hrrest/resources/hrappsrvc/departments/10` to return a response just about this object.

   You can also use variables as queries. Enter a variable delimited by { }, for example `{id}`. When you send the request, a Replace Variables dialog is displayed where you enter the value for each variable.

16.2.1.4 How to Include Fragments in the Request

You can use a fragment as part of the URI so that the server locates a specific location at the destination.

To use a fragment:

1. Click to open the Compose Address dialog.
2. To use a fragment which locates a specific location at the destination, enter the fragment name in **Fragment**.

16.2.1.5 How to Use Connection Names from the Application

The connection URI is a specific type of URI that you can use with the REST client. The form for a connection URI is `connection://`*`connection-name`*.

In this example, the connection with the name `rest_conn` identifies a connection to `http://localhost:7101" name="rest_conn`.

```
<Reference className="oracle.adf.model.connection.url.HttpURLConnection"
name="rest_conn" xmlns="">
      <Factory className="oracle.adf.model.connection.url.URLConnectionFactory"/>
      <RefAddresses>
         <XmlRefAddr addrType="rest_conn">
            <Contents>
               <urlconnection url="http://localhost:7101" name="rest_conn"/>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
```

To use a URI based on this connection in the REST Client, type `c` in **Address**. The editor displays a list of possible URIs, and one of them will be `connection://rest_conn`. When you run the request, this connection resolves to an actual URI of `http://localhost:7101`.

In some cases, such as this example, you also need to add an additional part of the path in order to return a valid response. For example, `connection://rest_conn/departments`.

To create a URI based on the application:

1. Open the application's `connections.xml` file and identify the name of the connection you want to use. In the menu bar of the Connections view, click . connections.xml open in the OEPE source editor.
2. In the REST Client page of the REST Service Editor, start typing the connection name in **Address**.
3. Choose the connection you want from the list of those available, and add any additional part of the path.

You can use a connection URI for a connection that has authentication set in the application. In the `connections.xml` file, `HttpURLConnection` for the connection has an `adfCredentialStoreKey` set to a value.

When you click , an Authentication Information dialog is displayed where you enter the username and password associated with the connection.

16.2.1.6 How to Create Persistent Connections

You can create persistent connections, which will be available to you at any time, rather than just for the current session. Persistent addresses are available from the names you enter for them.

Persistant connections appear in the Connections view. For more information, see Section 16.8, "Using the Connections View."

To create a persistent connection:

1. In the REST Client page of the REST Service Editor, click to open the Manage Connection wizard.
2. In the dialog, shown in Figure 16–7, enter a name and the URL for the connection. In this page of the wizard, you can:
   - Test the URI by clicking .
   - Open the URI in a browser by clicking .

- If you need to specify authentication details for the connection, select **Requires Authentication** and enter the authentication values on the remaining pages of the wizard. For more information, see Section 16.2.1.8, "How to Use Authentication."
- To specify security policies, click **Show Policies** and select the appropriate one. For more information, see Section 15.3, "Accessing Secure Web Services."

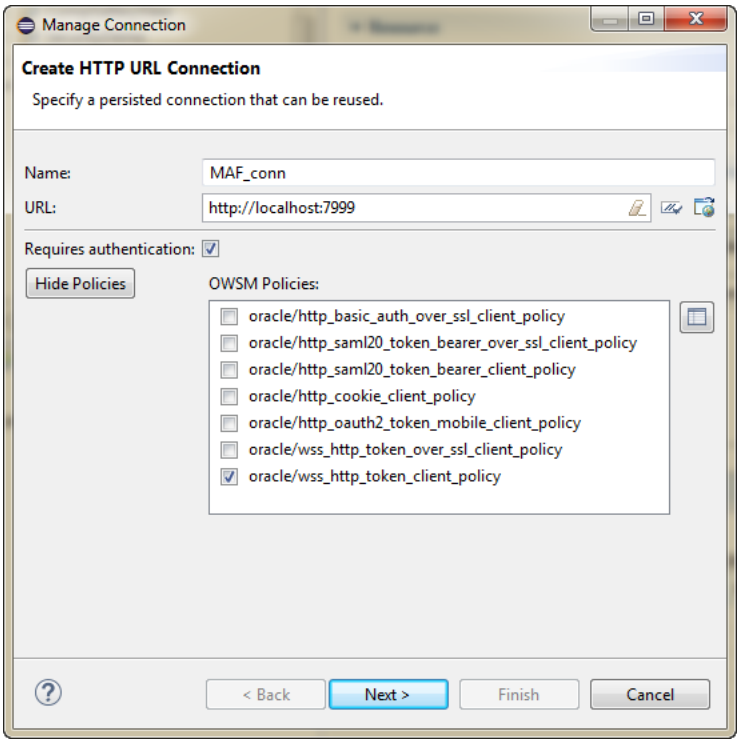
16.2.1.7 Overriding OWSM Policies

You can override some of the OWSM Mobile Agent properties to modify the underlying behavior of the OWSM policy being enforced. This is only available for the following policies:

- `oracle/http_basic_auth_over_ssl_client_policy`
- `oracle/wss_http_token_over_ssl_client_policy`
- `oracle/wss_http_token_client_policy`

You can set `preemptive` to be either `true` or `false`. Setting `preemptive` to `true` on these policies inserts a basic authentication header into the first request to a secured REST web service

Figure 16–7 Creating a Persistent Connection



To use override properties to modify an OWSM policy:

1. In the Manage Connection dialog, select the OWSM policies you want to set an overri[illegible]operty for.
2. Click to open the Override Properties dialog.
3. Click to open the Select Override Property dialog. Select from the list of available properties and click **OK**.

4. In the Override Properties dialog, choose the value for the property. Click **OK** to return to the Manage Connection dialog.

16.2.1.8 How to Use Authentication

When you connect to a REST service which is protected by authentication, you enter the authentication as part of the connection information. In the REST Client page, this is either:

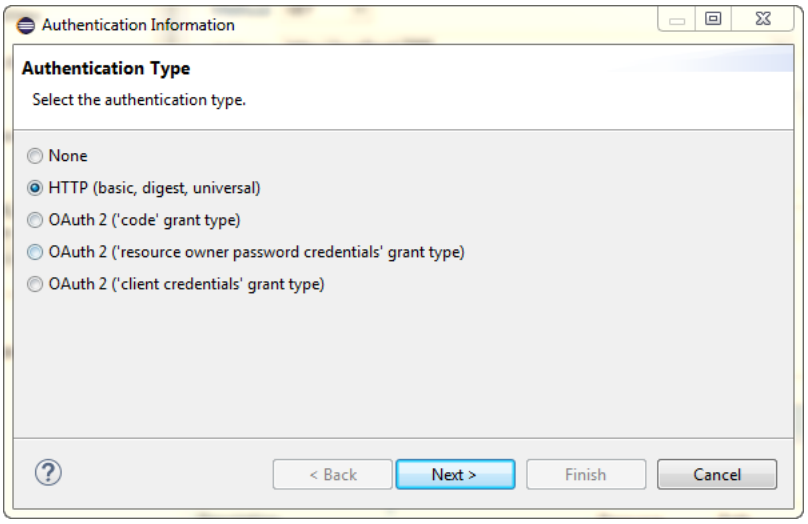
- When you creat ession connection by entering a URI in the Address field when you click to open the Compose Address dialog. In this case, click .

  Alternatively, you can use a connections URI which points to an application connection which already has authentication applied. For more information, see Section 16.2.1.5, "How to Use Connection Names from the Application,".

- When you create a persistent connection by clicking to open the Manage Connection wizard. In this case, select Requires Authentication on the create HTTP URL Connection page.

The REST Service Editor supports HTTP Basic and OAUTH 2 connections, as shown in Section 16–8, "REST Service Authentication.":

Figure 16–8 REST Service Authentication



- HTTP Basic, digest, universal: Enter a username and password to be used when sending a request to a server.
- OAuth 2 ("code" grant type): OAuth 2 authentication using an authorization URI.
- OAuth 2 ("resource owner password credentials" type): OAuth 2 authentication using a username and password.
- OAuth2 ("client credentials" grant type): OAuth 2 authentication using client credentials.

To enter authentication for a REST service connection:

1. On the REST Client page:
   - If you cr ng a session connection by entering a URI in ddress field or clicking to open the Compose Address dialog, click to open the Authentication Information dialog.

- If you create a persistent connection by clicking to open the Manage Connection wizard, select **Requires Authentication** on the create HTTP URL Connection page.

2. On the Authentication Type page, choose the type of authentication you want to use and click **Next**.
3. Enter the authentication details for the connection and click **Finish**.

You may also have to use authentication when you create Java artifacts from a modeled REST API to use with a MAF application. For more information, see Section 16.7.1, "How to Generate Java Artifacts for REST Services."

16.2.2 Sending Requests to the REST Service

Once you have specified the URI you can send the request to the server.

To send a request to the REST Service:

1. Choose the method you want to use.
2. Ensure that the correct URI is specified in **Address**.
3. Run the request by either:
   - With the cursor in **Address**, press Return.
   - Click .

16.2.3 What Happens When You Send a Request

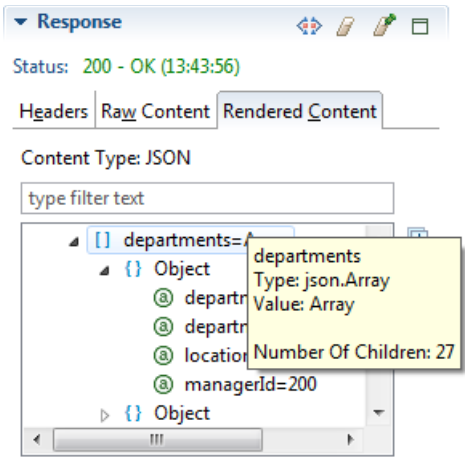
When you send the request the HTTP response returned by the server to the client is displayed in the Response area, as shown in Figure 16–9.

The HTTP status code of the response is shown.

The response itself is displayed in the three tabs:

- Headers, which shows the names and associated values of the headers, for example `Content-Length`, `ContentType`, `Date`.
- Raw Conte ws the string that comes from the server. You can wrap the lines by clicking . If the content is binary you can display it as a string by clicking .
- Rendered Content displays the response formatted according to the content type: JSON, XML, HTML, or images. Other content types are not displayed in this tab. Hover the mouse over the nodes to see additional information to help you understand the REST API.

Figure 16–9 Content of the Response



You can display the data types in the response area by clicking .

16.2.4 Generating Response Types

Because a REST API request does not specify an output, upon code generation OEPE returns `oracle.eclipse.tools.rest.runtime.client.ResponseObject` as the generated java return type. This allows you handle request response information in the most appropriate manner for your application.

You will to specifically handle the code in situations where the mediatype is either:

- Wildcard(`*/*`)
- Image(`image/*`

`ResponseObject` provides access to:

- **Response headers**—a `not-<code>null</code>` map with the response headers.
- **Mediatype**—a `not-<code>null</code>` array with either a zero length if there is no media type information or length equals `3` in which the first element is the full media type string (like `application/json`), the second element the media type `'type' (application)`, and the third the subtype `(json)`.
- **Charset**—`<code>null</code>` or the character set applicable to the response's media type.
- **Entity**—`<code>null</code>`, a not-empty string, or any object representing the entity of the response (the entity is typically a representation of the content returned by the server).
- **Raw Entity**—`<code>null</code>` or the object returned by the server as is, typically an input stream or a byte array. If this object's entity is not `{@link #isBufferedEntity() buffered}`, this method returns a `byte` array.

16.3 Modifying the Request Content

You can modify the content of the request message to determine what is returned in the response from the server. You can do this for requests in the REST Client page, and when you are modeling REST APIs in the REST API page.

You can use:

- HTTP headers
- Query parameters
- Input (when the method is PUT or POST)
- Output (when the method is GET or POST)

16.3.1 How to Use REST Headers in the Request

You can add HTTP headers to modify how the server responds to the request. For example, you can specify the content type to be returned, for example `Accept=application/json` for GET, or `Content-Type=application/json` for POST.

To specify the content type to be returned by the server:

1. With the Headers tab in the Request Details area selected, click to open the Add Header dialog.
2. In the dialog, enter the values for the request header and click **OK**. Content assist is available for the name and value fields, and will display possible options as you start typing.
3. In the Headers tab of the Request Details area, you can edit or delete a selected header, or reorder, or delete the headers using the appropriate buttons.

16.3.2 How to Use Query Parameters to Configure the Response.

You can use query parameters to configure the response the server returns, for example `verbose=false`.

To use query parameters:

1. With the Query Parameters tab in the Request Details area selected, click to open the Add Query Parameter dialog.
2. In the dialog enter the values for the query parameter and click **OK**.
3. In the Query Parameters tab of the Request Details area, you can edit or delete a selected query parameter, or reorder, or delete the query parameters using the appropriate buttons.

16.3.3 How to Send Input

If you are using the method PUT or POST, you can send input in the body of the request message.

Note: The only input on the REST Client is in the body of the request.

To send input:

1. Select the Input tab, and choose either Body or Representation.
2. For Body, either enter the input directly in the text area. For example:

   ```
   {"name":"demo department"}
   ```

   You can wrap the lines by clicking .
3. Alternatively, click **Show as parameter list** and click to open the Add Body Parameter dialog. Enter the name and value pairs, and click **OK**.

In the Input tab in the Request Details area the Query Parameters tab of the Request Details area, you can edit or delete a selected input parameter, or reorder, or delete the input parameters using the appropriate buttons.

4. For representation, click to open the Representation dialog, and choose the data type for the input.

   If necessary, deselect **All Attributes** and select just those you want to use. It is important that you select data types and attributes that can be returned by the server.

16.3.4 How to Specify Output

If you are using the method GET or POST, you can specify the output that is returned. For example, you could specify that only a few out of the available attributes of a data type are returned in the response.

To specify output:

1. Select the Output tab.
2. Choose whether the output is representation or redirection.
3. For representation, click to open the Representation dialog, and choose the data type for the response.

   If necessary, deselect **All Attributes** and select just those you want returned. It is important that you select data types and attributes that can be returned by the server.
4. For redirection, enter the required HTTP headers, and then enter the representation.

16.4 Importing and Modeling the REST API

The REST API and Data Types pages of the REST Service Editor allow you to model REST APIs. You can start by importing REST Client information from an existing live REST service which you may be developing an application to run against. Alternatively, you can create a completely new REST API, for example, to test how your application may run against a proposed REST service.

You can import a RAML definition from an MCS API. It is saved as a local REST API, allowing you to work with it.

Once you are happy with the REST API you can copy your changes to the REST Client page to run the request against a live REST service and examine the response that is returned.

16.4.1 Importing REST Client Information

Before you can model the REST API for an existing REST service, you must import REST client information from the service. You can choose to import requests, or data types, or both.

Before you begin, you must have:

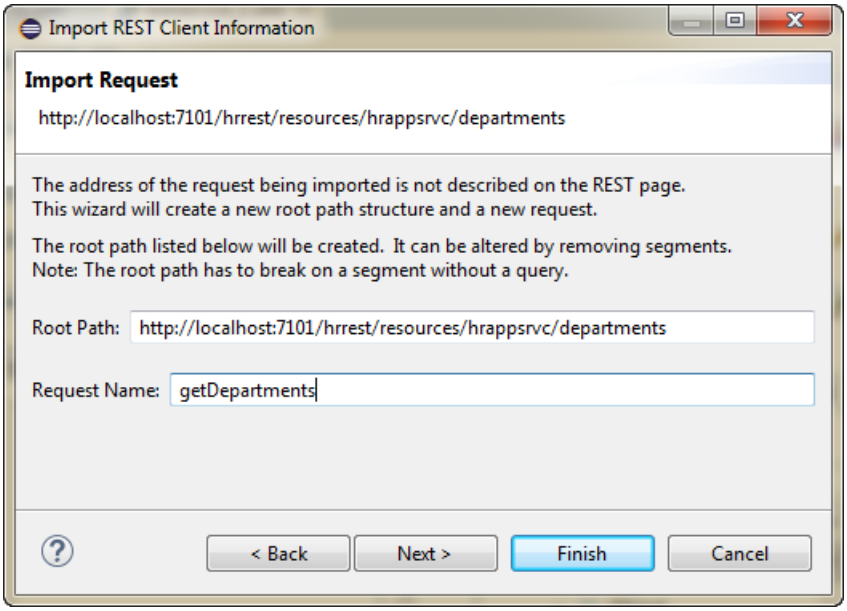
- Created a connection to the REST service in the REST Client page of the REST Service Editor, described in Section 16.2.1, "Specifying REST Service Connections."

- If you want to import data types, set an HTTP header to specify that the response is a JSON payload, described in Section 16.3.1, "How to Use REST Headers in the Request."
- Run the request against the REST service, described in Section 16.2.2, "Sending Requests to the REST Service."

To import REST Client information:

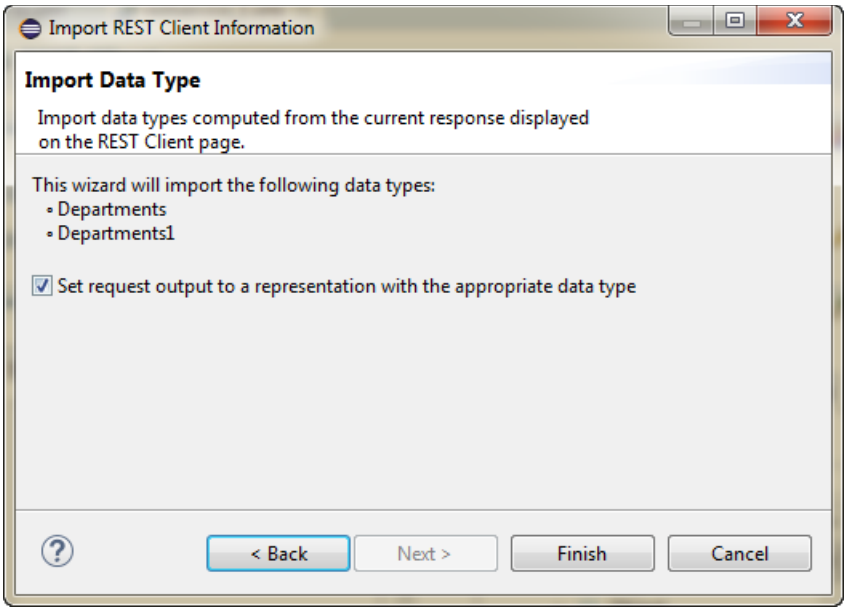
1. In the REST Client page of the REST Service Editor, click to open the Import REST Client Information wizard.
2. On the Import REST Client Information page of the wizard, choose what you want to import and click **Next**.
3. The Import Request page only appears if you are importing requests. Enter the root path and a name for the request, as shown in Figure 16–10. Click **Next**.

Figure 16–10 Importing Requests



4. The Import Data Types page only appears if you are importing data types. The wizard infers the data types from the tables and it displays them, as shown in Figure 16–11, To import the information, click **Finish**.

Figure 16–11 Importing Data Types

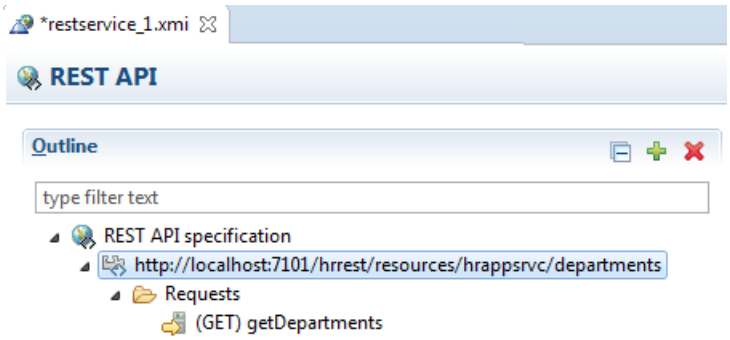


16.4.2 What Happens When You Import REST Client Information

When you import requests, or requests and data types, the REST Service Editor returns to the REST API page, where you can continue to model the REST API.

The path you specified in the wizard is shown as a node, and the request you entered is listed under it, as shown in Figure 16–12.

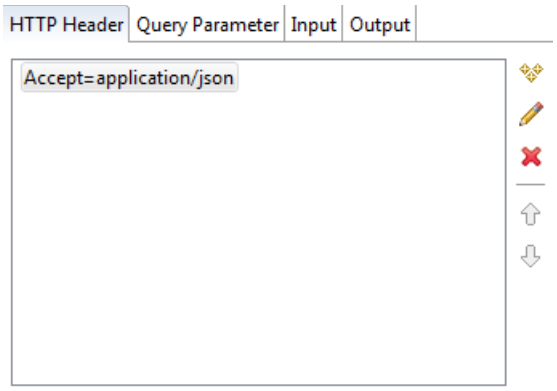
Figure 16–12 Imported Request



When you select the request, in this example `getDepartments`, you can see that:

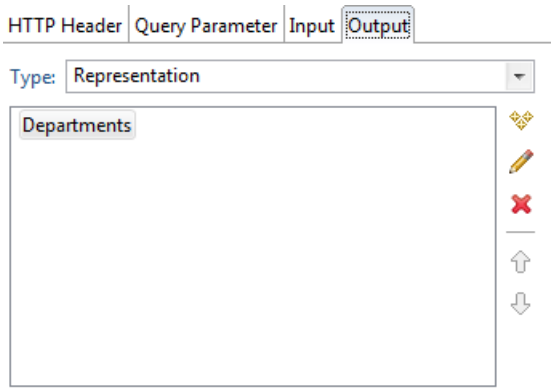
- The HTTP Header is imported, as shown in Figure 16–13.

Figure 16–13 Imported Header



- The Output is imported, as shown in Figure 16–14.

Figure 16–14 Imported Output



In addition, if you have imported data types the Data Types page will show the data types and attributes that the editor has inferred from the REST service.

16.4.3 Manually Modeling the REST API

Importing a REST API is a foolproof way of starting to model a REST API for a live service. Alternatively, you can completely model the REST API from scratch. You can also continue to model the API by:

- Specifying paths, which are addresses of REST service segments.
- Creating requests which can return responses for the server, and modify them using HTTP headers, query parameters, and specifying input and output parameters.
- Creating new data types and attributes on the Data Types page of the editor.

Once you are happy with the REST API you can copy your changes to the REST Client page to run the request against a live REST service and examine the response that is returned. For more information, see Section 16.6, "Testing Modeled Requests Against the REST Service."

Select a suitable node to further define the request.

In the HTTP Headers tab, you can:

- Define name-value pairs for the request header by clicking and entering them in the Add Header dialog.
- You can edit or delete a selected header, or reorder the headers using the buttons.

In the Query Parameter tab, you can:

- Define name-value pairs for the query parameters by clicking and entering them in the Query Parameters dialog.
- You can edit or delete a selected query parameter, or reorder the query parameter using the buttons.

Use the Input tab to send input with a PUT or POST request.

In the Output tab you can choose what happens to the results returned from the request:

1. Choose the type from none, Representation, or Redirection.
2. For Representation, click to open the Add Representation dialog where you specify the data type.

   For Redirection, enter header name value pairs in the Headers tab by clicking , then choose the Representation tab and specify the data type.

16.4.3.1 How to Create New Requests

To create a new request:

1. Right-click a node in the outline, and choose **New** and **Request**.
2. Under Request (on the right), choose a method and add a name, and define the rest of the request.

16.4.3.2 How to Create New Paths

You can create a new path to a different URI.

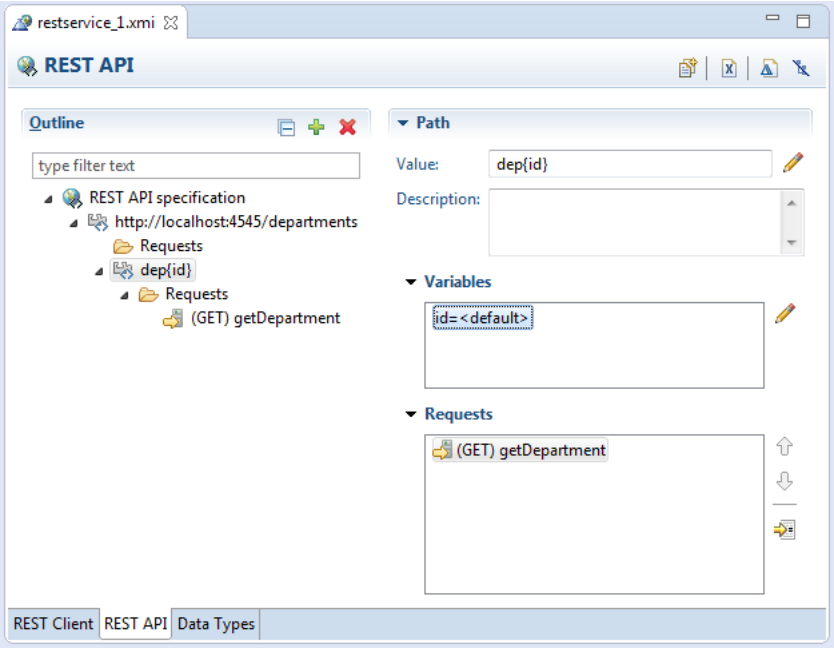
To create a new path:

1. Right-click a node in the outline, and choose **New** and **Path**.
2. Under Path (on the right), either:
   - Enter a variable using { }, for example {id}.
   - Enter a URI.
   - Click to open the Compose Address dialog where you can define the URI.

16.4.4 Working with Path Variable Types

On the REST API page of the REST Service Editor, a path segment may contain variables of the form `{variable-name}`, as shown in Figure 16–15. You can assign types to variables that are then applied in the generated code.

Figure 16–15 Using a Path Variable



For the Java domain, the default value of the variable is `java.lang.Object`.

When there are variables on a path segment that have the same name you can only assign them the default value.

The supported data types are:

- `<default>`
- `java.boolean`
- `java.byte`
- `java.char`
- `java.double`
- `java.float`
- `java.int`
- `java.long`
- `java.short`
- `java.lang.String`
- `java.math.BigInteger`
- `java.math.BigDecimal`
- `java.util.Date`
- `java.net.URI`
- `java.lang.Object`
- `java.time.LocalDateTime`
- `java.time.LocalDate`
- `java.time.LocalTime`

- `json.String`
- `json.Number`
- `json.TRUE`
- `json.FALSE`
- `json.NULL`

> **Notes:** If a variable has been assigned a data type other than the default type, and that variable is then renamed, the data type assignment is lost, and you can either undo the action, or assign the data type again.

16.4.4.1 How to Edit a Path Variable Type

To edit a path variable type:

1. Double-click on the variable in the Variables section on the right of REST API page under Path. This opens the Edit Variable Type dialog.
2. Choose the new data type for the variable from the list.

16.4.4.2 Path Variable Types in Artifact Generation

The type of a path variable is used in the artifact generation functionality of the REST Service Editor to give you more control over how code is generated as the type is used in the generated method signature. For more information about path variables and generation, see Section 16.7.1, "How to Generate Java Artifacts for REST Services."

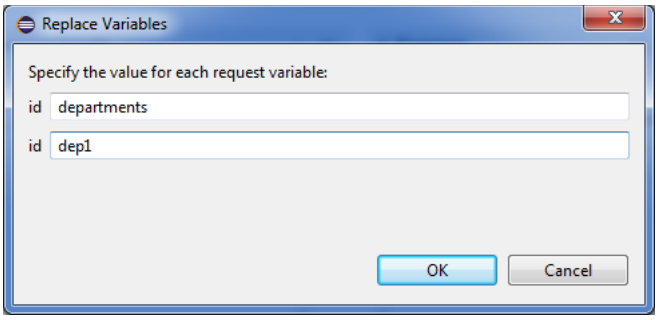
If you have specified a type for a path variable, the type is used in the generated method signature. The string value representation of the path variable will replace the corresponding path variable template value. In the case of duplicate path variables, for example `{id}/{id2}/{id}`, the method signature will contain mapping information for path variable `id` and `id2` and the string value equivalent of the type represented by the `id` parameter will replace all matching values within this path segment, and the string value equivalent of `id2` will replace its value within the specified path segment.

You can control the behavior of the parameter variable default type. In the Artifact Generation wizard described below, the pass through default behavior is to set the Parameter Type to `java.lang.Object`. So where the REST API Data type is `<default>`, and No Parameter Default Type is chosen in Generation Settings page of the wizard, the Java method parameter type signature is set to `java.lang.Object`. In the Generation Settings UI page, can control the Parameter Default Type by setting `java.lang.String` or `java.lang.Object` explicitly. This choice is persisted as an annotation within the REST Description `.xmi`, so it is a stateful choice when you re-enter the Artifact Generation wizard.

16.4.4.3 Path Variable Types and the REST Client

If you have specified types for variables with duplicate names, then when you send a request from the REST Client page of the REST Service Editor the Replace Variables dialog is displayed so that you can specify the value for each request variable, as shown in Figure 16–16.

Figure 16–16 Replacing Variables in the REST Client



You can make use of the type information on the Replace Variables dialog.

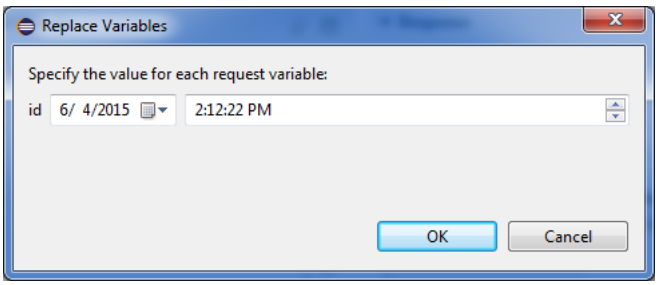
- For example, if there is a `java.boolean type` for a path variable, the dialog displays a check box.
- You can validate that the value entered for a path variable with type `java.float`, for example, conforms to the format for a Java floating point number.

In the Replace Variables dialog, `<default>`, `java.lang.String` and `java.lang.Object` are treated like strings. The JSON types are mapped to Java types:

- `json.String` is mapped to `java.lang.String`
- `json.Number` is mapped to `java.math.BigDecimal`
- `json.True` and `json.FALSE` are mapped to `java.boolean`
- `json.NULL` is mapped to `java.lang.Object`, which is treated as `java.lang.String` in the dialog.

For the `Boolean` type and for any of the date/time types (`java.util.Date`, `java.time.LocalDateTime`, `java.time.LocalDate` and `java.time.LocalTime`), the Replace Variables dialog represents these variables as shown in Figure 16–17, which illustrates `java.time.LocalDateTime` and `java.util.Date`.

Figure 16–17 Replace Variables Dialog showing Date and Time Variables

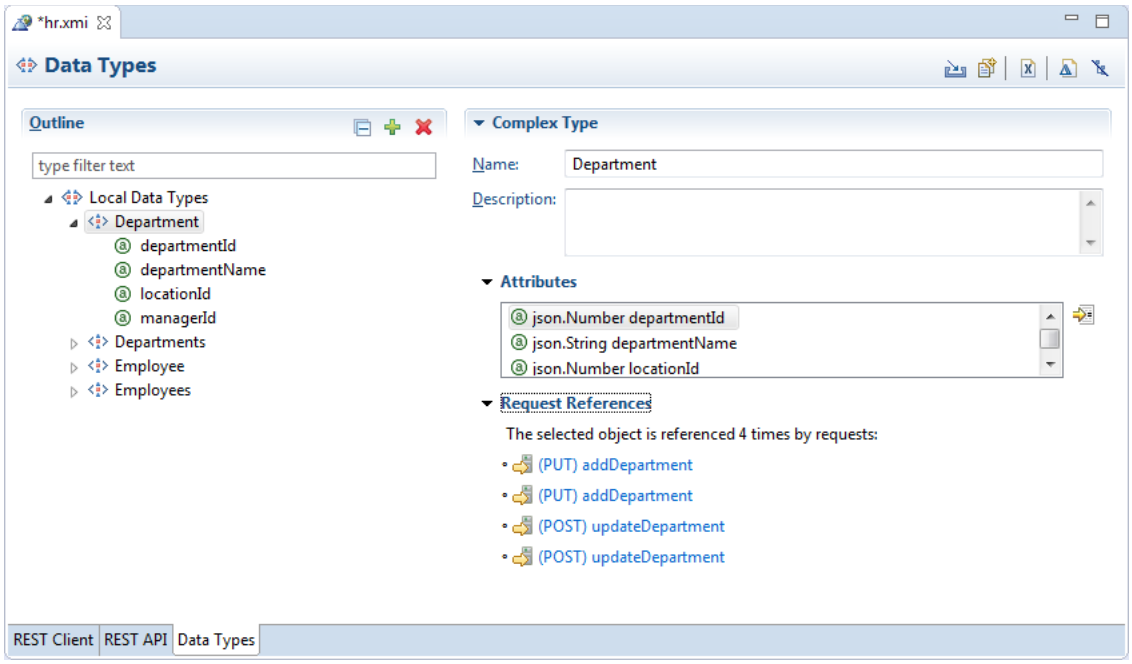


16.5 Importing and Modeling Data Types

The Data Types page of the REST Service Editor allows you to work with data types. When you use the REST Client to connect to a REST service, the editor infers the data types used by the service you are connected to, and displays them here.

You import data types associated with an API in an MCS backend, or with a RAML file on the local file system using the Data Types page of the REST Service Editor.

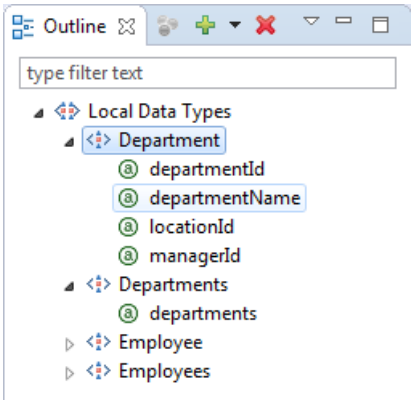
Figure 16–18 Data Types of REST API



In Figure 16–18, `Departments` is a local data type that has the attributes listed on the right.

The Outline, on the left of the Data Types page is also available in the Outline window, shown in Figure 16–19.

Figure 16–19 Outline Window



16.5.1 How to Create Data Types

You can create new complex or simple data types.

To create a complex data type:

1. Right-click **Local Data Types**, and choose **New** and **Complex Type**.
2. Under Complex Type, enter a name for the type.
3. In the outline, right-click the new type and choose **New** and **Attribute**.
4. In the Attributes area (under Complex Type on the right), click to open the Select Data Type dialog where you can choose from the list of available data types.

To create a simple data type:

1. Right-click **Local Data Types**, and choose **New** and **Simple Type**.
2. Under Simple Type, enter a name for the type.

16.5.2 How to Import Data Types

You can import data types from:

- An Eclipse workspace
- An external file
- MCS Backend API
- RAML file from local file system

To import data types:

1. In the Data Types page of the REST Service Editor, click to open the Import Data Type Information wizard.
2. In the Select a REST import mechanism, choose the source for the data types. The options are:

   JSON Schema. Choose to import data types from a JSON schema definition.

   JSON Text. Choose to import data types from a JSON example text.

   Mobile Backend's Data Type. Choose when the data types are associated with an API from MCS Backend.

   RAML File. Choose to import the data types used in a RAML file on the local file system.
3. If you have selected JSON Schema or JSON Text, click **Browse**, and in the Select Content File navigate to the location of the file containing the JSON payload. Alternatively, you can copy the JSON payload and paste it directly into the Context text area.
4. In the Select Content File dialog, enter the character set for the JSON payload, for example `UTF-8`.
5. In the Import Data Type Information wizard, click **Next**.
6. On the Import Data Type page you will see a summary of the data types that will be imported or merged.

   To merge types that match existing domain types, select that option.

   Click **Finish**.

16.6 Testing Modeled Requests Against the REST Service

You can copy requests from the modeled REST API to the REST Client page of the REST Service Editor to run them against the live REST service.

> **Note:** You may not be able to test all secured modeled requests against the REST service where OWSM Mobile agent is required. However, Basic Authorization with OWSM policy is supported.

To run the request from the modeled API:

1. In the REST API page, select the request you want to run.
2. Click , which is in the upper right of the Request area.
3. The REST Client page opens with the request ready to run.
4. Click .

16.7 Creating REST Service Artifacts

Oracle Enterprise Pack for Eclipse lets you create Java artifacts to create a MAF AMX application that runs against a REST service.

Use the REST Service Editor, described in Section 16.1, "Introduction to Working with REST Services." to connect to the REST service you want to use and use the functionality available to you to model the REST API to achieve the results you want.

The steps to perform before generating Java artifacts that you can use in a MAF application are:

1. Create a REST service description, described in Section 16.1.6, "How to Create a REST Service Description."
2. Connect to the REST service, described in Section 16.2.1, "Specifying REST Service Connections."
3. Model the REST API until you are confident that it represents the functionality you want, described in Section 16.4, "Importing and Modeling the REST API."

16.7.1 How to Generate Java Artifacts for REST Services

You generate Java artifacts in the REST Service Editor from either the REST API page or the Data Types page.

Before you begin, ensure that you have tested the requests and data types against a live version of the REST service, and that the responses from the server are what you expect.

To generate Java artifacts:

1. In the REST Service Editor, select either the REST API page or the Data Types page.
2. Click to open the Artifact Generation wizard.
3. On the Select an artifact generator page, choose the code that is compatible with the Oracle MAF runtime that you are developing the application for. Click **Next**.
4. On the BASE URL Selection page:
   - Base Resource Path is the path at the base of the REST API specification.
   - Name and URL are a persistent connection. Click to open the Manage Connection wizard.
5. In the Manage Connection dialog, enter a name and the URL for the connection. In this page of the wizard, you can:
   - Test the URI by clicking .
   - Open the URI in a browser by clicking .
   - If you need to specify authentication details for the connection, select **Requires Authentication** and enter the authentication values on the remaining pages of the wizard.

Click **Finish** to return to the RESTful Web Service wizard, and click **Next**.

6. On the Java Class and Package Names page:
   - Expand the REST API Model Path to check that it is correct.
   - Change the Java Name, if necessary.

   Click **Next**.
7. On the Generation Settings page:
   - Choose the source folder from those available.
   - Accept the package prefix, or enter a new one, or browse to a different one.

     Note: If there are any contents in this package location, they can potentially be overwritten. Oracle recommends that multiple BASE URL selections are written into different package schemes.

   - **Parameter Default Type** When the REST API Data type is `<default>`, the default is `java.lang.Object`. If necessary, set to `java.lang.String`.
   - **Generate Service Class** is selected by default. Deselect if you want to write your own.
   - **MAF Rest Classpath** Select when you are generating an API to work with MCS, so that additional dependencies are added to the project classpath and runtime deployment.

   Click **Next**.
8. On the Service Class page:
   - Examine the request listed and the method name it is associated with. To change the method name, select the request and edit the method name in **Method Name**.
   - To add a new request, click . to open the Select Requests dialog. Choose one or more requests (use Shift and Ctrl for multi select), and click **OK**.

   Click **Next**.
9. On the Summary page, review the options you have specified. When you are satisfied, click **Finish**. A Confirm Changes dialog is displayed. If you have happy to proceed, click **OK**.

16.7.2 What Happens When You Generate Java Artifacts

Code is generated for the requests you have selected for the REST service. If you have already generated artifacts and there are differences, a diff window appears allowing you to reconcile the changes you want to keep.

The generated files are listed in the Project Explorer under the package you named, shown in Figure 16–20.

16.7.3 About the Generated Artifacts for REST Services

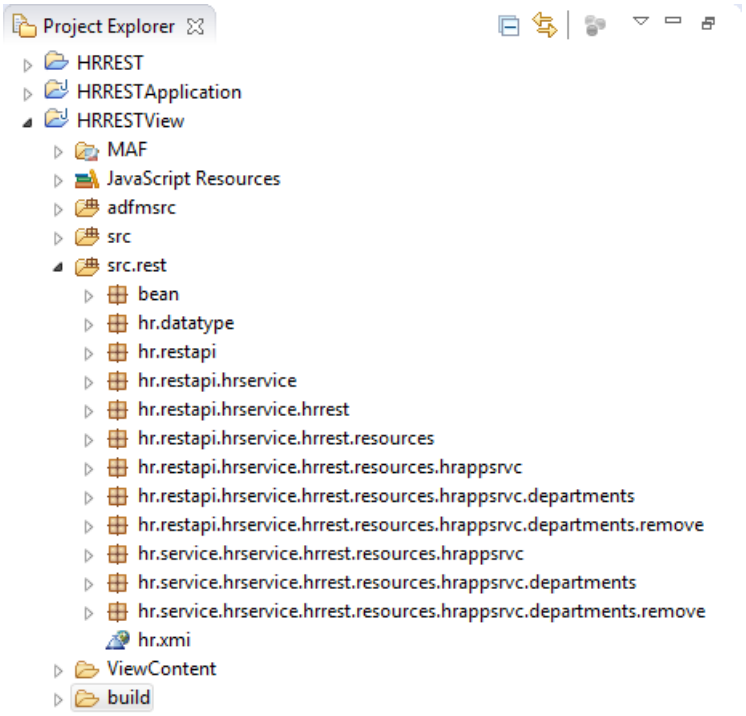
When you generate JAVA artifacts from a modeled REST API in the REST Service Editor, there are three types of package generated:

- `datatypes`.

- `restapi`
- `services`

Figure 16–20 shows the generated packages for an application called `hrrest` which have been generated into a package called `hr`.

Figure 16–20 Generated Files



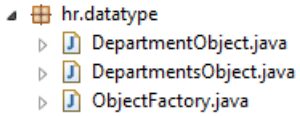
16.7.3.1 Generated Data Type Artifacts

In the datatype package, `hr.datatype` in Figure 16–20, there is:

- One interface for each data type.
  - The data type interface exposes the modeled attribute as getters and setters.
  - Instances of the data type support reflection.
- An object factory.

You can see this in Figure 16–21, where there is an interface for each data type and an object factory.

Figure 16–21 Generated Data Types Artifacts



The data type interfaces contain getters and setters. In this example, `DepartmentObject.java` contains:

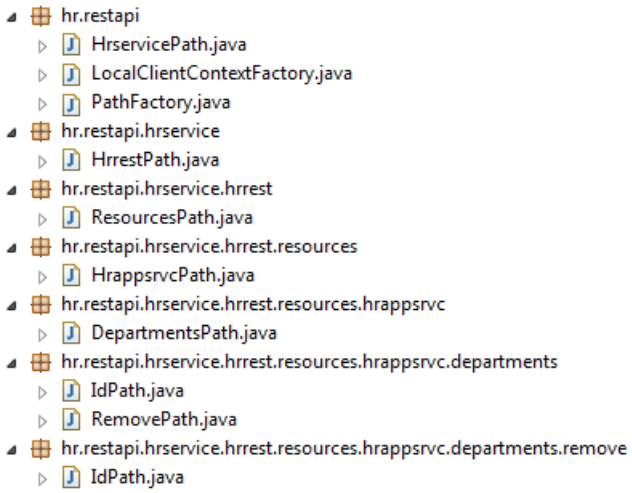
```
public interface DepartmentObject extends ObjectType {
  BigDecimal getDepartmentId();
  void setDepartmentId(BigDecimal value);
```

```
  String getDepartmentName();
  void setDepartmentName(String value);
  BigDecimal getLocationId();
  void setLocationId(BigDecimal value);
  BigDecimal getManagerId();
  void setManagerId(BigDecimal value);
}
```

16.7.3.2 Generated REST API Artifacts

The generated REST API artifacts have a path interface that has a nested interface that exposes the requests as Java methods. A number of REST API packages are generated, as shown in Figure 16–22.

Figure 16–22 Generated REST API Artifacts



The Request interfaces contain:

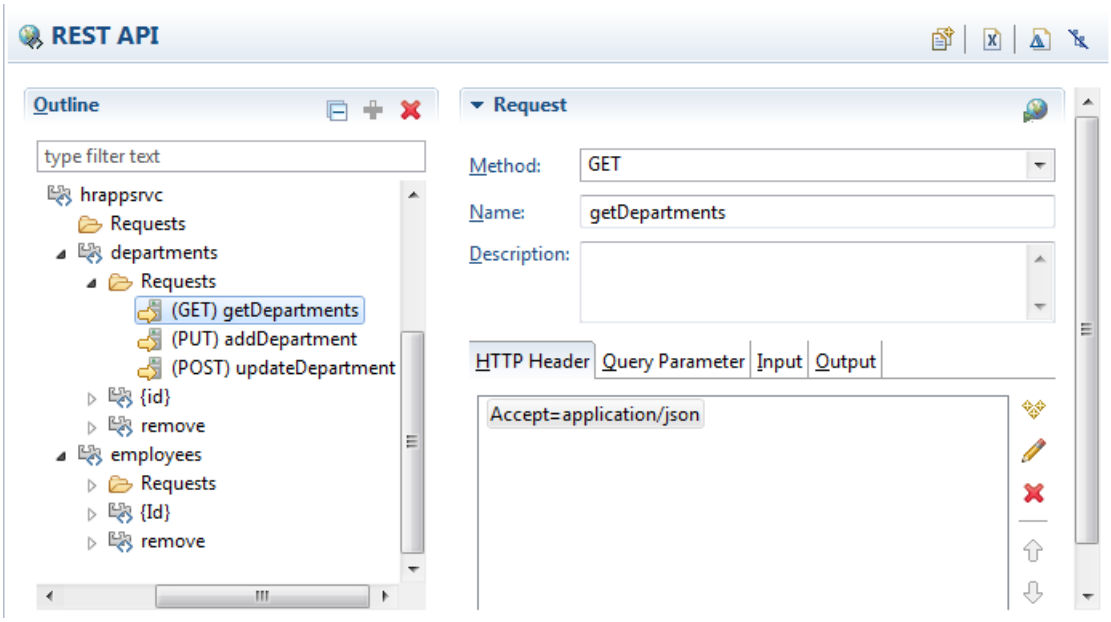
- A method for each request to the REST service, for example `createDepartmentJSON`
- The Invoke method that exposes the class

For example, `IdPath.java` in `hr.restapi.hrservice.hrrest.resources.hrappsrvc.departments` contains:

```
public interface IdPath extends PathObject {
  @RequestAccessor
  public static interface Request {
    @Action(HTTPMethod.GET)
    @Header(name="Accept", value="application/json")
    @Output(representations=@Representation(type=DepartmentsObject.class))
    DepartmentsObject getDepartmentById() throws RequestException;
  }
  @Override
  DepartmentsPath parent();
  /**
   * @return creates a new object that exposes the requests
   *         defined for this path
   */
  Request invoke();
}
```

It relates to the `getDepartments` request in the REST API page of the REST Service Editor, shown in Figure 16–23, where the HTTP header is `Accept=application/json`, and the output is representations of the data type `Departments`.

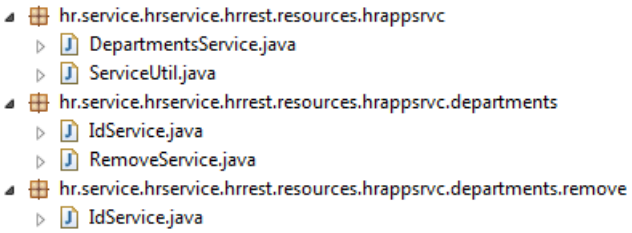
Figure 16–23 getDepartments in REST API Page



16.7.3.3 Generated Service Artifacts

The generated services packages, shown in Figure 16–24, contains one POJO for each path that defines a request, (apart from the root which does not have a requests). The POJO is a class you can use with Data Controls, and it gives guidance on how to use the content of other packages.

Figure 16–24 Generated Service Artifacts



In the example, `DepartmentsService.java` is a POJO and it contains a method for each request method on the REST API path.

```
public class DepartmentsService {
  public DepartmentsObject getDepartments() throws Exception {
    try (ClientContext context =
LocalClientContextFactory.INSTANCE.create(ServiceUtil.CONNECTION_NAME)) {
      return
PathFactory.INSTANCE.createHrservicePath(context).path(context.getConnectionPath()
)
        .getHrrestPath()
        .getResourcesPath()
        .getHrappsrvcPath()
```

```
        .getDepartmentsPath()
        .invoke()
        .getDepartments();
    }
  }
  public DepartmentObject addDepartment(DepartmentObject department) throws
Exception {
    try (ClientContext context =
LocalClientContextFactory.INSTANCE.create(ServiceUtil.CONNECTION_NAME)) {
      return
PathFactory.INSTANCE.createHrservicePath(context).path(context.getConnectionPath()
)
        .getHrrestPath()
        .getResourcesPath()
        .getHrappsrvcPath()
        .getDepartmentsPath()
        .invoke()
        .addDepartment(department);
    }
  }
  public DepartmentObject updateDepartment(DepartmentObject department) throws
Exception {
    try (ClientContext context =
LocalClientContextFactory.INSTANCE.create(ServiceUtil.CONNECTION_NAME)) {
      return
PathFactory.INSTANCE.createHrservicePath(context).path(context.getConnectionPath()
)
        .getHrrestPath()
        .getResourcesPath()
        .getHrappsrvcPath()
        .getDepartmentsPath()
        .invoke()
        .updateDepartment(department);
    }
  }
}
```

16.7.4 How to Use the Generated Artifacts in Your MAF Application

To allow your MAF applications to interact with REST services, you use the REST Service Editor to model the REST API and then you can generate artifacts to use in your applications.

The data logic is exposed through a JavaBean data control which you create from the generated service POJO.

To create a JavaBean data control for a REST service:

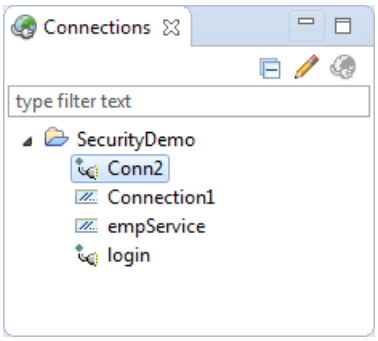
1. Select the generated service POJO described in Section 16.7.3.3, "Generated Service Artifacts."
2. In the Project Explorer or Package Explorer right-click the service file and choose **Model Components > Create Data Control** to open the New Data Control wizard.
3. Complete the wizard. The JavaBean data control is created and you can use it in your MAF application.

16.8 Using the Connections View

The Connections View displays the connections defined by a MAF application that are shared between the editors and artifacts of a MAF application. For example, connections that are used in the REST Client or for security.

You create the connection in the editor or tool in which is it first used, for example in the REST Client window and use the Connections View to edit subsequently the connection, or to reuse the connection elsewhere in the MAF app.

Figure 16–25 MAF App Connections in the Connections View



The Connections view lists all available projects that can host a connection. So a MAF application that uses the MAF 2.0.2 runtime or later will be listed in the view. An ADF application or a MAF application that uses an earlier runtime will not be listed. As shown in Figure 16–25, under the node for the application the connections are listed. They are:

- Connections of type URL, which use the icon.
- Mobile login connections which use any of the available authentication types: HTTP Basic, Web Single Sign-on, and OAuth2. These use the .

You can double click a connection name to open the Manage Connection wizard, where you can edit various settings of the connection.

The content of these connections is stored in `connections.xml` for the project, and that file is updated when you make changes from the Connections view. If you delete a project, the node for that project is removed from the Connections view.

16.8.1 How to Open the Connections View

The Connections view is available with the Oracle MAF perspective. If it is closed, you can open it using **Window > Show View > Connections** from the main menu.

16.8.2 How to Edit a Connection

In the Connections view, expand the node for the project and double-click the connection you want to edit.

Alternatively, select the connection and click .

If a connection is invalid, for example if the authentication details are not valid, the connection has a red cross at the node. You can edit the connection to provide the correct details.

16.8.3 How to Reuse a Connection

In the Connections view, expand the node for the project and select the connection you want.

Drag the connection to the appropriate field where you want to reuse it.

16.8.4 How to Open connections.xml

In the Connections view, select the node for the project and click .

The `connections.xml` file opens in the OEPE source editor.

17

Working with Oracle Cloud Services

This chapter describes how to connect to and use Oracle Mobile Cloud Service to develop mobile services and mobile applications. It also describes how a mobile application can access data hosted on Oracle Java Cloud Service.

This chapter includes the following sections:

- Section 17.1, "Introduction to Working with Oracle Cloud Services"
- Section 17.2, "Introduction to Working with MCS"
- Section 17.3, "Enabling Mobile Applications to Access Data Hosted on Oracle Cloud"

17.1 Introduction to Working with Oracle Cloud Services

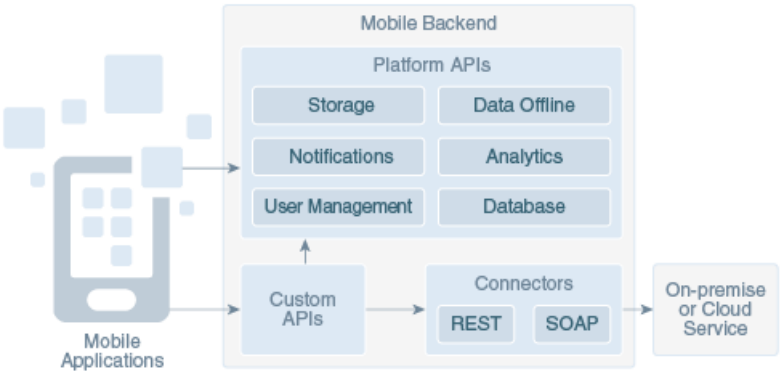
You can use OEPE to work with Oracle Cloud services to develop and deploy groupings of APIs that are designed to support a specific set of applications.

17.2 Introduction to Working with MCS

Oracle Mobile Cloud Service enables you to develop and deploy groupings of APIs that are designed to support a specific set of applications. These APIs are available in the MCS mobile backend. You can develop mobile applications which you associate with the mobile backend to access those APIs.

Figure 17–1 illustrates the overall architecture of Oracle Mobile Cloud Service and how it works with mobile applications and enterprise services.

Figure 17–1 MCS Architecture



The mobile backend for MCS contains the platform APIs, custom APIs and connectors, including REST.

For more information about MCS, see http://cloud.oracle.com/mobile

17.2.1 Getting Started with MCS

OEPE allows you to create mobile applications that use services provided on an MCS backend.

Before you can sign in to an Oracle Cloud service, you must have a user account that defines your credentials (user name, password, and identity domain) and access rights to the service.

For information about requesting a trial subscription to an Oracle Cloud service, or purchasing a subscription to an Oracle Cloud service, see *Getting Started with Oracle Cloud*.

When an Oracle Cloud service is activated, Oracle sends the sign-in credentials and URL to the designated administrator. The administrator then creates an account for each user who needs access to the service. Check your email or contact your administrator for your account credentials and service URL.

Once you have a subscription to MCS, you create a connection to MCS following the instructions in "Integrating Oracle Cloud Services" in *Oracle Enterprise Pack for Eclipse User's Guide*.

17.2.1.1 Consuming MCS Services in MAF Applications

You can use mobile services created in MCS to build mobile apps.

In this case, the MCS APIs you will typically use include:

- Tooling APIs for:
  - Asset Catalog (for APIs, API implementations, mobile backends, connectors)
  - Notifications
  - UI Tooling for Mobile API test invocation
  - User management for roles, user names, to use for testing the service
- Collections APIs to access from custom code
- System API for integrating notifications from the custom code

The steps you perform include the following:

1. Search for the API you want to use.
2. Implement the API:
   a. Download the scaffolding.
   b. Unzip and open the Javascript file in OEPE.
   c. Implement the endpoints in OEPE by editing methods in the Javascript file:

      Get the connectors available for the custom code to use.

      Get the notification API for integration within the custom code.
   d. Rezip the scaffolding and upload it to the implementations for the service.

3. Test the API endpoints from the IDE by selecting the mobile backend and the backend version. Unless the API is set to allow unauthenticated access, you will have to use a user name and password.

17.2.1.2 Developing Mobile Clients for MCS Mobile Services

You can create mobile applications that use services provided on an MCS backend.

- Tooling APIs for:
  - Asset Catalog (for APIs, policies, notifications)
  - Storage
  - UI Tooling for Mobile API test invocation
  - User management for roles, user names, to use for testing the service
- Platform APIs for storage collection
- Custom APIs for runtime URLs of the services defined in MCS by the customer

The steps you perform include the following:

1. Authenticate with MCS.
2. Download the Mobile Cloud Service SDK and start the project.
3. Search for the mobile backend you want to use.
4. Get the selected APIs for the mobile backend, including URL information and either a RAML or Swagger description.
5. Get the storage collections for the mobile backend.
6. Develop a Java client for REST API and the mobile client.
7. Test the Java client in OEPE. Unless the API is set to allow unauthenticated access, you will have to use a user name and password
8. Run and test the mobile client.

17.2.1.3 Using Mobile Backends

The resources for mobile applications including APIs are stored in a container called a mobile backend. Within the mobile backend you register the apps that can access the mobile backend and you select the PAIs that you want available for those apps.

You can have multiple backends, each serving a set of applications. In addition, you can have APIs that are used by multiple backends.

When an app accesses APIs through MCS, it is always in the context of a mobile backend. The app authenticates with the mobile backend credentials as well as with a user name and password (or an anonymous access key). If the called API includes calls to other APIs within the backend, the identity and credentials of the original caller are propagated through the chain of calls.

For more information about mobile backends, and using Oracle Mobile Cloud Service as a platform, see "Using Oracle Mobile Cloud Service" in *Using Oracle Mobile Cloud Service*

17.2.1.4 Importing a REST API

In order to work with a REST API you import a RAML definition from an MCS API. There are two ways that you can do this:

- From a connection in Oracle Cloud view.

- Using the REST Service Editor.

Before you begin:

You need to have a connection from OEPE to MCS in the Oracle Cloud view.

17.2.1.4.1 Importing a REST API from Oracle Cloud View

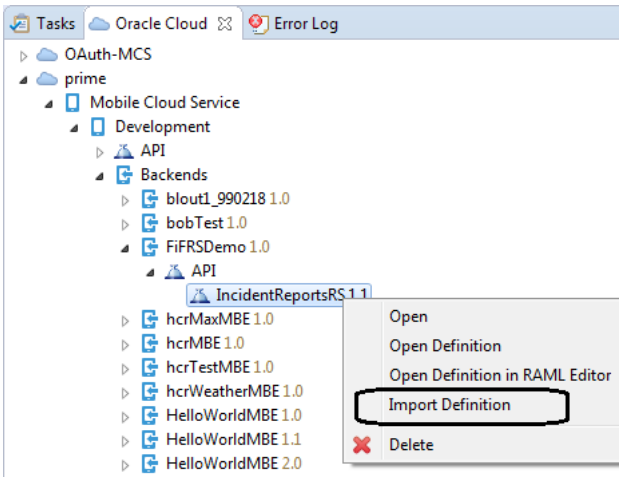
To import a definition:

1. In the Oracle Cloud view, expand the connection and Oracle Cloud Service.

   To import a definition from API, expand that node to show the API node you want.

   To import a definition from MCS mobile backends, expand that node and choose the backend you want to show the API node, as shown in Figure 17–2.

Figure 17–2 Importing a REST API



2. Right click and choose **Import Definition**.

   The Import Definition dialog opens.

3. On the REST Service Description of the wizard, choose the parent folder where the REST Service Description file will be created, and enter a file name.

4. On the Backend information page you choose the backend that is associated with the API. This page also shows whether Anonymous access is supported if basic authentication is enabled.

   If you have selected the API from the API node, choose the backend from the list of those associated with the API.

   If you chose the API from a specific backend then that backend is displayed and you cannot change it.

5. The RAML Preview page of the wizard allows you to choose the options to use during import of the REST API. These include:

   - (Only when importing from API) The method to use if there are conflicts when you are importing a RAML definition to an existing REST Service description file.

     Merge: Use when the existing methods should be merged with new methods from RAML definition.

     Replace: Use when the new method should replace the existing methods.

Create: Use when a new method should be created with new name in case of conflicts with existing methods.

- The resources and methods to import. The wizard shows a tree created from the information in the RAML definition in the API. By default, all resources and methods are selected. Deselect those that you do not want to import.
- You can launch the connection wizard if you want to edit the Base URI and store the information in `connections.xml`. The connection wizard will be opened when you click **Finish**.

  The connection wizard will be populated with data fetched from the MCS API, and the connection defined in the wizard is used to create the root path of the REST Service description.

17.2.1.4.2 Importing a REST API Using the REST Service Editor You can import RAML definition from:

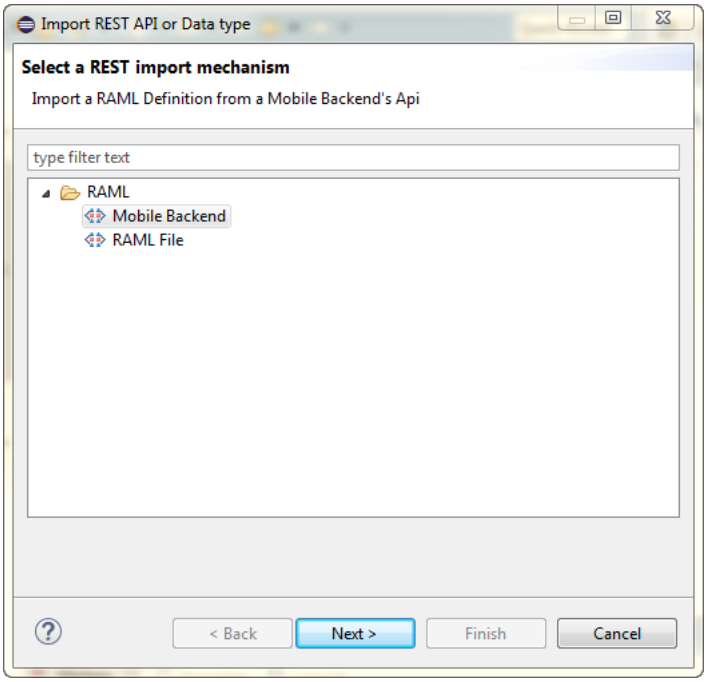
- A Mobile Cloud Service backend's API. You must have a connection to Oracle Mobile Cloud Service.
- A RAML file.

For more information about the REST Service Editor, see Section 16, "Working with REST Services."

To import a REST API:

1. On the REST API tab of the REST Service Editor, click to open the Import REST API or Data type dialog, shown in Figure 17–3.

Figure 17–3 Importing RAML Definitions



2. Choose the source of the RAML definition: Mobile Backend, or RAML File.
3. Click **Next** and complete the wizard.
   - Mobile Backend.

On the Backend information page you choose the backend that is associated with the API. This page also shows whether Anonymous access is supported if basic authentication is enabled.

If you have selected the API from the API node, choose the backend from the list of those associated with the API.

If you chose the API from a specific backend then that backend is displayed and you cannot change it.

- RAML File. Choose the file containing the RAML definition from the file system.

4. The RAML Preview page of the wizard allows you to choose the options to use during import of the REST API. These include:

   - The method to use if there are conflicts when you are importing a RAML definition to an existing REST Service description file.

     Merge: Use when the existing methods should be merged with new methods from RAML definition.

     Replace: Use when the new method should replace the existing methods.

     Create: Use when a new method should be created with new name in case of conflicts with existing methods.

   - The resources and methods to import. The wizard shows a tree created from the information in the RAML definition in the API. By default, all resources and methods are selected. Deselect those that you do not want to import.

   - You can launch the connection wizard if you want to edit the Base URI and store the information in `connections.xml`. The connection wizard will be opened when you click **Finish**.

     The connection wizard will be populated with data fetched from the MCS API, and the connection defined in the wizard is used to create the root path of the REST Service description.

17.3 Enabling Mobile Applications to Access Data Hosted on Oracle Cloud

Mobile applications can access RESTful web services by creating a web service data control. Depending on the content type, mobile applications can access cloud data by dragging and dropping a data control into a MAF AMX user interface component, as described in Section 11.3.2, "How to Add UI Components and Data Controls to an MAF AMX Page," or programmatically, for applications whose content is delivered from a either a remote web server, or from locally stored HTML files.

17.3.1 How to Authenticate Against Oracle Cloud

You use the MAF Login Server Connection dialog to create a login server connection to authenticate against Oracle Cloud.

Before you begin:

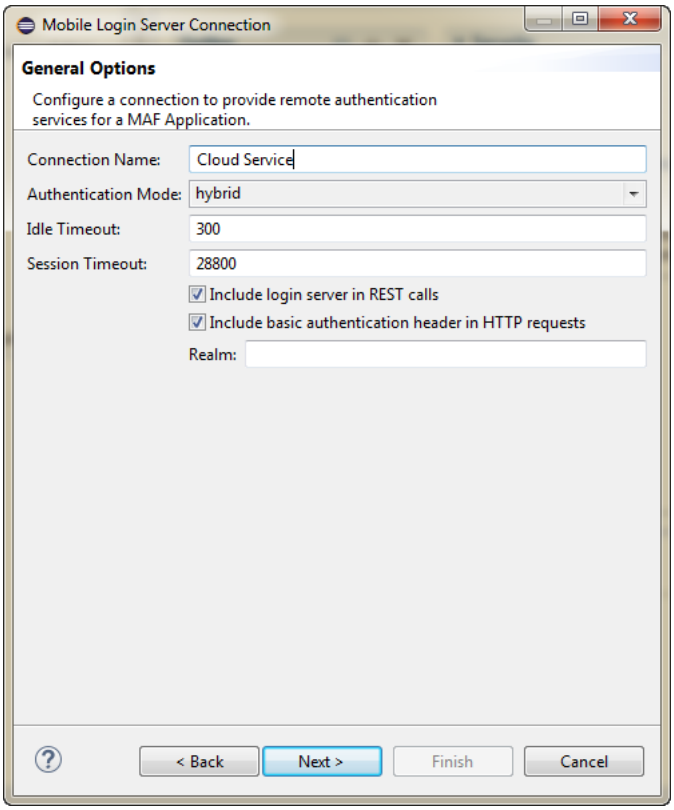
Obtain the Oracle Cloud URL that is used for the login server connection.

To create a login URL with an Oracle Cloud endpoint:

1. In the Project Explorer, expand the assembly project, then MAF and double-click MAF Application Editor.

2. In the editor, select **Security** in the outline, and then click **Create** next to **Default Login Server**.
3. Complete the General Options page of the Mobile Login Server Connection dialog, shown in Figure 17–4, by entering a name for the connection in the **Connection Name** field.

Figure 17–4 Creating the Login to Oracle Cloud



4. Click **Next**. In the HTTP Basic Options page, enter the URL for Oracle Cloud in the **Login URL** field.

   Test the connection by clicking **Test**.

   For more information, refer to Section 29.6.2, "How to Designate the Login Page."

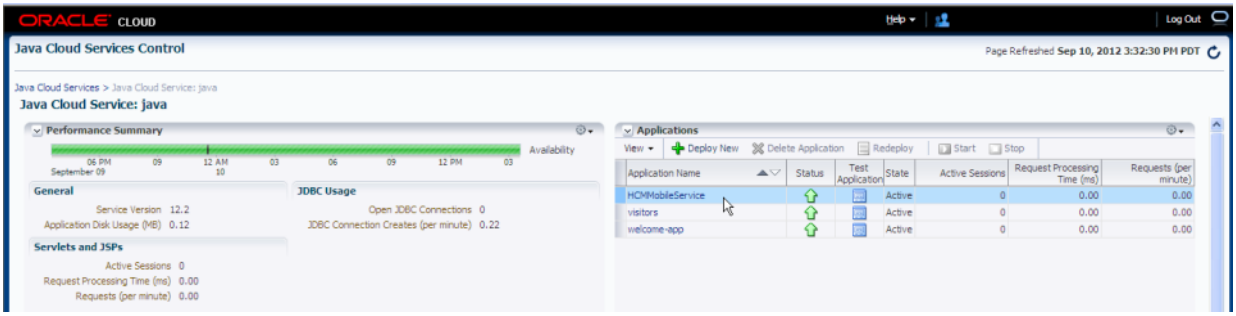
17.3.2 How to Create a Web Service Data Control to Access Oracle Java Cloud

The Create Data Service Control Wizard enables you to create the data control that accesses the hosted data. You use the WSDL URL of the web service deployed to Oracle Java Cloud to create this data control. If you do not know this URL, then you must create the URL to the WSDL document by appending the web service port name and `?wsdl` to the application context root.

Before you begin:

You must have access to a web service application that has been deployed to Oracle Java Cloud Service. This application must be available through the Applications pane of the Oracle Java Cloud Service Control home page. In addition, its Status and State must be noted as both Up and Active, respectively, as illustrated by the HCMMobileService application shown in Figure 17–5.

Figure 17–5 The Java Cloud Services Control Home Page



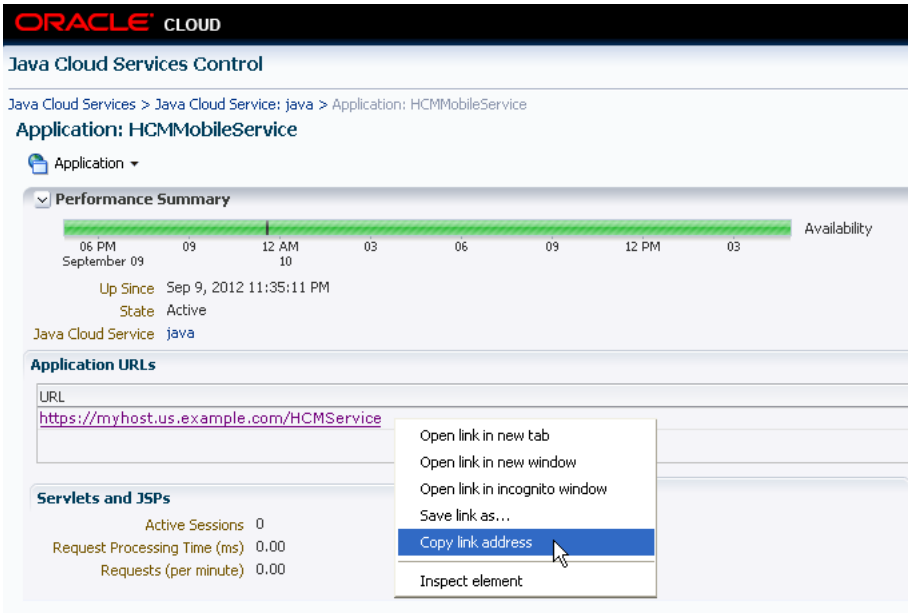
To create a web service data control:

1. Obtain the application context root of the web service hosted on Oracle Cloud as follows:

   a. Traverse to the application home page, shown in Figure 17–6, by clicking the application in the Applications pane (shown in Figure 17–5).

   b. Copy the URL, as shown in Figure 17–6. This URL is the application context root of the WSDL document.

Figure 17–6 Copying the Web Service Application Context Root

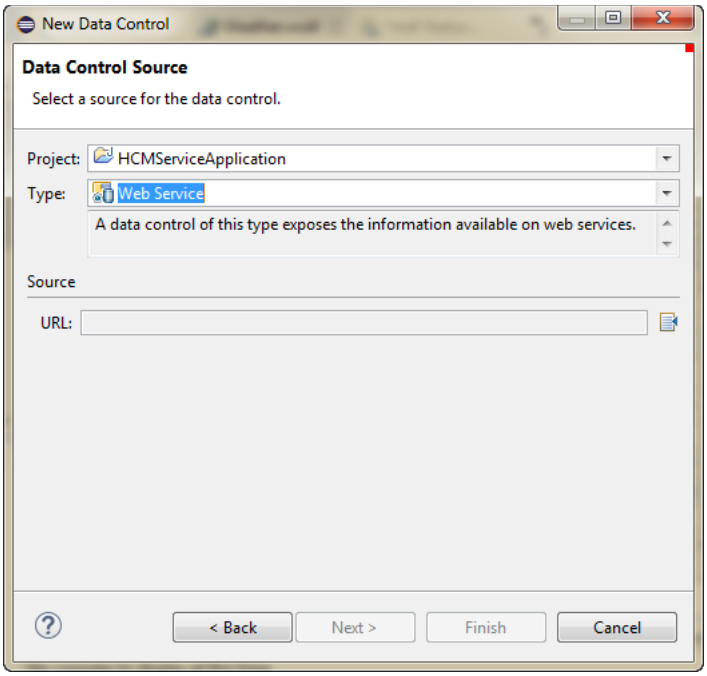


2. Choose **File** > **New** > **Other** to open the New dialog.

3. Expand **Oracle** > **Mobile Application Framework** and choose **Data Control** and click **Next**.

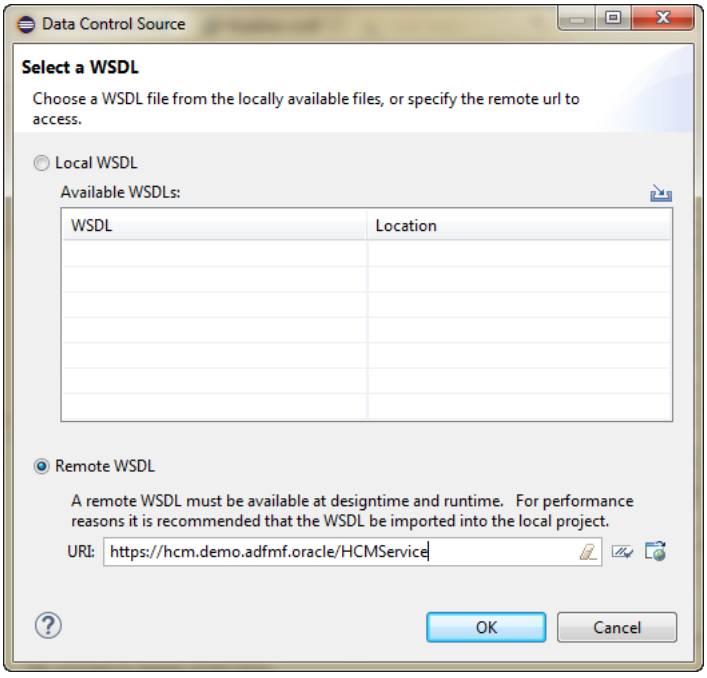
4. In the New Data Control wizard, shown in Figure 17–7, choose the project, and then select `Web Service` as the **Type**.

Figure 17–7 Creating a Data Control



5. Click next to **URL** to open the Data Control Source dialog, as shown in Figure 17–8.

Figure 17–8 Entering the URL for the WSDL Document



6. In the **URL** field, paste the URL of the web service that is deployed to (and currently running on) Oracle Cloud Java Service and click **OK**, and on the Data Control Source page, click **Next**.

7. On the Data Control Details page of the wizard, set the id for the new data control, and click **Next**.
8. On the Web Service Data Control page of the wizard, choose the services you want and click Next.
9. Review the options you have chosen in the summary page of the wizard, then click **Finish**.

17.3.2.1 Configuring the Policy for Web Services

You must configure a policy for a web service that is secured on Oracle Cloud. Using the Edit Data Service Control Policies dialog, described at Section 15.3, "Accessing Secure Web Services," you can select the `oracle/wss_http_token_over_ssl_client_policy`. For descriptions of this (and other) policies, see "Determining Which Predefined Policies to Use" and "Predefined Policies" chapters in *Oracle Fusion Middleware Securing Web Services and Managing Policies with Oracle Web Services Manager*.

17.3.3 What Happens When You Deploy a Mobile Application that Accesses Oracle Java Cloud Service

After you deploy the application, the operations of the web service data control retrieve the data from a web service running on the Oracle Java Cloud Service instance.

18

Creating Custom MAF AMX UI Components

This chapter describes how to create custom MAF AMX UI components and specify them as part of the development environment.

This chapter includes the following sections:

- Section 18.1, "Introduction to Creating Custom UI Components"
- Section 18.2, "Using MAF APIs to Create Custom Components"
- Section 18.3, "Creating Custom Components"

18.1 Introduction to Creating Custom UI Components

Using a combination of JavaScript and APIs provided by MAF, you can create new, fully functional interactive UI components and add them to a tag library to be used in your MAF AMX application feature.

18.2 Using MAF APIs to Create Custom Components

MAF provides the following APIs for creating custom components:

- Static APIs (see Section 18.2.1, "How to Use Static APIs")
- `AmxEvent` Classes (see Section 18.2.2, "How to Use AmxEvent Classes")
- `TypeHandler` (see Section 18.2.3, "How to Use the TypeHandler")
- `AmxNode` (see Section 18.2.4, "How to Use the AmxNode")
- `AmxTag` (see Section 18.2.5, "How to Use the AmxTag")
- `VisitContext` (see Section 18.2.6, "How to Use the VisitContext")
- `AmxAttributeChange` (see Section 18.2.7, "How to Use the AmxAttributeChange")
- `AmxDescendentChanges` (see Section 18.2.8, "How to Use the AmxDescendentChanges")
- `AmxCollectionChange` (see Section 18.2.9, "How to Use the AmxCollectionChange")
- `AmxNodeChangeResult` (see Section 18.2.10, "How to Use the AmxNodeChangeResult")
- `AmxNodeStates` (see Section 18.2.11, "How to Use the AmxNodeStates")
- `AmxNodeUpdateArguments` (see Section 18.2.12, "How to Use the AmxNodeUpdateArguments")

18.2.1 How to Use Static APIs

Table 18–1 lists static APIs that you can use to create custom UI components.

Table 18–1 Static APIs

Return Type	Function Name	Parameters	Description

`Function`	`adf.mf.api.amx.TypeHandler.register`	`String namespaceUrl,`
		`String tagName,`
		`adf.mf.api.amx.TypeHandler precreatedClass`
			Registers a `TypeHandler` class with a tag namespace and name.
			Returns the registered `adf.mf.api.amx.TypeHandler` subclass so that prototype functions can be added.
			The `precreatedClass` is optional but can be used if you first create a class that inherits from `adf.mf.api.amx.TypeHandler`.

`void`	`adf.mf.api.amx.addBubbleEventListener`	`Node domNode,`
		`String eventType,`
		`Function listener,`
		`Object eventData`
			Registers a bubble event listener (such as tap, taphold, keydown, touchstart, touchmove, touchend, focus, blur, resize, and so on).
			Note that web browsers do not support all event types on all DOM nodes (see the browser documentation for details).
			The `eventData` is optional and serves as extra data to be made available to the listener function.

`void`	`adf.mf.api.amx.removeBubbleEventListener`	`Node domNode,`
		`String eventType,`
		`Function listener`
			Unregisters a bubble event listener that was added through `adf.mf.api.amx.addBubbleEventListener`.
			Note that the removal of the meta events tap and taphold will cause all touchstart and touchend listeners, including those of other meta events, to become removed from the element as opposed to only the specified listener being removed.

Table 18–1 (Cont.) Static APIs

Return Type	Function Name	Parameters	Description
void	adf.mf.api.amx.addDragListener	Node domNode, Object payload, Object eventData	Allows an element to trigger MAF AMX drag events. The Object payload defines three member functions: start, drag, end. The first parameter of each function is the DOM event, the second parameter is a dragExtra Object with the following members: ■ eventSource: the DOM event source. ■ pageX: the x coordinate of the event. ■ pageY: the y coordinate of the event. ■ startPageX: the original pageX. ■ startPageY: the original pageY. ■ deltaPageX: the change in pageX. ■ deltaPageY: the change in pageY. ■ originalAngle: if defined, it is the original angle of the drag in degrees where 0 degrees is East, 90 is North, -90 is South, 180 is West. and the following modifiable member flags: ■ preventDefault ■ stopPropagation The eventData is optional and serves as extra data to be made available to the listener functions.
void	adf.mf.api.amx.removeDomNode	Node domNode	Removes a DOM node and its children, but prior to that removes event listeners added through adf.mf.api.amx.addBubbleEventListener.
void	adf.mf.api.amx.emptyHtmlElement	Node domNode	Empties an HTML element by removing children DOM nodes and calling adf.mf.api.amx.removeDomNode on each of the children nodes.

Table 18–1 (Cont.) Static APIs

Return Type	Function Name	Parameters	Description
void	adf.mf.api.amx.processAmxEvent	adf.mf.api.amx.AmxNode amxNode, String amxEventType, String attributeValueName, String newValue, AmxEvent amxEvent, Function finishedCallback	Processes an AmxEvent. Change the value if attributeValueName is defined, process the appropriate setPropertyListener and actionListener subtags, and then process the [amxEventType]Listener attribute. If a finished callback is provided, it will be invoked once the event is processed.
void	adf.mf.api.amx.acceptEvent	None	Determines whether it is safe to proceed with invoking adf.mf.api.amx.processAmxEvent in order to avoid preparing anything you might need to pass into that function (for example, when in the middle of a page transition or in an environment such as a design-time preview).
void	adf.mf.api.amx.invokeEl	String expression, Array<String> params, String returnType, Array<String> paramTypes, Function successCallback, Function failureCallback	Represents a utility similar to adf.mf.el.invoke() for invoking an EL method, with a difference that it refrains from execution in environments such as design-time previews.
void	adf.mf.api.amx.enableAmxEvent	adf.mf.api.amx.AmxNode amxNode, Node domNode, String eventType	Allows a DOM node to trigger custom MAF AMX events such as tapHold and swipe for amx:showPopupBehavior, amx:setPropertyListener, and so on.
void	adf.mf.api.amx.doNavigation	String outcome	Tells the controller that there is an intention to perform navigation for a given outcome.
void	adf.mf.api.amx.validate	Node domNode, Function successCallback	Prevents an operation, such as navigation, when there are unsatisfied validators (required or amx:validationBehavior). The successCallback is invoked if allowed to proceed.

Table 18–1 (Cont.) Static APIs

Return Type	Function Name	Parameters	Description
`void`	`adf.mf.api.amx.showLoadingIndicator`	`Number failSafeDuration`, Function failSafeClientHandler	Shows the busy indicator. The parameters: ■ failSafeDuration: The approximate duration (non-negative integer in milliseconds) that MAF waits between showing and hiding the loading indicator (assuming some other trigger has not already shown the indicator). If this parameter is not specified or is set to `null`, then MAF uses the value of 10000 (10 seconds). ■ failSafeClientHandler: The optional JavaScript function that is invoked when the `failSafeDuration` has been reached. This function can be used to decide how to proceed. This function must return a `String` defined by one of the following values: - `hide`: to hide the indicator as in the default fail-safe. - `repeat`: to restart the timer for another duration where the function may get invoked again. - `freeze`: to keep the indicator up and wait indefinitely; the page may become stuck in a frozen state until restarted. To prevent the indicator from being displayed for longer than necessary, hide it.
`void`	`adf.mf.api.amx.hideLoadingIndicator`	None	Hides one instance of the loading indicator.
`Object`	`adf.mf.api.amx.createIterator`	`Object dataItems`	Creates an iterator that supports either a JavaScript array of objects or iterator over a tree node iterator (collection model). Returns an iterator `Object` with `next`, `hasNext`, and `isTreeNodeIterator` functions where `next` returns undefined if no more objects are available.

Table 18–1 (Cont.) Static APIs

Return Type	Function Name	Parameters	Description

`void`	`adf.mf.api.amx.bulkLoadProviders`	`Object treeNodeIterator,`
		`Number startingPoint,`
		`Number maximumNumberOfRowsToLoad,`
		`Function successCallback,`
		`Function failCallback`
			Bulk-loads a set of data providers so they are cached and are locally accessible.

`String`	`adf.mf.api.amx.buildRelativePath`	`String url`	Builds the relative path based on the specified resource assuming it is relative to the current MAF AMX page. If there is a protocol on the resource, then it is assumed to be an absolute path and left unmodified.
`void`	`adf.mf.api.amx.markNodeForUpdate`	`adf.mf.api.amx.AmxNodeUpdateArguments args`	Function for `TypeHandler` instances to notify MAF of a state change to an `AmxNode` that requires the `AmxNode` hierarchy to be updated at that node and below. If a custom `createChildrenNodes` method exists on the `TypeHandler`, it is called again for these `AmxNode` instances. This allows `AmxNode` instances that stamp their children to add new stamps due to a user change. The `refresh` method is called on the `AmxNode` with the provided properties if the `AmxNode` is ready to render. If the `AmxNode` is not ready to render, MAF waits for any EL to be resolved and the `refresh` method is called once all the data are available.

Note: Other public APIs are available in the `adf.mf.el` package for logging, translation, and data channel.

18.2.2 How to Use AmxEvent Classes

Table 18–2 lists `AMXEvent` classes that you can use when creating custom UI components.

Table 18–2 AMXEvent Classes

Class Name	Parameters	Description
`adf.mf.api.amx.ActionEvent`	None	An event triggering an outcome-based navigation. See also `oracle.adfmf.amx.event.ActionEvent` in *Java API Reference for Oracle Mobile Application Framework*.
`adf.mf.api.amx.MoveEvent`	`Object rowKeyMoved,` `Object rowKeyInsertedBefore`	An event for notifying that a specified row has been moved. It contains the key for the row that was moved along with the key for the row before which it was inserted. See also `oracle.adfmf.amx.event.MoveEvent` in *Java API Reference for Oracle Mobile Application Framework*.
`adf.mf.api.amx.SelectionEvent`	`Object oldRowKey,` `Array<Object> selectedRowKeys`	An event for changes of selection for a component. See also `oracle.adfmf.amx.event.SelectionEvent` in *Java API Reference for Oracle Mobile Application Framework*.
`adf.mf.api.amx.ValueChangeEvent`	`Object oldValue,` `Object newValue`	An event for changes of value for a component. See also `oracle.adfmf.amx.event.ValueChangeEvent` in *Java API Reference for Oracle Mobile Application Framework*.

18.2.3 How to Use the TypeHandler

Table 18–3 lists `TypeHandler` APIs that you can use to create custom UI components.

Table 18–3 TypeHandler APIs

Return Type	Function Name	Parameters	Description
`HTMLElement`	`render`	`adf.mf.api.amx.AmxNode amxNode,` `String id`	Creates an initial DOM structure and returns the root element of the structure. This member function is required and must be defined.
`void`	`init`	`HTMLElement rootElement,` `adf.mf.api.amx.AmxNode amxNode`	Represents the handler invoked after all `create` functions that belong to the set of components created with this component are invoked.
`void`	`postDisplay`	`HTMLElement rootElement,` `adf.mf.api.amx.AmxNode amxNode`	Represents the handler invoked after all `init` functions that belong to the set of components created with this component are invoked.

Table 18–3 (Cont.) TypeHandler APIs

Return Type	Function Name	Parameters	Description
`Boolean`	`createChildrenNodes`	`adf.mf.api.amx.AmxNode amxNode`	Selectively adds `AmxNode` children for processing. Note that if one of the children is shown, the use of this function prevents processing of the other children. Should return `false` if MAF is to create the children nodes instead of the custom implementation. This function is optional.
`adf.mf.api.amx.AmxNodeChangeResult`	`updateChildren`	`adf.mf.api.amx.AmxNode amxNode,` `adf.mf.api.amx.AmxAttributeChange attributeChanges`	Represents a handler for one of the following: ■ removing any old children and creating and adding any new children to the `AmxNode`. ■ through the return value, declaring what `adf.mf.api.amx.AmxNodeChangeResult` action should be taken. This function is optional.
`adf.mf.api.amx.AmxNodeChangeResult`	`getDescendentChangeAction`	`adf.mf.api.amx.AmxNode amxNode,` `adf.mf.api.amx.AmxDescendentChanges descendentChanges`	Allows a type handler to customize the handling of changes to descendent `AmxNode` instances.
`void`	`refresh`	`adf.mf.api.amx.AmxAttributeChange attributeChanges,` `adf.mf.api.amx.AmxDescendentChanges descendentChanges`	Allows a type handler to selectively refresh the HTML in response to a change. This method is called after the `updateChildren` method. The `attributeChanges` defines the changed attributes. If `descendentChanges` is not `null`, it defines the changes for any descendent nodes that need to be refreshed.
`Boolean`	`isFlattenable`	None	Declares whether or not the `AmxNode` is flattenable.Note that a flattened AmxNode might not have any behavior related to rendering: a type handler for a flattened AmxNode can only control child node creation and visiting, but cannot influence rendering.

Table 18–3 (Cont.) TypeHandler APIs

Return Type	Function Name	Parameters	Description
Boolean	visit	adf.mf.api.amx.VisitContext visitContext, Function visitCallback	Handles an AmxNode tree visitation starting from this AmxNode. The visitCallback function to invoke when visiting uses parameters visitContext and AmxNode. Returns whether or not the visitation is complete and should not continue.
Boolean	visitChildren	adf.mf.api.amx.AmxNode amxNode, adf.mf.api.amx.VisitContext visitContext, Function visitCallback	Handles an AmxNode tree visitation starting from the children of this AmxNode. The visitCallback function to invoke when visiting uses parameters visitContext and AmxNode. Returns whether or not the visitation is complete and should not continue.
void	preDestroy	HTMLElement rootElement, adf.mf.api.amx.AmxNode amxNode	Handles anything just before the current view is destroyed; when about to navigate to a new view. Typically used to save client state such as scroll positions (see adf.mf.api.amx.setClientState).
void	destroy	HTMLElement rootElement, adf.mf.api.amx.AmxNode amxNode	Handles anything after the new view is displayed and the old view is being removed.

18.2.4 How to Use the AmxNode

Table 18–4 lists AmxNode APIs that you can use to create custom UI components.

Table 18–4 AmxNode APIs

String	Function Name	Parameters	Description
String	getId	None	Gets the unique identifier for this AmxNode. This value contributes to the ID on the root DOM element.
adf.mf.api.amx.AmxTag	getTag	None	Gets the AmxTag that created this AmxNode.
adf.mf.api.amx.TypeHandler	getTypeHandler	None	Gets the TypeHandler object associated with this AmxNode.

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description
`void`	`setClientState`	`Object payloadJsonObject`	Stores or replaces the client state for the specified `AmxNode` ID. Type handlers should call this function whenever a state change happens (for example, something that should be cached so that when the user navigates to a new page and then comes back, it would be restored like a scroll position). That said, it is not always feasible to detect when a state change happens so you may need to update the state for your component just before the view is going to be discarded. There are two possible scenarios for which you need to account: 1. `refresh`: for redrawing pieces of the DOM structure (within the same view). 2. `preDestroy`: for navigating to a new view and later navigating back. The `payloadJsonObject` is the client state data to store for the lifetime of this view instance.
`Object`	`getClientState`	None	Gets the `payloadJsonObject` that was previously stored through the `setClientState` function during this view instance (undefined if not available).
`void`	`setVolatileState`	`Object payloadJsonObject`	Stores or replaces the client state for the specified `AmxNode` ID. Type handlers should call this function whenever a volatile state change happens (for example, something that should be forgotten when navigating to a new MAF AMX page but should be kept in case a component is redrawn). The `payloadJsonObject` is the volatile state data to store until navigation occurs.
`Object`	`getVolatileState`	None	Gets the `payloadJsonObject` that was previously stored through the `setVolatileState` function since the last navigation (undefined if not available).
`Object`	`getConverter`	None	Get the converter, if applicable, for this `AmxNode`.
`void`	`setConverter`	`Object converter`	Set the converter for this `AmxNode`.

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description
`String`	`storeModifyableEl`	`String nameOfTheAttribute`	For an attribute, creates and stores an EL expression that may be used to set EL values into the model. The value is context-insensitive and may be used to set a value at any time. Common use is to set a value based on user interaction. This function may be called by type handlers. Returns `null` if the subject attribute is not bound to an EL value.
`Object`	`getStampKey`	None	Gets the stamp key for the `AmxNode`. The stamp key identifies `AmxNode` instances that are produced inside of iterating containers. This is provided by the parent `AmxNode`. An example tag that uses stamp keys is the `amx:iterator` tag. Returns `null` if the `AmxNode` is not stamped.
`Array<String>`	`getDefinedAttributeNames`	None	Gets a list of the attribute names that have been defined for this node.
`Object`	`getAttribute`	`String name`	Gets an attribute value for the attribute of the given name. Return value may be `null`. Returns undefined if the attribute is not set or is not yet loaded.
`void`	`setAttributeResolvedValue`	`String name,` `Object value`	Used by the type handler or MAF to store the attribute value for an attribute onto the `AmxNode`. This function does not update the model.
`void`	`setAttribute`	`String name,` `String value`	Sets the value of an attribute on the model. This value is sent to the Java side to update the EL value. The value on the `AmxNode` is not updated by this call. Instead, it is expected that a data change event will update the `AmxNode`.
`Boolean`	`isAttributeDefined`	`String name`	Checks whether the attribute was defined by the user.
`adf.mf.api.amx.AmxNode`	`getParent`	None	Gets either the parent `AmxNode` or `null` if at the top level.

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description
void	addChild	adf.mf.api.amx.AmxNode child, String facetName	Adds a child AmxNode to this AmxNode. The facetName should be null if the child does not belong in a facet.
Boolean	removeChild	adf.mf.api.amx.AmxNode child	Removes a child AmxNode from this AmxNode. Note that the child is removed from the hierarchy, but not the DOM for it. It is up to the caller to remove the DOM. This is to allow type handlers to handle animation and other transitions when DOM is replaced. Returns whether or not the child was found and removed.
Boolean	replaceChild	adf.mf.api.amx.AmxNode oldChild, adf.mf.api.amx.AmxNode newChild	Replaces an existing child with another child. Returns whether or not the old one was found and replaced.
Array<adf.mf.api.amx.AmxNode>	getChildren	String facetName, Object stampKey	Gets children AmxNodes. The two parameters are optional. The facetName can be null to get the non-facet children. Returns an empty array if no children exist or if there are no children for the given qualifiers.
Map<String, Array<adf.mf.api.amx.AmxNode>>	getFacets	Object stampKey	Gets all of the facets of the AmxNode. The stampKey is optional; if provided, it retrieves the facet AmxNode instances for a given stamp key.
Boolean	visit	adf.mf.api.amx.VisitContext visitContext, Function visitCallback	Performs a tree visitation starting from this AmxNode. The visitCallback function should accept the visitContext and the AmxNode as arguments. Returns whether or not the visitation is complete and should not continue.
Boolean	visitChildren	adf.mf.api.amx.VisitContext visitContext, Function visitCallback	Performs a tree visitation starting from the children of this AmxNode. The visitCallback function should accept the visitContext and the AmxNode as arguments. Returns whether the visitation is complete and should not continue.

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description

`Boolean`	`visitStampedChildren`	`Object stampKey,`
		`Array<String> facetNamesToInclude,`
		`Function filterCallback,`
		`adf.mf.api.amx.VisitContext visitContext,`
		`Function visitCallback`
			Convenience function for type handlers that stamp their children to visit the children `AmxNode` from inside of a custom `visitChildren` function.
			When `facetNamesToInclude` is empty, no facets are processed for this stamp. When `facetNamesToInclude` is `null`, all facets are processed for this stamp.
			The `filterCallback` may be `null`. The `filterCallback` must return a `Boolean` of `true`, meaning the tag will be used to create children, or `false`, meaning the tag will not be processed.
			The `visitCallback` should accept the `visitContext` and `AmxNode` as arguments.
			Returns whether or not the visitation is complete and should not continue.

`Array<adf.mf.api.amx.AmxNode>`	`getRenderedChildren`	`String facetName,` `Object stampKey`	Gets the rendered children of the `AmxNode`. The `facetName` indicates from which facet to retrieve the rendered children, or `null` for the non-facet children. If the `stampKey` is provided, it retrieves the children `AmxNode` instances for a given stamp key. Returns the children that should be rendered for the given stamp key. It flattens any components that can be flattened (flattenable) and does not return any non-rendered ones.
`Boolean`	`isFlattenable`	None	Determines whether or not the `AmxNode` is flattenable. Note that a flattened AmxNode might not have any behavior related to rendering: a type handler for a flattened AmxNode can only control child node creation and visiting, but cannot influence rendering.
`adf.mf.api.amx.AmxNodeStates`	`getState`	None	Gets the current state of the `AmxNode` (as a constant value from `adf.mf.api.amx.AmxNodeStates`).

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description
`void`	`setState`	`state`	Moves the `adf.mf.api.amx.AmxNodeStates` state of the `AmxNode`. Should only be called by MAF or the `AmxNode`'s type handler.
`HTMLElement`	`render`	None	Renders the `AmxNode`. Returns the root element rendered or `null` if the child is not rendered or if there is no type handler for this `AmxNode`.
`Array<HTMLElement>`	`renderDescendants`	`String facetName,` `Object key`	Renders the subnodes of this `AmxNode` (if applicable, it flattens to the nearest descendant). If `facetName` is not `null`, it renders the children of that facet. If facetName is `null`, the non-facet children are rendered. The optional key is used for rendering the children `AmxNode` instances for that stamping key. Returns an array of the root elements for each `subNode`.
`void`	`rerender`	None	Rerenders the `AmxNode`.

Table 18–4 (Cont.) AmxNode APIs

String	Function Name	Parameters	Description
Boolean	isRendered	None	Checks the state of the AmxNode to see whether or not it should be rendered. The AmxNode is considered to be renderable if it is in the ABLE_TO_RENDER, RENDERED or PARTIALLY_RENDERED state.
void	refresh	adf.mf.api.amx.AmxAttributeChange attributeChanges, adf.mf.api.amx.AmxDescendentChanges descendentChanges	Refreshes the DOM of an AmxNode. This method is called after the updateChildren method and should be implemented by type handlers that wish to update their DOM in response to a change.
void	createStampedChildren	Object stampKey, Array<String> facetNamesToInclude, Function filterCallback	Convenience function for type handlers that stamp their children to create child AmxNode instances from inside of a custom createChildrenNodes function. This function creates children for any UI tags. If facetNamesToInclude is empty, the facets are not processed for this stamp. If facetNamesToInclude is null, all the facets are processed. If the facetNamesToInclude includes a null value inside the array, children for non-facet tags are created. The filterCallback is an optional function to filter the children that are created. The filterCallback function is invoked with the AmxNode, the stampKey, the child tag, and the facet name (or null for non-facets). The filterCallback function must return a boolean. If true, the tag is used to create children; if false, the tag is not processed.

18.2.5 How to Use the AmxTag

Table 18–5 lists AmxTag APIs that you can use to create custom UI components.

Table 18–5

Return Type	Function Name	Parameters	Description
`String`	`getNamespace`	None	Gets the XML namespace URI for the tag.
`String`	`getNsPrefixedName`	None	Returns the tag name including the namespace as its prefix (not the local `xmlns` prefix). This is the full XML name such as `"http://xmlns.example.com/custom:custom"`.
`String`	`getName`	None	Gets the tag name. This is the local XML tag name without the prefix.
`adf.mf.api.amx.AmxTag`	`getParent`	None	Gets the parent tag or `null` if it is the top-level tag.
`String`	`getTextContent`	None	Returns the text content of the tag.
`Array<adf.mf.api.amx.AmxTag>`	`findTags`	`String namespace,` `String tagName`	Recursively searches the tag hierarchy for tags with the given namespace and tag name. Returns the current tag if it matches.
`Array<adf.mf.api.amx.AmxTag>`	`getChildren`	`String namespace,` `String tagName`	Gets the children of the tag. Provides for optional filtering of the children namespaces and tag names. If a namespace is `null`, all the children are returned. If `tagName` is `null`, the children are not filtered by tag name.
`Array<adf.mf.api.amx.AmxTag>`	`getChildrenFacetTags`	None	Get all of the children facet tags. This function is meant to assist the creation of the `AmxNode` process.

Table 18–5 (Cont.)

Return Type	Function Name	Parameters	Description
`adf.mf.api.amx.AmxTag`	`getChildFacetTag`	`String name`	Gets the facet tag with the given name. This function is meant to assist the code if the presence of a facet changes the behavior of a type handler. Returns `null` if the facet is not found.
`Array<adf.mf.api.amx.AmxTag>`	`getChildrenUITags`	None	Gets all children tags that are UI tags. This function is meant to assist in creation of the `AmxNode` process. This function not return any facet tags.
`Array<String>`	`getAttributeNames`	None	Gets all of the attribute names for the attributes that are specified on the tag.
`Boolean`	`isAttributeElBound`	`String name`	Determines whether or not the given attribute is bound to an EL expression (as opposed to a static value).
`String`	`getAttribute`	`String name`	Gets the attribute value (may be an EL string) for the attribute of the given name. Returns undefined if the attribute is not specified.
`Map<String, String>`	`getAttributes`	None	Gets a key-value pair map of the attributes and their values.
`Boolean`	`isUITag`	None	Determines whether or not the node is a UI tag with a type handler and renders content.

Table 18–5 (Cont.)

Return Type	Function Name	Parameters	Description
`Object{name:string, children:Array<adf.mf.api.amx.AmxTag>}`	`getFacet`	None	Gets the tags for the children of this facet and the name of the facet if this tag is a facet tag. This is a convenience function for building the `AmxNode` tree. Returns an object with the name of the facet and the children tags of the facet. Returns `null` if the tag is not an `amx:facet` tag.
`adf.mf.api.amx.AmxNode`	`buildAmxNode`	`adf.mf.api.amx.AmxNode parentNode,` `Object stampKey`	Creates a new instance of an `AmxNode` for this tag given the stamp ID. If the tag is a facet tag, the tag creates an `AmxNode` for the child tag. This function does not initialize the `AmxNode`. Instead, it returns either an uninitialized `AmxNode` or `null` for non-UI tags.
`adf.mf.api.amx.TypeHandler`	`getTypeHandler`	None	Gets the type handler for this tag.

18.2.6 How to Use the VisitContext

Table 18–6 lists `VisitContext` APIs that you can use when creating custom UI components.

Table 18–6 VisitContext APIs

Return Type	Function Name	Parameters	Description
`Boolean`	`isVisitAll`	None	Determines whether or not all nodes should be visited.
`Array<adf.mf.api.amx.AmxNode>`	`getNodesToWalk`	None	Gets the nodes that should be walked during visitation. This list does not necessarily include the nodes that should be visited (callback invoked).
`Array<adf.mf.api.amx.AmxNode>`	`getNodesToVisit`	None	Get the list of nodes to visit.
`Array<adf.mf.api.amx.AmxNode>`	`getChildrenToWalk`	`adf.mf.api.amx.AmxNode parentAmxNode`	Determine which child AmxNode instances, including facets (if any), should be walked of the given parent `AmxNode`. Allows for type handlers to optimize how to walk the children if not all are being walked. May return `null`.

18.2.7 How to Use the AmxAttributeChange

Table 18–7 lists `AmxAttributeChange` APIs that you can use when creating custom UI components.

Table 18–7 AmxAttributeChange APIs

Return Type	Function Name	Parameters	Description
`Array<String>`	`getChangedAttributeNames`	None	Gets the names of the attributes that have been affected during the current change.
`Boolean`	`isCollectionChange`	`String name`	Determines whether the attribute change is a collection change.
`adf.mf.api.amx.AmxCollectionChange`	`getCollectionChange`	`String name`	Gets the collection model change information for an attribute. Returns null if no change object is available.
`String`	`getOldValue`	`String name`	Gets the value of the attribute before the change was made.
`Boolean`	`hasChanged`	`String name`	Determines whether the attribute with the given name has changed.
`Number`	`getSize`	None	Gets the number of attribute changes.

18.2.8 How to Use the AmxDescendentChanges

Table 18–8 lists `AmxAttributeChange` APIs that you can use when creating custom UI components.

Table 18–8 AmxDescendentChanges APIs

Return Type	Function Name	Parameters	Description
`Array<adf.mf.api.amx.AmxNode>`	`getAffectedNodes`	None	Gets the unrendered changed descendent `AmxNode` instances.
`adf.mf.api.amx.AmxAttributeChange`	`getChanges`	`adf.mf.api.amx.AmxNode amxNode`	Gets the changes for a given AmxNode.
`adf.mf.api.amx.AmxNodeStates`	`getPreviousNodeState`	`adf.mf.api.amx.AmxNode amxNode`	Gets the state of the descendent AmxNode before the changes were applied.

18.2.9 How to Use the AmxCollectionChange

Table 18–9 lists `AmxCollectionChange` APIs that you can use when creating custom UI components.

Table 18–9 AmxCollectionChange APIs

Return Type	Function Name	Parameters	Description
`Boolean`	`isItemized`	None	Determines whether or not the change to the collection may be itemized: the keys and elements on that collection were identified, so the `TypeHandler` can update just the appropriate items as opposed to rerendering the entire list from scratch.
`Array<String>`	`getCreatedKeys`	None	Gets either an array of keys that were created, or `null` if the change cannot be itemized.
`Array<String>`	`getDeletedKeys`	None	Gets either an array of the keys that were removed, or null if the change cannot be itemized.
`Array<String>`	`getUpdatedKeys`	None	Gets either an array of the keys that were updated, or null if the change cannot be itemized.
`Array<String>`	`getDirtiedKeys`	None	Gets either an array of the keys that were dirtied, or null if the change cannot be itemized.

18.2.10 How to Use the AmxNodeChangeResult

Table 18–10 lists `AmxNodeChangeResult` APIs that you can use when creating custom UI components.

Table 18–10 AmxNodeChangeResult APIs

Members	Description
`adf.mf.api.amx.AmxNodeChangeResult["NONE"]`	Takes no action in response to an attribute change on a non-rendered descendent `AmxNode`.
`adf.mf.api.amx.AmxNodeChangeResult["REFRESH"]`	The attribute and its child AmxNode instances have been updated by the type handler and the DOM will be updated by the type handler's `refresh` function.
`adf.mf.api.amx.AmxNodeChangeResult["RERENDER"]`	The AmxNode and its child AmxNode instances have been updated by the type handler, but the DOM should only be recreated as there is no need to modify the `AmxNode` hierarchy so the `refresh` function will not be called on the type handler.
`adf.mf.api.amx.AmxNodeChangeResult["REPLACE"]`	The type handler cannot handle the change. The DOM, as well as the `AmxNode` hierarchy should be recreated. This value may only be returned from the `updateChildren` method on a type handler and cannot be returned from the `getDescendentChangeAction` method.

18.2.11 How to Use the AmxNodeStates

Table 18–11 lists `AmxNodeStates` APIs that you can use when creating custom UI components.

Table 18–11 AmxNodeStates APIs

Members	Description
`adf.mf.api.amx.AmxNodeStates["INITIAL"]`	Initial state. The `AmxNode` has been created but not populated.
`adf.mf.api.amx.AmxNodeStates["WAITING_ON_EL_EVALUATION"]`	EL-based attributes needed for rendering have not been fully loaded yet.
`adf.mf.api.amx.AmxNodeStates["ABLE_TO_RENDER"]`	EL attributes have been loaded, but the `AmxNode` has not yet been rendered.
`adf.mf.api.amx.AmxNodeStates["PARTIALLY_RENDERED"]`	The EL is not fully loaded, but the `AmxNode` has partially rendered itself (reserved for future use).
`adf.mf.api.amx.AmxNodeStates["RENDERED"]`	The `AmxNode` has been fully rendered.
`adf.mf.api.amx.AmxNodeStates["UNRENDERED"]`	The AmxNode is not to be rendered.

18.2.12 How to Use the AmxNodeUpdateArguments

Table 18–12 lists `AmxNodeUpdateArguments` APIs that you can use when creating custom UI components.

Table 18–12 AmxNodeUpdateArguments APIs

Return Type	Function Name	Parameters	Description
`Array<adf.mf.api.amx.AmxNode>`	`getAffectedNodes`	None	Gets an array of affected `AmxNode` instances.
`Map<String,Boolean>`	`getAffectedAttributes`	`String amxNodeId`	Gets an object representing the affected attributes for a given AmxNode ID.
`Map<String,adf.mf.api.amx.AmxCollectionChange>`	`getCollectionChanges`	`String amxNodeId`	Gets the collection changes for a given AmxNode and property. The returned map is keyed by attribute name. Returns undefined if there are no changes for the `AmxNode`.
`void`	`setAffectedAttribute`	`adf.mf.api.amx.AmxNode amxNode,` `String attributeName`	Marks an attribute of an AmxNode as affected.

`void`	`setCollectionChanges`	`String amxNodeId,`
		`String attributeName,`
		`adf.mf.api.amx.AmxCollectionChange collectionChanges`
			Sets the collection changes for a given `AmxNode`'s attribute.

18.3 Creating Custom Components

You can create a custom UI component through the use of JavaScript and MAF APIs. This component's JavaScript file can be added to your project through the application feature-level includes. When you add your custom tag library, it is entered into the Components window's list of tag libraries and, when this library is selected, your custom component becomes available in the Components window, with its attributes displayed in the Properties window.

Before you begin:

Familiarize yourself with APIs described in Section 18.2, "Using MAF APIs to Create Custom Components."

To create a custom component:

1. Produce a JavaScript file that registers a tag namespace and series of one or more type handlers using the `adf.mf.api.amx.TypeHandler.register` API (see Table 18–1, " Static APIs" and the example later in this section).
2. For each type handler, implement a rendering member function.
3. Optionally, implement other functions.
4. Attach one or more of your JavaScript and CSS files to the MAF AMX application feature. For examples, see the following sample applications available from **File** > **New** > **MAF Examples**:
   - `custom.js` and `custom.css` files included in the MAF sample application called CompGallery.
   - WorkBetter sample application contains a custom search component.

   Alternatively, you can perform a design-time packaging.

5. For each MAF AMX page that uses one of the customs components, add an `xmlns` entry in the `view` element:

```
xmlns:custom="http://xmlns.example.com/custom"
```

This example shows a JavaScript file that declares custom components.

```
(function() {
   // TypeHandler for custom "x" elements
   var x = adf.mf.api.amx.TypeHandler.register("http://xmlns.example.com/custom",
                                               "x");
   x.prototype.render = function(amxNode) {
      var rootElement = document.createElement("div");
      rootElement.appendChild(document.createTextNode("Hello World"));
      return rootElement;
   };

   // TypeHandler for custom "y" elements
   var y = adf.mf.api.amx.TypeHandler.register("http://xmlns.example.com/custom",
                                               "y");

   y.prototype.render = function(amxNode) {
      var rootElement = document.createElement("div");
      rootElement.appendChild(document.createTextNode("Goodbye World"));
      return rootElement;
   };

})();
```

For examples of how to create custom UI components, see the `custom.amx`, `customOther.amx`, `exampleEvents.amx`, and `exampleList.amx` files included in the MAF sample application called CompGallery. This sample application is available from **File** > **New** > **MAF Examples**.

19

Using the Local Database in MAF AMX

This chapter describes how to use the local SQLite database with a MAF application.

This chapter includes the following sections:

- Section 19.1, "Introduction to the Local SQLite Database Usage"
- Section 19.2, "Using the Local SQLite Database"

19.1 Introduction to the Local SQLite Database Usage

SQLite is a relational database management system (RDBMS) specifically designed for embedded applications.

SQLite has the following characteristics:

- It is ACID-compliant: like other traditional database systems, it has the properties of atomicity, consistency, isolation, and durability.
- It is lightweight: the entire database consists of a small C library designed to be embedded directly within an application.
- It is portable: the database is self-contained in a single file that is binary-compatible across a diverse range of computer architectures and operating systems

For more information, see the SQLite website at `http://www.sqlite.org`.

For a sample usage of the local SQLite database, see the MAF example application called StockTracker, which you can open from **File** > **New** > **MAF Examples**. For more information, see Section 19.2.8, "What You May Need to Know About the StockTracker Sample Application."

19.1.1 Differences Between SQLite and Other Relational Databases

SQLite is designed for use as an embedded database system, one typically used by a single user and often linked directly into the application. Enterprise databases, on the other hand, are designed for high concurrency in a distributed client-server environment. Because of these differences, there is a number of limitations compared to Oracle databases. Some of the most important differences are:

- Concurrency
- SQL Support and Interpretation
- Data Types
- Foreign Keys

- Database Transactions
- Authentication

For more information, see the following:

- Documentation section of the SQLite website at http://www.sqlite.org/docs.html
- "Limits In SQLite" available from the Documentation section of the SQLite website at http://www.sqlite.org/limits.html

19.1.1.1 Concurrency

At any given time, a single instance of the SQLite database may have either a single read-write connection or multiple read-only connections.

Due to its coarse-grained locking mechanism, SQLite does not support multiple read-write connections to the same database instance. For more information, see "File Locking And Concurrency In SQLite Version 3" available from the Documentation section of the SQLite website at http://www.sqlite.org/lockingv3.html.

19.1.1.2 SQL Support and Interpretation

Although SQLite complies with the SQL92 standard, there are a few unsupported constructs, including the following:

- `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`
- `GRANT`
- `REVOKE`

For more information, see "SQL Features That SQLite Does Not Implement" available from the Documentation section of the SQLite website at http://www.sqlite.org/omitted.html.

For information on how SQLite interprets SQL, see "SQL As Understood by SQLite" available from the Documentation section of the SQLite website at http://www.sqlite.org/lang_createtable.html.

19.1.1.3 Data Types

While most database systems are strongly typed, SQLite is dynamically typed and therefore any value can be stored in any column, regardless of its declared type. SQLite does not return an error if, for instance, a string value is mistakenly stored in a numeric column. For more information, see "Datatypes In SQLite Version 3" available from the Documentation section of the SQLite website at http://www.sqlite.org/datatype3.html.

19.1.1.4 Foreign Keys

SQLite supports foreign keys. It parses and enforces foreign key constraints. For more information, see the *SQLite Foreign Key Support* available from the Documentation section of the SQLite site at http://www.sqlite.org/foreignkeys.html.

19.1.1.5 Database Transactions

Although SQLite is ACID-compliant and hence supports transactions, there are some fundamental differences between its transaction support and Oracle's:

- Nested transactions: SQLite does not support nested transactions. Only a single transaction may be active at any given time.
- Commit: SQLite permits either multiple read-only connections **or** a single read-write connection to any given database. Therefore, if you have multiple connections to the same database, only the first connection that attempts to modify the database can succeed.
- Rollback: SQLite does not permit a transaction to be rolled back until all open `ResultSets` have been closed first.

For more information, see "Distinctive Features of SQLite" available from the Documentation section of the SQLite website at `http://www.sqlite.org/different.html`.

19.1.1.6 Authentication

SQLite does not support any form of role-based or user-based authentication. By default, anyone can access all of the data in the file. However, MAF provides encryption routines that you can use to secure the data and prevent access by users without the valid set of credentials. For more information, see Section 19.2.7, "How to Encrypt and Decrypt the Database."

19.2 Using the Local SQLite Database

MAF contains an encrypted SQLite 3.7.9 database.

A typical SQLite usage requires you to know the following:

- How to Connect to the Database
- How to Use SQL Script to Initialize the Database or How to Initialize the Database on a Desktop
- How to Encrypt and Decrypt the Database
- How to Use the VACUUM Command

19.2.1 How to Connect to the Database

Connecting to the SQLite database is somewhat different from opening a connection to an Oracle database. That said, once you have acquired the initial connection, you can use most of the same JDBC APIs and SQL syntax to query and modify the database.

You use the `java.sql.Connection` object associated with your application to connect to the SQLite database. When creating the connection, ensure that every SQLite JDBC URL begins with the text `jdbc:sqlite:`.

This example shows how to open a connection to an unencrypted database.

```
java.sql.Connection connection = new SQLite.JDBCDataSource
             ("jdbc:sqlite:/path/to/database").getConnection();
```

This example shows how to open a connection to an encrypted database.

```
java.sql.Connection connection = new SQLite.JDBCDataSource
             ("jdbc:sqlite:/path/to/database").getConnection(null,"password");
```

In the preceding example, the first parameter of the `getConnection` method is the user name, but since SQLite does not support user-based security, this value is ignored.

Note: SQLite does not display any error messages if you open an encrypted database with an incorrect password. Likewise, you are not alerted if you mistakenly open an unencrypted database with a password. Instead, when you attempt to read or modify the data, an `SQLException` is thrown with the message "Error: file is encrypted or is not a database".

19.2.2 How to Use SQL Script to Initialize the Database

Typically, you can use an SQL script to initialize the database when the application starts. This example shows the SQL initialization script that demonstrates some of the supported SQL syntax (described in Section 19.1.1.2, "SQL Support and Interpretation") through its use of the `DROP TABLE`, `CREATE TABLE`, and `INSERT` commands and the `NUMBER` and `VARCHAR2` data types.

```
DROP TABLE IF EXISTS PERSONS;

CREATE TABLE PERSONS
(
PERSON_ID NUMBER(15) NOT NULL,
FIRST_NAME VARCHAR2(30),
LAST_NAME VARCHAR2(30),
EMAIL VARCHAR2(25) NOT NULL
);

INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 100,
'David', 'King', 'steven@king.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 101,
'Neena', 'Kochhar', 'neena@kochhar.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 102, 'Lex',
'De Haan', 'lex@dehaan.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 103,
'Alexander', 'Hunold', 'alexander@hunold.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 104,
'Bruce', 'Ernst', 'bruce@ernst.net');
```

To use the SQL script, add it to the application project of your MAF application as a resource. Suppose a sample script has been saved as `initialize.sql` in the `META-INF` directory. The next example shows the code that you should add to parse the SQL script and execute the statements.

```
private static void initializeDatabaseFromScript() throws Exception {
   InputStream scriptStream = null;
   Connection conn = null;
   try {
      // ApplicationDirectory returns the private read-write sandbox area
      // of the mobile device's file system that this application can access.
      // This is where the database is created
      String docRoot = AdfmfJavaUtilities.getDirectoryPathRoot
                               (AdfmfJavaUtilities.ApplicationDirectory);
      String dbName = docRoot + "/sample.db";

      // Verify whether or not the database exists.
      // If it does, then it has already been initialized
      // and no furher actions are required
      File dbFile = new File(dbName);
      if (dbFile.exists())
         return;
```

```
      // If the database does not exist, a new database is automatically
      // created when the SQLite JDBC connection is created
      conn = new SQLite.JDBCDataSource("jdbc:sqlite:" + docRoot +
                                       "/sample.db").getConnection();

      // To improve performance, the statements are executed
      // one at a time in the context of a single transaction
      conn.setAutoCommit(false);

      // Since the SQL script has been packaged as a resource within
      // the application, the getResourceAsStream method is used
      scriptStream = Thread.currentThread().getContextClassLoader().
                           getResourceAsStream("META-INF/initialize.sql");
      BufferedReader scriptReader = new BufferedReader
                                        (new InputStreamReader(scriptStream));
      String nextLine;
      StringBuffer nextStatement = new StringBuffer();

      // The while loop iterates over all the lines in the SQL script,
      // assembling them into valid SQL statements and executing them as
      // a terminating semicolon is encountered
      Statement stmt = conn.createStatement();
      while ((nextLine = scriptReader.readLine()) != null) {
         // Skipping blank lines, comments, and COMMIT statements
         if (nextLine.startsWith("REM") ||
             nextLine.startsWith("COMMIT") ||
             nextLine.length() < 1)
            continue;
         nextStatement.append(nextLine);
         if (nextLine.endsWith(";")) {
            stmt.execute(nextStatement.toString());
            nextStatement = new StringBuffer();
         }
      }
      conn.commit();
   }
   finally {
      if (conn != null)
         conn.close();
   }
}
```

> **Note:** In the example above, the error handling was omitted for simplicity.

You invoke the database initialization code (see the previous example) from the `start` method of the `LifeCycleListenerImpl`, as the next example shows.

```
public void start() {
   try {
      initializeDatabaseFromScript();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
```

```
                e);
      }
   }
```

19.2.3 How to Initialize the Database on a Desktop

Because SQLite databases are self-contained and binary-compatible across platforms, you can use the same database file on iOS, Android, Windows, Linux, and Mac OS platforms. In complex cases, you can initialize the database on a desktop using third-party tools (such as MesaSQLite, SQLiteManager, and SQLite Database Browser), and then package the resulting file as a resource in your application.

To use the database, add it to the application project of your MAF application as a resource. Suppose a database has been saved as `sample.db` in the `META-INF` directory. The example below shows the code that you should add to copy the database from your application to the mobile device's file system to enable access to the database.

This example shows how to initialize the database on a desktop.

```
private static void initializeDatabase() throws Exception {
   InputStream sourceStream = null;
   FileOutputStream destinationStream = null;
   try {
      // ApplicationDirectory returns the private read-write sandbox area
      // of the mobile device's file system that this application can access.
      // This is where the database is created
      String docRoot = AdfmfJavaUtilities.getDirectoryPathRoot
                               (AdfmfJavaUtilities.ApplicationDirectory);
      String dbName = docRoot + "/sample.db";

      // Verify whether or not the database exists.
      // If it does, then it has already been initialized
      // and no furher actions are required
      File dbFile = new File(dbName);
      if (dbFile.exists())
         return;

      // Since the database has been packaged as a resource within
      // the application, the getResourceAsStream method is used
      sourceStream = Thread.currentThread().getContextClassLoader().
                           getResourceAsStream("META-INF/sample.db");
      destinationStream = new FileOutputStream(dbName);
      byte[] buffer = new byte[1000];
      int bytesRead;
      while ((bytesRead = sourceStream.read(buffer)) != -1) {
         destinationStream.write(buffer, 0, bytesRead);
      }
   }
   finally {
      if (sourceStream != null)
         sourceStream.close();
      if (destinationStream != null)
         destinationStream.close();
   }
}
```

Note: In the previous example, the error handling was omitted for simplicity.

You invoke the database initialization code (see the previous example) from the `start` method of the `LifeCycleListenerImpl`, as the next example shows.

```
public void start() {
   try {
      initializeDatabase();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
                e);
   }
}
```

19.2.4 What You May Need to Know About Commit Handling

`Commit` statements are ignored when encountered. Each statement is committed as it is read from the SQL script. This auto-commit functionality is provided by the SQLite database by default. To improve your application's performance, you can disable the auto-commit to allow a regular execution of `commit` statements by using the `Connection`'s `setAutoCommit(false)` method.

19.2.5 Limitations of the MAF's SQLite JDBC Driver

The following methods from the `java.sql` package have limited or no support in MAF:

- The `getByte` method of the `ResultSet` is not supported. If used, this method will throw an `SQLException` when executed.
- The `execute` method of the `Statement` always returns `true` (as opposed to returning `true` only for statements that return a `ResultSet`).

19.2.6 How to Use the VACUUM Command

When records are deleted from an SQLite database, its size does not change. This leads to fragmentation and, ultimately, results in degraded performance. You can avoid this by periodically running the `VACUUM` command.

> **Note:** The `VACUUM` can take a significant amount of time when run on large databases (approximately 0.5 seconds per megabyte on the Linux computer on which SQLite is developed). In addition, it can use up to twice as much temporary disk space as the original file while it is running.

Typically, the `VACUUM` command should be run from a properly registered `LifeCycleListener` implementation (see .

19.2.7 How to Encrypt and Decrypt the Database

MAF allows you to provide the SQLite database with an initial or subsequent encryption through the use of various APIs. Some of these APIs enable you to specify your own password for encrypting the database. Others are used when you prefer MAF to generate and, optionally, manage the password.

To encrypt the database with your own password:

1. Establish the database connection (see Section 19.2.1, "How to Connect to the Database").
2. Use the following utility method to encrypt the database with a new key:

```
AdfmfJavaUtilities.encryptDatabase(connection, "newPassword");
```

To permanently decrypt the database encrypted with your own password:

1. Open the encrypted database with the correct password.
2. Use the following utility method:

```
AdfmfJavaUtilities.decryptDatabase(connection);
```

> **Caution:** If you open a database incorrectly (for example, use an invalid password to open an encrypted database), and then encrypt it again, neither the old correct password, the invalid password, nor the new password can unlock the database resulting in the irretrievable loss of data.

To encrypt the database using the MAF-generated password:

1. Generate a password using the following method:

```
GeneratedPassword.setPassword("databasePasswordID", "initialSeedValue");
```

   This method requires both a unique identifier and an initial seed value to aid the cryptographic functions in generating a strong password.
2. Retrieve the created password using the previously-specified ID as follows:

```
char[] password = GeneratedPassword.getPassword("databasePasswordID");
```

3. Establish the database connection (see Section 19.2.1, "How to Connect to the Database").
4. Encrypt the database as follows:

```
AdfmfJavaUtilities.encryptDatabase(connection, new String(password));
```

To decrypt the database and delete the MAF-generated password:

1. Obtain the correct password as follows:

```
char[] password = GeneratedPassword.getPassword("databasePasswordID");
```

2. Establish the database connection and decrypt the database as follows:

```
java.sql.Connection connection =
                    SQLite.JDBCDataSource("jdbc:sqlite:/path/to/database").
                    getConnection(null, new String(password));
```

3. Optionally, delete the generated password using the following method:

```
GeneratedPassword.clearPassword("databasePasswordID");
```

19.2.8 What You May Need to Know About the StockTracker Sample Application

The StockTracker sample application uses a custom SQLite database file that is packaged within this application. The database file contains a table with four records which include information on four stocks. When the application is activated, it reads data from the table and displays the four stocks. The information about the stocks can be subject to CRUD operations: the stocks can be created, reordered, updated, and deleted through the user interface. All the CRUD operations, including reordering of stocks, are updated in the SQLite database.

20

Implementing Application Feature Content Using Remote URLs

This chapter describes how application features with content from remote URLs can access (or be restricted from) device services. It also describes how to implement a whitelist in a MAF plus enable a navigation bar on remote URL pages that render in the MAF application's web view.

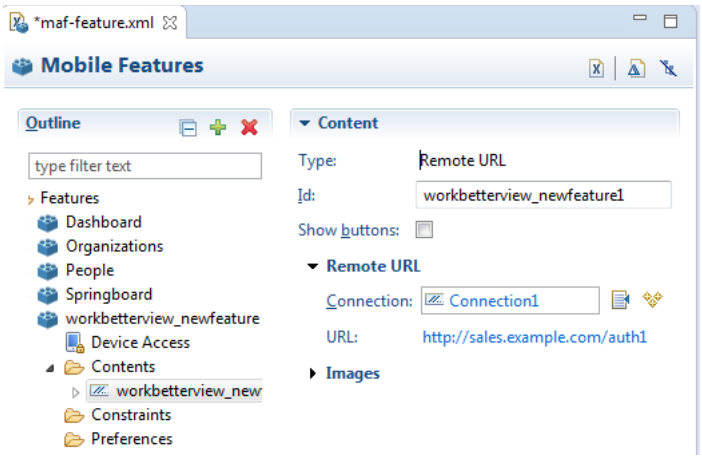
This chapter includes the following sections:

- Section 20.1, "Introduction to Remote URL Applications"
- Section 20.2, "Enabling Remote Applications Access Container Services"
- Section 20.3, "Whitelisting Remote URLs in Your MAF Application"
- Section 20.4, "Enabling the Browser Navigation Bar on Remote URL Pages"

20.1 Introduction to Remote URL Applications

By configuring the content type for an application feature as Remote URL in the MAF Features Editor, as shown in Figure 20–1, you create a browser-based application that is served from the specified URL. Such server-hosted applications differ from client applications written in MAF AMX, local HTML, or a platform-specific language such as Objective-C in that they are intended for occasional use and cannot directly access the device's memory or services (such as the camera, contacts, or GPS). These interactions are instead contingent upon the capabilities of the device's browser. For details about configuring Remote URL content, see Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML."

Figure 20–1 Configuring Remote URL Content



20.2 Enabling Remote Applications Access Container Services

Remote URL applications that open within the MAF web view use Apache Cordova JavaScript APIs to access device features and MAF JavaScript APIs to access the MAF container services. You use a JavaScript `<script>` tag that references the `base.js` libraries to enable this access.

To access MAF or Cordova JavaScript APIs from within a server-rendered web application (for example, getting and setting EL expressions, getting information about the application, taking a photo, or accessing contacts), you must use the virtual path `/~maf.device~/` when including `base.js` so that the browser will identify the request as being for a MAF resource and not for the remote server. This approach works in both remote as well as local HTML pages and is the best way to include `base.js` in an HTML feature (regardless of where it is being served from).The following example shows how to include `base.js` from a device HTML page or from a remote HTML page:

```
<html>
  <head>
    <script src="/~maf.device~/www/js/base.js"></script>
...
```

When the container code reads `/~maf.device~/` in the requested URL it then resolves the URL locally, as shown in the following example, and treats it as a native request.

```
<script src="http://your.domain.ip/~maf.device~/www/js/base.js"></script>
```

where `your.domain.ip` is the domain from the remote URL.

MAF then reads the file from the file system in the container code and sends the local file content to the web view.

In addition to using the virtual path `/~maf.device~/`, as already described, verify that the Allow Native Access property (`allowNativeAccess`) is set to the default value of `true` in the `maf-application.xml` file for the application feature that specifies the remote URL application as its content type. The following example shows this property in a `maf-application.xml` source file:

```
<adfmf:featureReference refId="remoteAppfeature1" id="fr1"
allowNativeAccess="true"/>
```

If this property is false, the remote URL application feature cannot access the container services.

20.3 Whitelisting Remote URLs in Your MAF Application

Use a whitelist if you allow a remote URL application feature to access container services and you want to restrict the list of URLs that can access the services.

Performing this task is commonly known as *whitelisting*. For example, Figure 20–2 shows a remote URL application that renders `http://www.oracle.com`. Assume that the **Watch the Webcast on demand** link in Figure 20–2 navigates to a non-oracle.com URL. This is not desirable as you do not know and cannot trust what actions the non-oracle URL may perform if you have granted access to container services. Implementing a whitelist in your MAF application can restrict access to oracle.com URLs and prevent untrusted URLs from accessing container services if you have granted a remote URL application feature access to container services.

Use a Cordova plugin if you want to implement a whitelist in your MAF application. For an overview of how Cordova plugins implement whitelists, see the "Whitelist Guide" in the Apache Cordova documentation at `https://cordova.apache.org/docs/en/latest/guide/appdev/whitelist/index.html`.

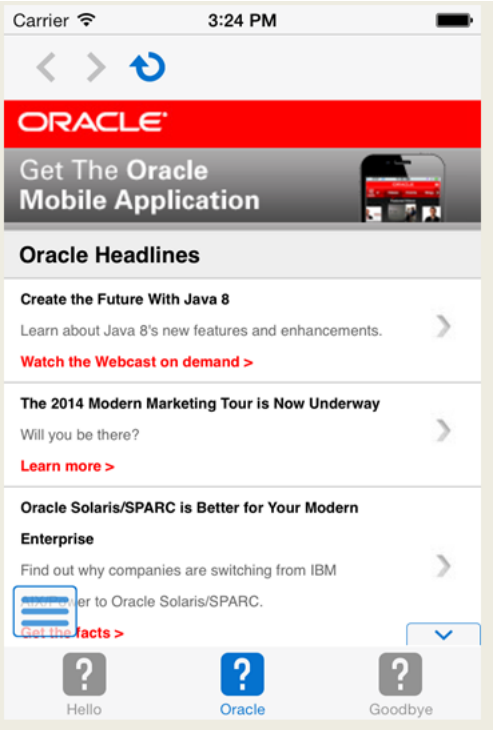
Apache Cordova provides a plugin (`cordova-plugin-whitelist`) that you can configure to implement a whitelist in a MAF application that you deploy to Android. You download this plugin, modify it, configure it with whitelist entries, and register it in your MAF application. For iOS and UWP, you configure whitelist entries in the `plugin.xml` of a plugin that you develop or modify.

> **Tip:** Configure the `plugin.xml` file in Apache Cordova plugin's for Android (`cordova-plugin-whitelist`) with whitelist entries for each of Android, iOS and UWP. With this approach, you register one plugin in your MAF application that implements whitelists for all platforms to which you deploy your MAF application. The following sections provide sample `plugin.xml` entries that demonstrate how you can implement a whitelist for each platform.

For specific information, see:

- Android platform, see Section 20.3.1, "How to Whitelist Remote URLs on the Android Platform"
- iOS platform, see Section 20.3.2, "How to Whitelist Remote URLs on the iOS Platform"
- Universal Windows Platform, see Section 20.3.3, "How to Whitelist Remote URLs on Universal Windows Platform"

Figure 20–2 Remote URL in MAF Web View



20.3.1 How to Whitelist Remote URLs on the Android Platform

Implementing a whitelist in a MAF application that you deploy to Android can be accomplished by writing a Cordova plugin.

Alternatively, download the `cordova-plugin-whitelist` plugin and modify it as follows:

- Open the file `src/android/WhitelistPlugin.java` in a text editor
- In each of the following methods, change `"return null;"` to `"return false;"`:
  - `shouldAllowNavigation(String url)`
  - `shouldAllowRequest(String url)`
  - `shouldOpenExternalUrl(String url)`
- Save the modified file

Configure the whitelist entries in the `plugin.xml` file, and register it in your MAF application. For more information about `cordova-plugin-whitelist`, see the Apache Cordova documentation at `https://cordova.apache.org/docs/en/latest/cordova-plugin-whitelist/`. The latter documentation provides examples that demonstrate how you can implement navigation and network request whitelists plus implement content security policies using the plugin.

The following sample illustrates how the plugin.xml file could be configured to whitelist HTTP and HTTPS URLs from the oracle.com domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
id="cordova-plugin-whitelist" version="1.2.2">
```

```
    <name>Whitelist</name>
    <description>Cordova Network Whitelist Plugin</description>
    <license>Apache 2.0</license>
    <keywords>cordova,whitelist,policy</keywords>

    <engines>
      <engine name="cordova-android" version=">=4.0.0" />
    </engines>

    <platform name="android">
        <config-file target="res/xml/config.xml" parent="/*">
            <allow-navigation href="http://*.oracle.com/*"/>
            <allow-navigation href="https://*.oracle.com/*"/>
            <feature name="Whitelist" >
                <param name="android-package"
value="org.apache.cordova.whitelist.WhitelistPlugin"/>
                <param name="onload" value="true" />
            </feature>
        </config-file>

        <source-file src="src/android/WhitelistPlugin.java"
target-dir="src/org/apache/cordova/whitelist" />

        <info>
           This plugin is only applicable for versions of cordova-android greater
than 4.0. If you have a previous
           platform version, you do *not* need this plugin since the whitelist
will be built in.
        </info>
    </platform>
</plugin>
```

Once you complete implementing the whitelist that you want, add the plugin to your MAF application, as described in Section 9.3, "Registering Additional Plugins in Your MAF Application."

For more information about developing a plugin for use in a MAF application, see Section 9.1, "Introduction to Using Plugins in MAF Applications."

20.3.2 How to Whitelist Remote URLs on the iOS Platform

Implementing a whitelist in a MAF application that you deploy to iOS can be accomplished by writing a Cordova plugin.

Develop and/or register a Cordova plugin in your MAF application that enables whitelisting. The following sample illustrates a plugin that can be used to whitelist HTTP and HTTPS URLs from the `oracle.com` domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
id="maf-ios-whitelist-plugin" version="1.0.0">

    <name>Whitelisting</name>
    <description>Plugin to white list remote URLs in a MAF app on
iOS.</description>

    <platform name="ios">
        <config-file target="config.xml" parent="/*">
            <allow-navigation href="http://*.oracle.com/*"/>
            <allow-navigation href="https://*.oracle.com/*"/>
```

```
            <feature name="CDVIntentAndNavigationFilter">
                <param name="ios-package" value="CDVIntentAndNavigationFilter"/>
                <param name="onload" value="true"/>
            </feature>
        </config-file>
    </platform>
</plugin>
```

Whitelisting works for hyperlinks and top level URIs. Whitelisting does not work for resource loading, includes, XMLHttpRequests, and URIs used in HTML `src` attributes.

For more information about developing a plugin for use in a MAF application, see Section 9.1, "Introduction to Using Plugins in MAF Applications." Once you complete development of your plugin, register it in your MAF application, as described in Section 9.3, "Registering Additional Plugins in Your MAF Application."

20.3.3 How to Whitelist Remote URLs on Universal Windows Platform

Implementing a whitelist in a MAF application that you deploy to UWP can be accomplished by writing a Cordova plugin.

Develop and/or register a Cordova plugin in your MAF application that enables whitelisting.

See the following annotated sample for a number of examples that demonstrate how you can implement whitelist(s) in a plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns=http://apache.org/cordova/ns/plugins/1.0
xmlns:uap=http://schemas.microsoft.com/appx/manifest/uap/windows10

id="maf-cordova-plugin-windows-whitelist" version="1.0.0">

    <name>MAF Windows Whitelist</name>
    <description>MAF Windows Plugin to whitelist URLs that have Window RT
access</description>
    <keywords>cordova,whitelist</keywords>

    <!-- windows 10 -->
    <platform name="windows">

        <!-- In a MAF application, local package URLs have access to all Windows
Runtime(RT) API. Access to Windows RT API is essential
             for Cordova plugins that need to make device calls. External website
URLs like: http:// and https:// do NOT have access to
             Windows RT API by default in MAF. This plugin allows modifications to
the application manifest so that Windows RT API
             access and hence Cordova device access can be granted to external
URLs.
        -->

        <config-file target="package.appxmanifest"
parent="/Package/Applications/Application/uap:ApplicationContentUriRules">

            <!-- The following example provides access to the Window RT API to a
single page -->
            <uap:Rule Match="http://example.com/crmapp/contacts/contact.html"
Type="include" WindowsRuntimeAccess="all" />

            <!-- The following example denies access to Window RT API to specific
directories inside the package -->
```

```
            <uap:Rule Match="ms-appx-web:///FARs/ViewController/public_
html/noNativeAcessFeature/*" Type="include" WindowsRuntimeAccess="none" />

            <!– The following example allows everything in the oracle.com domain
to access the Windows RT API in the application-->
            <uap:Rule Match="http://*.oracle.com/*" Type="include"
WindowsRuntimeAccess="all" />

        </config-file>
    </platform>
</plugin>
```

Once you complete implementing the whitelist that you want, add the plugin to your MAF application, as described in Section 9.3, "Registering Additional Plugins in Your MAF Application."

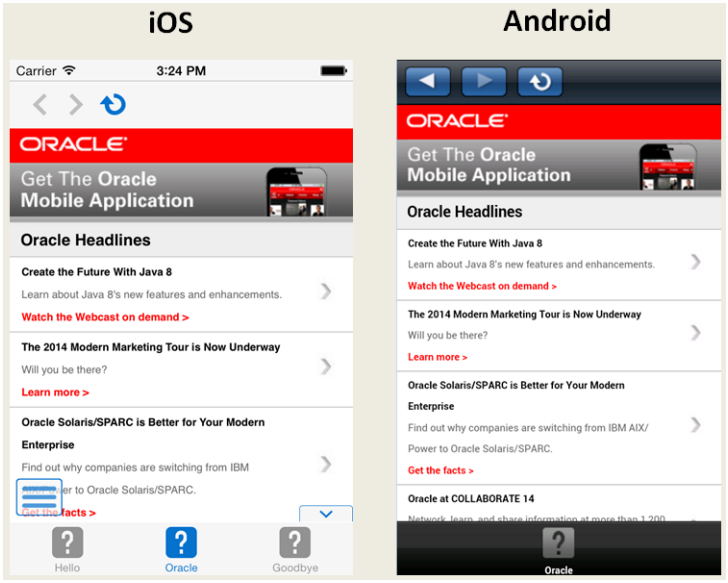
For more information about developing a plugin for use in a MAF application, see Section 9.1, "Introduction to Using Plugins in MAF Applications."

20.4 Enabling the Browser Navigation Bar on Remote URL Pages

MAF enables you to add a navigation bar with buttons for back, forward, and refresh actions for application features implemented as remotely served web content that open within the MAF web view, as shown in Figure 20–3. The forward and back buttons are disabled when either navigation forward or back is not possible.

Note: The back button is disabled on Android-powered devices.

Figure 20–3 A Remote Web Page Displaying the Navigation and Refresh Buttons



20.4.1 How to Add the Navigation Bar to a Remote URL Application Feature

You enable users to navigate through, or refresh remote content through the Content tab of the MAF Feature Editor.

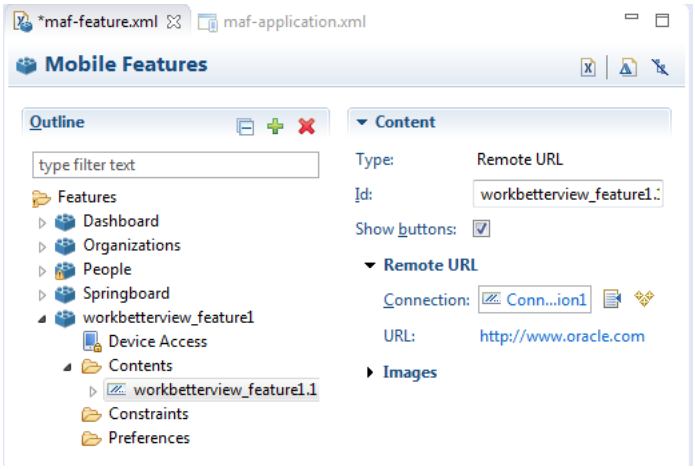
Before you begin:

Designate an application feature's content be delivered from a remotely hosted application by first selecting **Remote URL** and then by creating the connection to the host server, as described in Section 5.2, "Defining the Application Feature Content as Remote URL or Local HTML."

To enable a navigation bar:

1. Select the Remote URL application feature listed under **Features** in the MAF Feature Editor outline.
2. Expand **Content** and select the Remote URL content.
3. In the Content section, select **Show buttons**, as shown in Figure 20–4.

Figure 20–4 Selecting Navigation Options



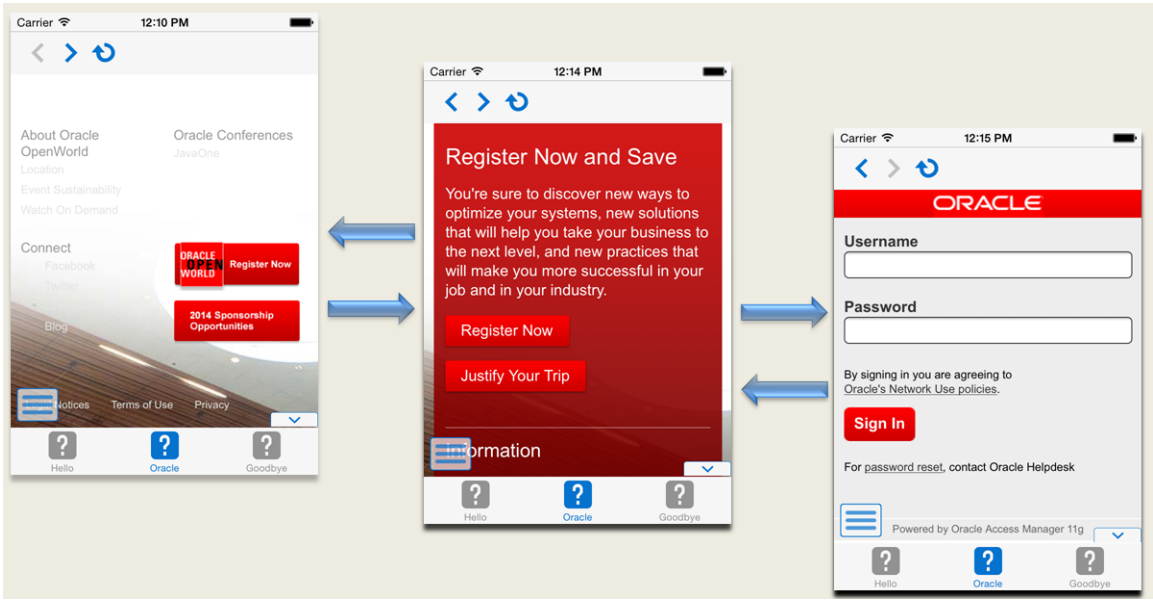
20.4.2 What Happens When You Enable the Browser Navigation Buttons for a Remote URL Application Feature

OEPE updates the `adfmf:remoteURL` element with an attribute called `showNavButtons`, which is set to `true`, as shown in the example below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
  <adfmf:feature id="oraclemobile" name="oraclemobile">
    <adfmf:content id="oraclemobile.1">
      <adfmf:remoteURL connection="connection1"
                       showNavButtons="true"/>
    </adfmf:content>
  </adfmf:feature>
</adfmf:features>
```

After you deploy the application, MAF applies the forward, back, and refresh buttons to the web pages that are traversed from the home page of the Remote URL application feature, as shown in Figure 20–5.

Figure 20–5 Traversing Through a Remote URL Application Feature



21

Enabling User Preferences

This chapter describes how to create both application-level and application feature-level user preference pages.

This chapter includes the following sections:

- Section 21.1, "Creating User Preference Pages for a Mobile Application"
- Section 21.2, "Creating User Preference Pages for Application Features"
- Section 21.3, "Using EL Expressions to Retrieve Stored Values for User Preference Pages"
- Section 21.4, "Platform-Dependent Display Differences"

21.1 Creating User Preference Pages for a Mobile Application

Preferences enable you to add settings that can be configured by end users. Within both the `maf-application.xml` and `maf-feature.xml` files, the user preference pages are defined with the `<adfmf:preferences>` element. You also use the `<adfmf:preferences>` element to create the preferences that users manage within each application feature.

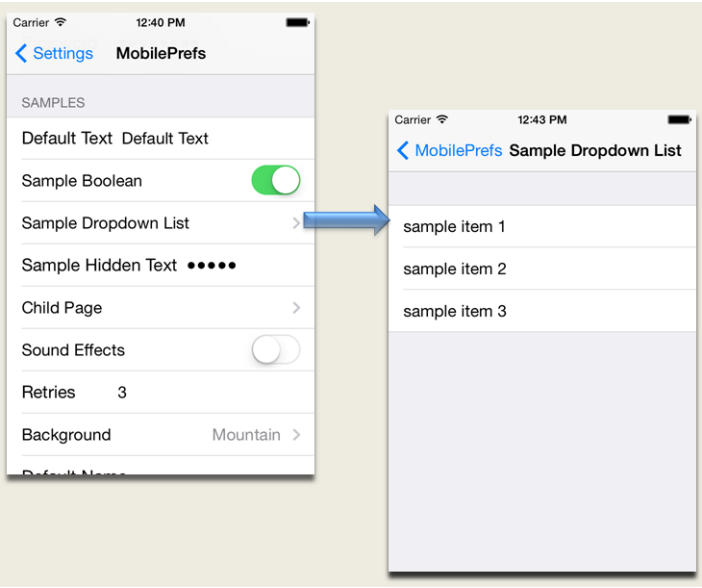
As shown in the following example, the child element of `<adfmf:preferences>` called `<adfmf:preferenceGroup>` and its child elements define the user preferences by creating pages that present options in various forms, such as text strings, dropdown menus, or in this case, as a child page that can present the user with additional options for application settings.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                   name="MobileApplication"
                   id="com.company.MobileApplication"
                   appControllerFolder="ApplicationController"
                   version="1"
                   vendor="oracle"
                   listener-class="application.LifeCycleListenerImpl">
  <adfmf:description>This app created by Mobile Application
Framework</adfmf:description>
  <adfmf:featureReference id="PROD"/>
  <adfmf:featureReference id="HCM"/>
  <adfmf:featureReference id="Customers"/>
      <adfmf:preferences>
<adfmf:preferenceGroup id="a" label="Prefs Group A">
        <adfmf:preferenceBoolean id="a1_sound" label="Sound Effects"/>
        <adfmf:preferenceNumber id="a2_retries" label="Retries" default="3"/>
```

```
        <adfmf:preferenceList id="a3_background" label="Background" default="3">
          <adfmf:preferenceValue name="None" value="0" id="pv4"/>
          <adfmf:preferenceValue name="Field" value="1" id="pv1"/>
          <adfmf:preferenceValue name="Galaxy" value="2" id="pv5"/>
          <adfmf:preferenceValue name="Mountain" value="3" id="pv6"/>
        </adfmf:preferenceList>
        <adfmf:preferenceText id="a4_name" label="Default Name"/>
        <adfmf:preferencePage id="aa" label="Prefs SubGroup AA">
          <adfmf:preferenceGroup id="aa_sec" label="Security">
            <adfmf:preferenceBoolean id="aa_sec_useSec" label="Use Security"/>
            <adfmf:preferenceNumber id="aa_sec_timeout" label="Timeout (secs)"
default="120"/>
          </adfmf:preferenceGroup>
        </adfmf:preferencePage>
      </adfmf:preferenceGroup>
      <adfmf:preferenceGroup id="b" label="Prefs Group B">
        <adfmf:preferenceBoolean id="b_cloudSync" label="Cloud Sync"/>
        <adfmf:preferenceList id="b_dispUsage" label="Display Usage As"
default="1">
          <adfmf:preferenceValue name="Percent" value="1" id="pv2"/>
          <adfmf:preferenceValue name="Minutes" value="2" id="pv3"/>
        </adfmf:preferenceList>
      </adfmf:preferenceGroup>
    </adfmf:preferences>
</adfmf:application>
```

Figure 21–1 shows an example of how opening child user preferences page can offer subsequent options.

Figure 21–1 User Preferences Pages



Preference pages are defined within the `<adfmf:preferenceGroup>` element and have the following child elements:

- `<adfmf:preferencePage>`—Specifies a new page in the user interface.
- `<adfmf:preferenceList>`—Provides users with a specific set of options.
  - `<adfmf:preferenceValue>`—A child element that defines a list element.

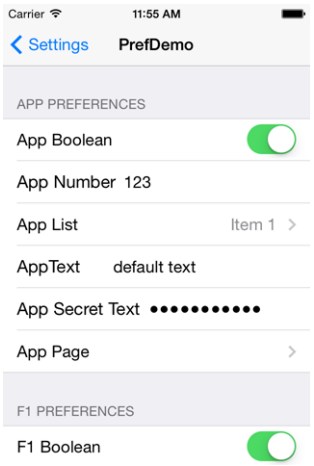
- `<adfmf:preferenceBoolean>`—A boolean setting.
- `<adfmf:preferenceText>`—A text preference setting.

See *Tag Reference for Oracle Mobile Application Framework* for more information on these elements and their attributes.

For an example of creating preference pages at both the application and application-feature levels, refer to the PrefDemo sample application. This sample application is available from **File** > **New** > **MAF Examples**.

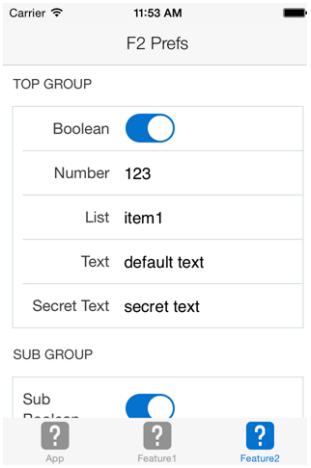
The PrefDemo application is comprised of an application-level settings page as well as three application feature preference pages, which are implemented as MAF AMX. Figure 21–2 shows the PrefDemo application settings page, which you invoke from the general settings page. In this illustration, the preference settings page is invoked from the iOS Settings application.

Figure 21–2 The PrefDemo Application Settings Page



The application feature preference pages, illustrated by *App*, *Feature1* (which is selected), and *Feature 2* in Figure 21–3, provide examples of preferences pages constructed from the MAF AMX Boolean Switch, Input Text, and Output Text components that use EL (Expression Language) to access the application feature and the various `<adfmf:preferences>` components configured within it. For more information, see Section 21.3, "Using EL Expressions to Retrieve Stored Values for User Preference Pages."

Figure 21–3 An Application Feature Preference Page from the PrefDemo Application

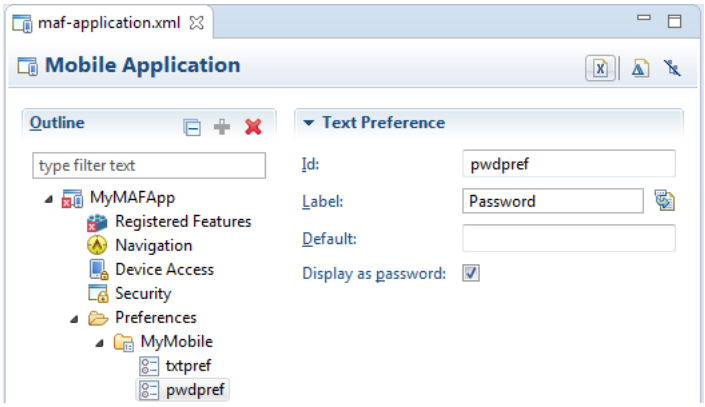


In the PrefDemo application, each MAF AMX preference page is referenced by a single bounded task flow comprised of a view activity and a control flow case that enables the page refresh.

21.1.1 How to Create Mobile Application-Level Preferences Pages

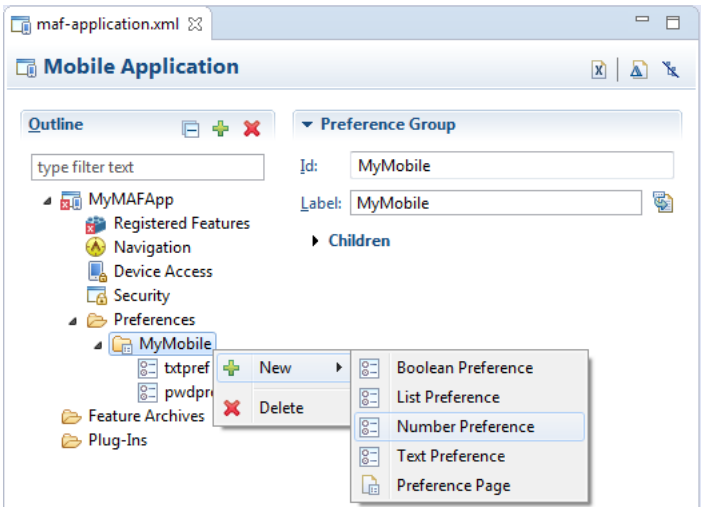
The Preferences section of the MAF Application Feature Editor, shown in Figure 21–4, enables you to build sets of application-level preference pages by nesting the child preference page elements within `<adfmf:preferenceGroup>`. The section presents the `<adfmf:preferenceGroup>` and its child elements as similarly named options (such as Preference Page, Preference List, Boolean Preference, and so on), which you assemble into a hierarchy (or tree).

Figure 21–4 Adding Mobile Application-Level Preferences Using the Preference Page



To ensure that the `maf-application.xml` file is well-formed, use the MAF Application Editor, shown in Figure 21–4, to construct the user preferences pages. This ensures that you can only select an element that can have the appropriate parent, child, or sibling relationship to a selected preferences element. For example, Figure 21–5 shows only the components that can be inserted within the Preference Group element, *MyMobile*. The editor also enables you to enter the values for the attributes specific to each preference element.

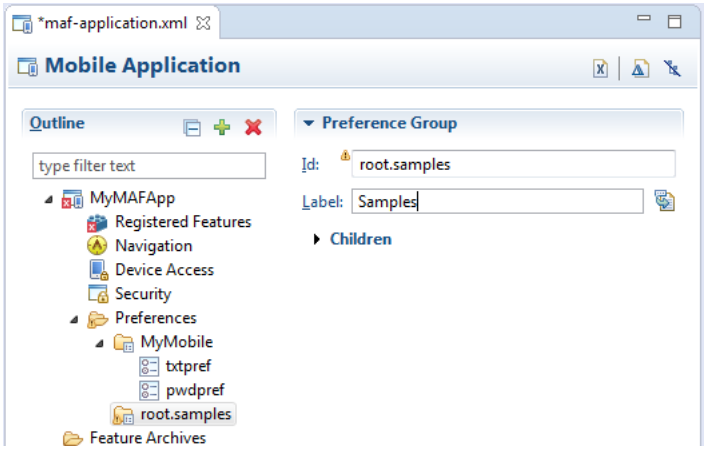
Figure 21–5 Choosing Preference Type



To create preferences pages:

1. In the MAF Application Editor, right-click **Preferences** and choose **New** > **Preference Group** to create the parent `<adfmf:preferenceGroup>` element.
2. Enter the following information for the new preference group, shown in Figure 21–6.

Figure 21–6 Defining the Parent Preference Group Element



- Enter a unique identifier for the Preference Group id element.
- Enter the descriptive text that displays in the user interface. For an example of how this text displays in the user interface, see *Samples* in Figure 21–1.

Note: If a preference ID has a character that is considered an illegal identifier in EL, OEPE displays a warning because the runtime cannot support it properly. There is no problem unless a feature content page attempts to reference the preference using EL.

21.1.1.1 How to Create a New User Preference Page

The Preference Page component enables you to create a new user interface page.

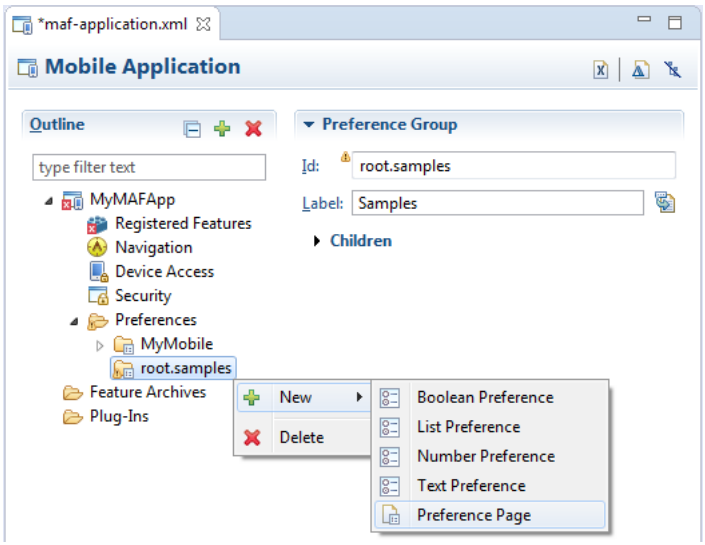
Before you begin:

You must create a Preferences Group element.

To create a new user preference page:

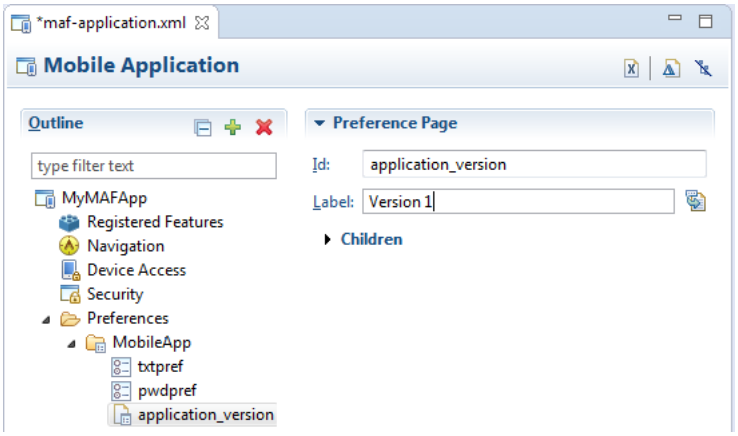
1. Select the Preference Group element.
2. Right-click and choose **New** > **Preference Page**, as shown in Figure 21–7.

Figure 21–7 Selecting the Preference Page Component



3. Define the following Preference Page attributes in the Insert Preference Page dialog, shown in Figure 21–8:
   - Enter a unique identifier for the Preference Page element.
   - Enter the descriptive text that displays in the user interface.

Figure 21–8 Inserting a Preference Page



4. Create the body of the preference page by inserting a child Preference Group element.

   Right-click the Preference Page and choose **New** > **Preference Group**. In Figure 21–8 you would right-click `application_version`.

5. After you define a unique identifier and display name for the child Preference Group, you can populate it with other elements, such as a Preference List element.

21.1.1.2 What Happens When You Add a Preference Page

After you define the Preference Page and its child Preference Group components in the overview editor, OEPE generates an `<adfmf:preferencePage>` with attributes similar to the example below. The `<adfmf:preferencePage>` is nested within a parent `<adfmf:preferenceGroup>` element.

This example shows how to add an `<adfmf:PreferencePage>` element.]]

```
<adfmf:preferences>
          <adfmf:preferenceGroup id="gen"
                                 label="MobileApp">
                     <adfmf:preferencePage id="application_version"
                                           label="Version">
                    <adfmf:preferenceGroup id="version_select"
                                           label="Select Your Version">
                         <adfmf:preferenceList id="edition"
                                               label="Edition"
                                               default="PERSONAL">
                              adfmf:preferenceValue name="Enterprise"
                                                    id="pv2"/>
                              <adfmf:preferenceValue name="Personal"
                                                     value="PERSONAL"
                                                     id="pv1"/>
                         </adfmf:preferenceList>
                    </adfmf:preferenceGroup>
                     </adfmf:preferencePage>
</adfmf:preferences>
```

21.1.1.3 How to Create User Preference Lists

Add a Preference List component to create a list of options.

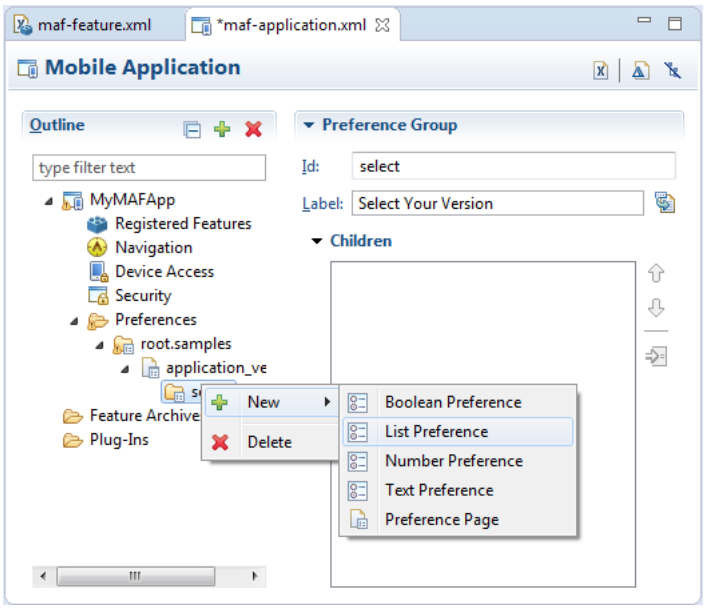
Before you begin:

You must create Preference Group as the parent to the Preference List or any other list-related component.

To create a user preference list:

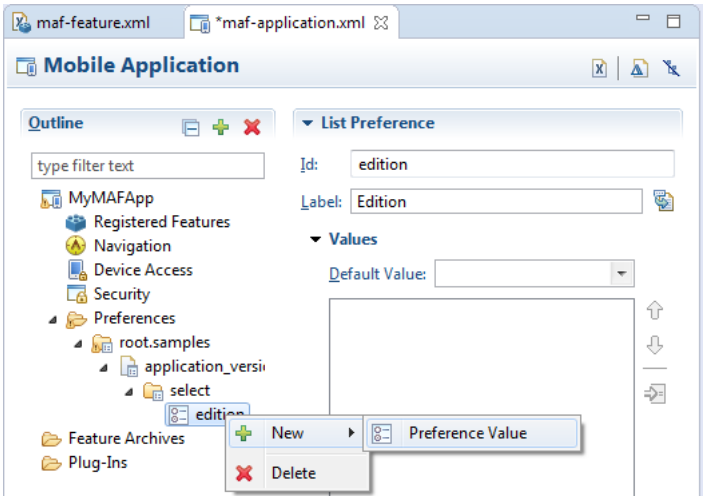
1. Select a Preference Group or Preference Page and then click **Add**, then Insert Inside, and then **Preference List**. Example 21–9 shows adding a Preference List as a child of a Preference Group component called *Select Your Version*.

Figure 21–9 Adding a Preference List Component to a Preference Group



2. Define the following attributes using for the List Preference, shown in Figure 21–10.
   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

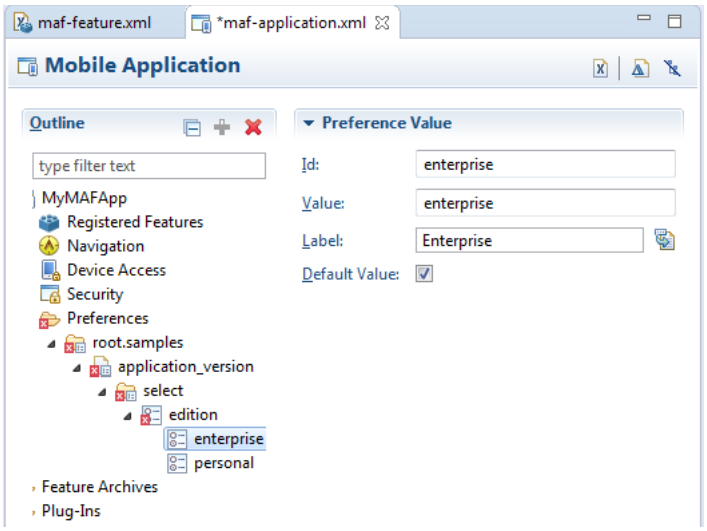
Figure 21–10 Inserting a Preference List



3. Define a list of items by right-clicking the list preference (in Figure 21–10, the list preference is `edition`) and choosing **Preference Value**.
4. Now you can define the values of the list. Enter the **Id**, **Value** and **Label**, as shown in Figure 21–11.

   You can present the user with a default setting by selecting **Default**. As illustrated in the example in the previous section, the default status is defined within the `<adfmf:preferenceList>` element as `default="ENTERPRISE"`.

Figure 21–11 Adding Preference Values



21.1.1.4 What Happens When You Create a Preference List

After you add Preference List component to a Preference Group and then define a series of Preference Values, OEPE updates the `<adfmf:preferences>` section with an `<adfmf:preferenceList>` element, as shown in Section 21.1.1.2, "What Happens When You Add a Preference Page.".

21.1.1.5 How to Create a Boolean Preference List

See the example in Section 21.1, "Creating User Preference Pages for a Mobile Application.".

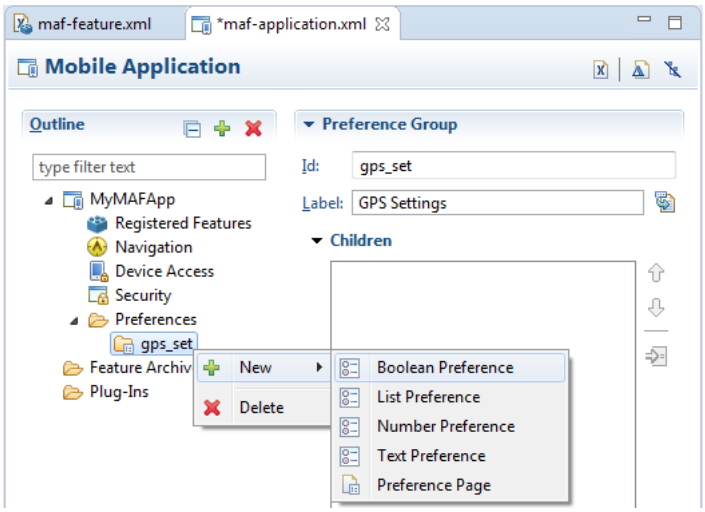
Before you begin:

Because an `<adfmf:preferenceBoolean>` element must be nested within an `<adfmf:preferenceGroup>` element, you must first insert a Preference Group component to the hierarchy.

To create a boolean preference list:

1. Select a Preference Group element, such as GPS Settings in Figure 21–12.

Figure 21–12 Adding a Boolean Preference to a Preference Group

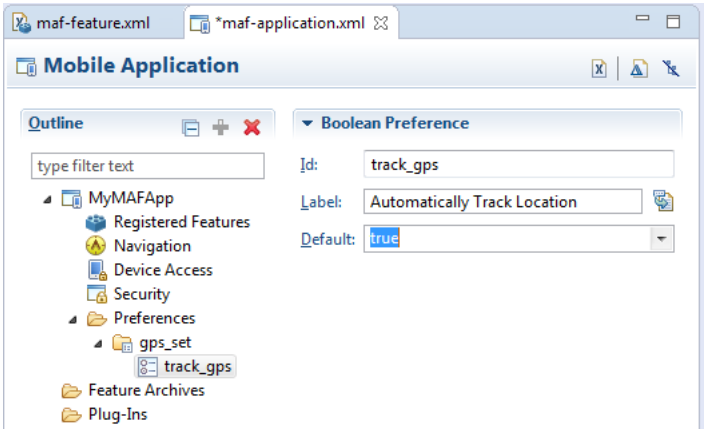


2. Right-click and choose **New** > **Boolean Preference**.

   Define the following attributes in Figure 21–13, and then click **OK**.

   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 21–13 Inserting a Boolean Preference



3. Accept the default value of `false`, or select `true`.

21.1.1.6 What Happens When You Add a Boolean Preference

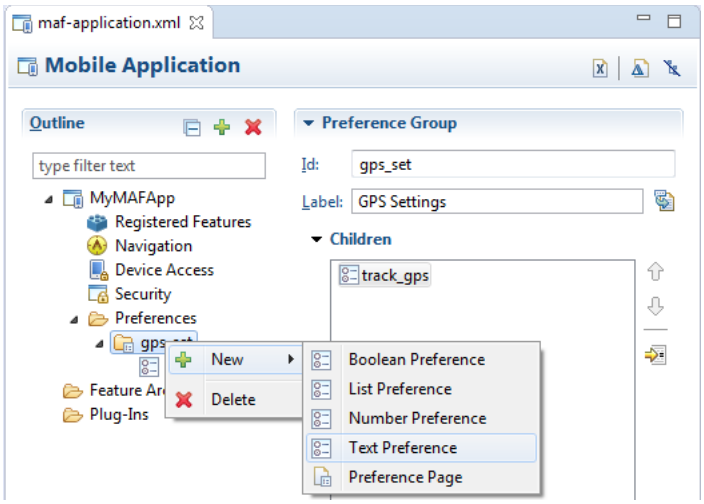
When you add a Boolean Preference and designate its default value, OEPE updates the `<adfmf:preferences>` section of the `maf-application.xml` file with a `<adfmf:preferenceBoolean>` element, as illustrated in the example below.

```
<adfmf:preferencePage id="gps_tracking"
                      label="Your_GPS_Locations">
   <adfmf:preferenceGroup id="gps"
                          label="GPS Settings">
      <adfmf:preferenceBoolean id="track_gps"
                               label="Automatically Track Location"
                               default="true"/>
</adfmf:preferencePage>
```

21.1.1.7 How to Add a Text Preference

Use the insert options, shown in Figure 21–14, to create a Text Preference, a dialog that enables users to store information or view default text.

Figure 21–14 Inserting a Text Preference



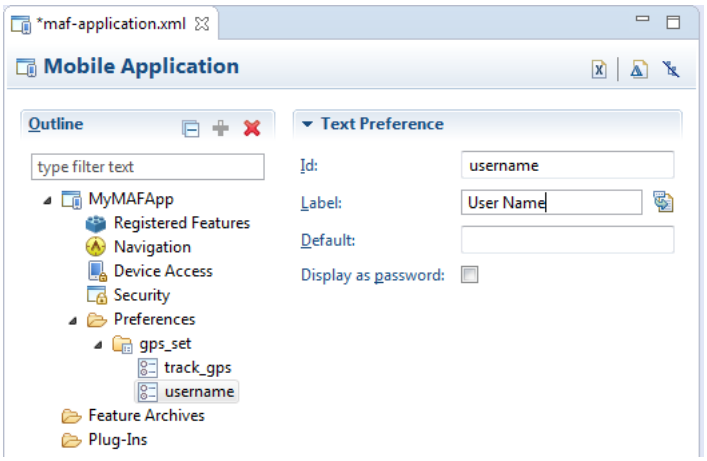
Before you begin:

Create a Preference Group element.

To create a text preference:

1. Select a Preference Group element.
2. Right-click and choose **New** > **Text Preference**.
3. Enter the following information into the Insert Text Preference dialog, shown in Figure 21–15, and then click **OK**.
   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.
   - Enter the default text value.

Figure 21–15 The Insert Text Preference Dialog



21.1.1.8 What Happens When You Define a Text Preference

When you add a Text Preference and designate its default value, OEPE updates the `<adfmf:preferences>` section of the `maf-application.xml` file with a `<adfmf:preferenceText>` element, as illustrated in the example below.

```
<adfmf:preferenceGroup id="security" label="Security">
   <adfmf:preferenceText id="serviceURL"
                         label="Security URL"
                         default="http://security.example.com/provider"/>
   <adfmf:preferenceText  id="username"
                          label="User Name"/>
   <adfmf:preferenceText  id="password"
                          label="Password"
                          secret="true"/>
</adfmf:preferenceGroup>
```

The Preference Group elements that define a security URL, user name, and password preference setting display similarly to Figure 21–16.

Figure 21–16 Text Preferences

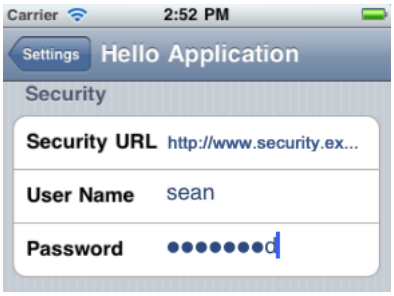


Figure 21–16 illustrates `<adfmf:preferenceText>` elements with a seeded value for the Security URL and an input value for the User Name. Because the MAF preferences are integrated with the iOS Settings application, the `secret="true"` attribute for the Password input text results in the application following the iOS convention of obscuring the user input with bullet points. For more information, see the description for the `isSecure` text field element in *Settings Application Schema Reference*, available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`) and Section 21.4, "Platform-Dependent Display Differences."

21.1.2 What Happens When You Create an Application-Level Preference Page

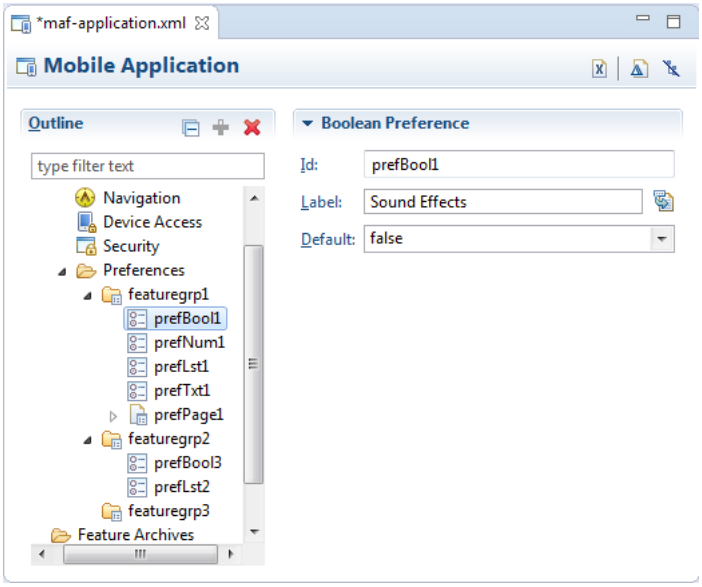
After you deploy the mobile application, the application-wide preference settings page is propagated to the device's global settings application, such as the Settings application on iOS-powered devices. For more information, see Appendix D, "Converting Preferences for Deployment."

21.2 Creating User Preference Pages for Application Features

You can distribute an application feature independently of its mobile application by adding a Feature Application Archive (FAR) `.jar` file containing the application feature to the library of another mobile application. You then reference the application feature in the application's `maf-application.xml` file. If an application feature requires a specific set of user preferences in addition to the general preferences defined for the consuming application, you can define them using the Preferences tab of the MAF

Feature Editor, shown in Figure 21–17. You build application feature preferences in the same manner as the application-level preferences, which are described in Section 21.1, "Creating User Preference Pages for a Mobile Application." After you define the preferences in the MAF Feature Editor, you then create the actual preference page by creating an application feature that references a MAF AMX page that is embedded with the Boolean Switch, Input, and Output components described in Section 12.3, "Creating and Using UI Components."

Figure 21–17 Setting Application Feature-Level Preferences



21.3 Using EL Expressions to Retrieve Stored Values for User Preference Pages

When creating an application feature-level preference page, you add EL expressions to the MAF AMX components, such as the Input Text component in the example later in this section].

To create a feature-level preference page:

1. Open the AMX Editor for the page.
2. Select the XML element (for example, `<amx:inputTest ...>` in the example below].
3. Open the Properties pane.
4. Select the Common tab in the Properties pane.
5. Click the Binding button on the Value attribute. This opens the OEPE EL expression builder.

This example shows how to reference preference values using EL in AMX components.

```
<amx:inputText label="Number" id="it1" inputType="number"
               value="#{preferenceScope.feature.Feature1.f1top.f1Number}"/>
```

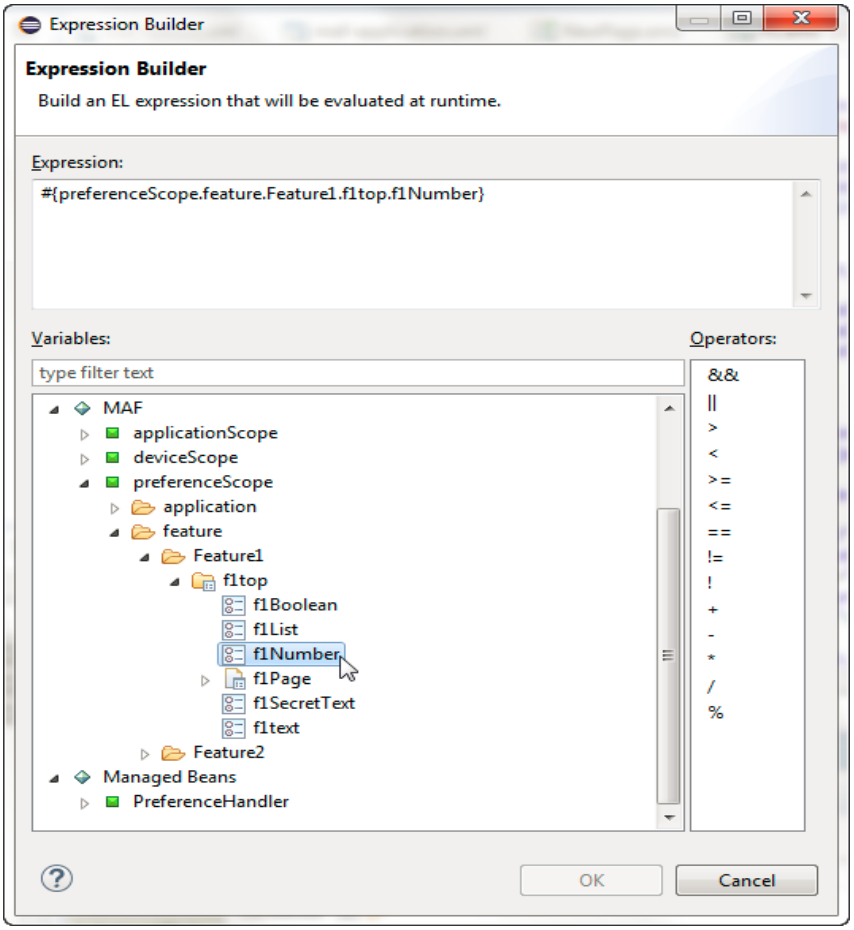
As illustrated in the example above, EL expressions use the `preferenceScope` object to enable applications to access an application feature-level preference. These EL expressions are in the following format:

`preferenceScope.feature.`*`feature-id.group-id.property-id`*

Figure 21–18 illustrates using the Expression Builder to create the EL expression.The preference itself is designated by the IDs configured for various components in `maf-feature.xml`, such as the ID of the application feature `<adfmf:feature id="Feature1">`), the ID of a Preference Group (`<adfmf:preferenceGroup id="f1top">`), and the ID of a preference property (`<adfmf:preferenceNumber id="f1Number">`).

The EL expression may include zero or more `group-id` and `property-id` elements.

Figure 21–18 Building an EL Expression for a Preference



21.3.1 What You May Need to Know About preferenceScope

An EL expression has the following resolution pattern:

- From the JavaScript layer, EL value expressions are resolved using the following JavaScript function:

  ```
  adf.mf.el.getValue(expression, success, failed)
  ```

  The resolution of `adf.mf.el.getValue` begins with an attempt to resolve the expression locally using the JS-EL parser and JavaScript Context Cache. If the

expression cannot be resolved locally, the expression is passed to the embedded Java layer for evaluation where it is resolved by the Java EL parser. This is done through the `GenericInvokeRequest` to the Model's `getValue` method.

- At the Java layer, an EL value expression is resolved using the following approach:

  ```
  String val =
  AdfmfJavaUtilities.evaluateELExpression("#{preferenceScope.feature.f0.vendor}")
  ;
  ```

  For a `setValue` method, the expression is resolved as follows:

  ```
  ValueExpression ve =
  AdfmfJavaUtilities.getValueExpression("#{preferenceScope.feature.f0.vendor}");
  ve.setValue(AdfmfJavaUtilities.getADFELContext(), value);
  ```

  Evaluation of the EL expression involves looking up the `preferenceScope` object. The evaluation is from left to right, where each token is resolved independently. After a token is resolved, it is used to resolve the next token (which is on its right).

Preferences cannot be exposed without the `preferenceScope` object. For more information about the `preferenceScope` object, see Section 13.3.5.3, "About the Mobile Application Framework Objects Category."

21.3.2 Reading Preference Values in iOS Native Views

MAF integrates APIs provided for a native UI (such as `UIView` or `UIViewController`) to allow certain configurations on iOS platform.

When the native UI is initialized, an instance of the `ADFSession` object becomes available. You can use its `getPreferences` method to instruct MAF to provide a listing of the available preferences for the application as defined in the `maf-application.xml` file. As shown in the next example, this method returns a `NSArray*` of preference property objects that can include the `id`, `value`, and `label` for the preference. This API call ensures that either the end user provided the value for a particular preference, or that the default value of the preference is returned.

This example shows how to get preferences.

```
//...
-(id) initWithADFSession:(id<ADFSession>) providedSession
{
   id me = [self init];
   session = providedSession;
   //...
   // Dump the preferences to the data display
   NSArray* prefsArray = [session getPreferences];
   NSString* prefs = [prefsArray JSONRepresentation];
   self.theData.text = [[NSString alloc ] initWithFormat:
        :@"%@\nUser Preferences = --> %@ <--", self.theData.text, prefs];
   //...
   return me;
}
```

21.4 Platform-Dependent Display Differences

The MAF preference pages maintain the native look-and-feel for both the iOS and Android platforms. Consequently, the MAF preference pages display differently on the two platforms. As shown in Table 21–1, preferences display inline on the iOS platform,

meaning that the system does not invoke dialog pages. With a few exceptions, the Android platform presents these components as dialogs.

Table 21–1 Preference Component Comparison by Platform

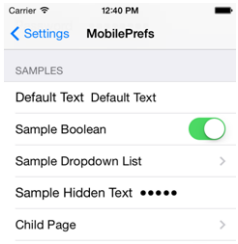
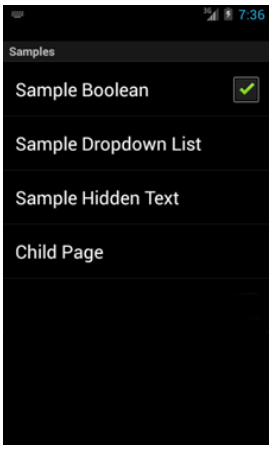
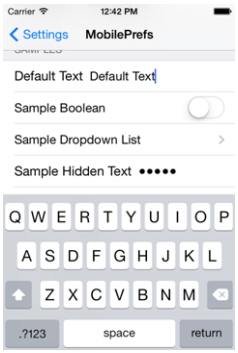
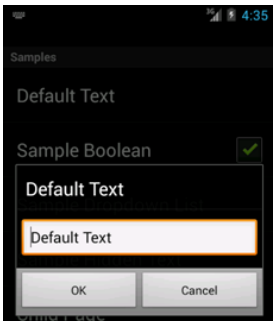
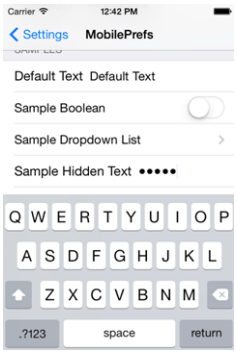
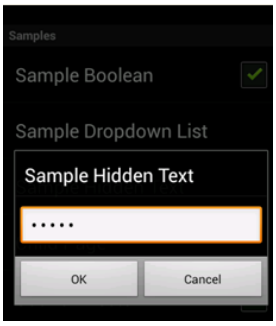
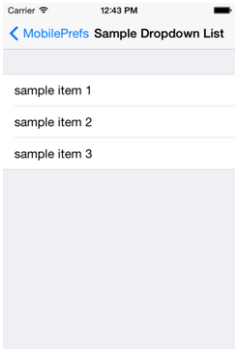
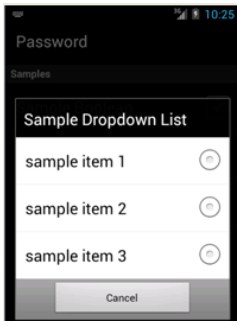
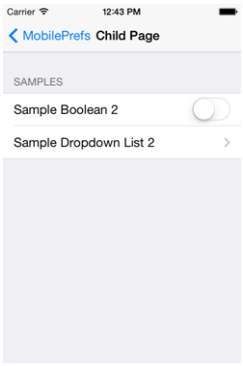
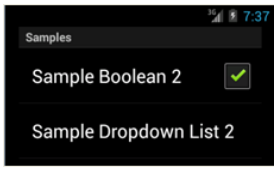
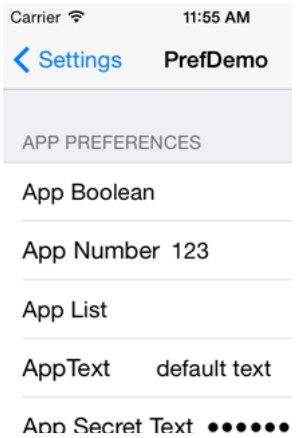
Component	iOS	iOS Display Examples	Android	Android Display Examples
Preference Groups (Category Selection)	The iOS platform displays the preference elements within their parent preference group.		The Android platform displays the preference elements within their parent preference group.	
Boolean Preference List	The Boolean preference is represented as value pair, such as *on* and *off*.		Android presents the Boolean preference as a check box.	
Text Preference	iOS displays the text inline.		Android displays the default text within an input field.	

Table 21–1 (Cont.) Preference Component Comparison by Platform

Component	iOS	iOS Display Examples	Android	Android Display Examples
Text Preference (as secret input text)	On iOS platforms, users enter text inline, with each character obscured by a bullet point after it has been entered. For more information, see Section 21.1.1.8, "What Happens When You Define a Text Preference."		Android launches an input text dialog and obscures each character with a bullet point after it has been entered.	
Single Item Selection List (from a Preference List)	iOS platforms display the single item selection list in a separate preferences page.		Android displays the single item selection list in a dialog.	
Preference Page	iOS launches a child preference page from a preference group.		Android launches a child preference page from a preference group.	

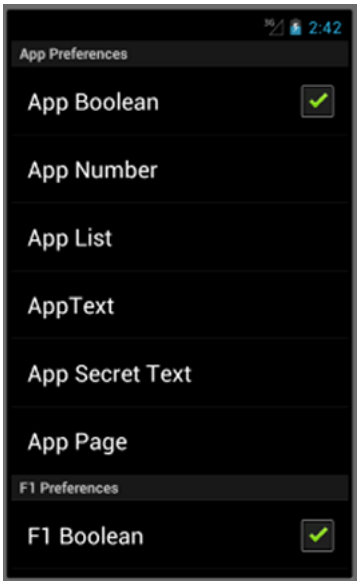
Although iOS and Android platforms have a Settings application, only the iOS platform supports integrating application-level preferences into the Settings application, as shown by the preferences in Figure 21–19.

Figure 21–19 Oracle Mobile Preferences in the iOS Settings Application



On Android-powered devices, users access application-specific preferences pages similar to the one shown in Figure 21–20 only when the application is running.

Figure 21–20 The Preferences Menu on an Android-Powered Device



22

Setting Constraints on Application Features

This chapter describes how to set constraints that can restrict an application feature based on user access privileges or device requirements.

This chapter includes the following sections:

- Section 22.1, "Introduction to Constraints"
- Section 22.2, "Defining Constraints for Application Features"

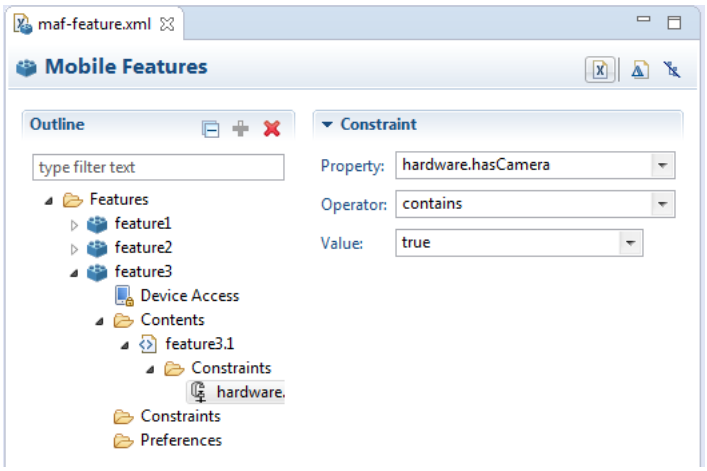
22.1 Introduction to Constraints

A constraint describes when an application feature or application content should be used. Constraints can restrict access based on users and user roles, the characteristics of the device on which the mobile application is targeted to run, and the hardware available on the device. You can set constraints at two levels: at the application feature level, where you control the visibility of an application feature on a user's device, and at the content level, where you can specify which type of MAF content can be delivered for an application feature. The overview editor for the `maf-feature.xml` file enables you to set both of these types of constraints. Constraints are evaluated by the MAF runtime and must evaluate to `true` to enable the end user to view or use specific content, or even access the application feature itself.

22.1.1 Using Constraints to Show or Hide an Application Feature

The Constraints folder under the Contents folder in the MAF Feature Editor, shown in Figure 22–1, enables you to set the application feature-level constraints. For example, an application feature that uses the device's camera displays within the mobile application's navigation bar or springboard only if the MAF runtime determines that the device actually has a camera function. You can also use feature level constraints to secure an application based on user roles and privileges. Figure 22–1 illustrates creating constraints that would allow only a user with administrator privileges to access the application, should the MAF runtime evaluate the constraint to `true`. If the runtime evaluates the constraint to `false`, then it prevents an end user from accessing the application feature, because it does not appear on the navigation bar or within the springboard.

Figure 22–1 Setting Application Feature-Level Constraints



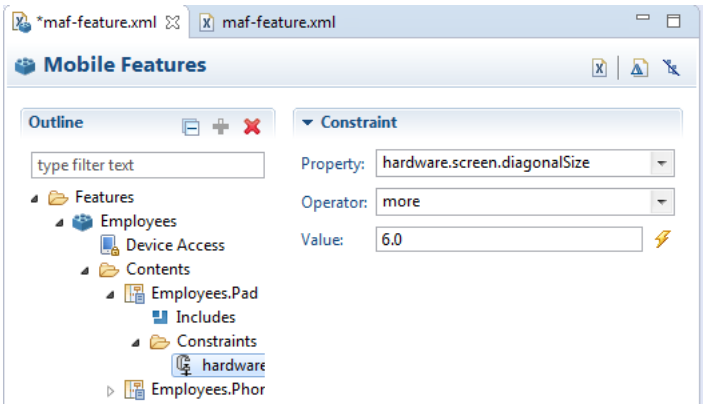
22.1.2 Using Constraints to Deliver Specific Content Types

To accommodate such factors as device hardware properties or user privileges, a single application feature can have more than one type of content to deliver different versions of the user interface. By setting constraints on the content of an application feature, you designate the conditions for determining what end users can see or how application pages can be used.

Using the Contents folder, shown in Figure 22–2, you can, for example, specify content that delivers one type of user interface for users who have been granted administrative privileges and a separate user interface for those who have basic user privileges. In addition, content-level constraints can accommodate the layout considerations of a device. Figure 22–2 illustrates how an application performs this using a constraint based on the screen width of a device to deliver AMX Mobile task flows that call pages tailored to the layout of the iPhone and the iPad. When an end user launches the application, the MAF runtime evaluates the constraint that is set for the Employees application feature. If the runtime ascertains that the diagonal width of the device's screen exceeds six inches, it selects the `Employees_pad_taskflow.xml` file, which calls the MAF AMX pages designed for the iPad. If this constraint evaluates to `false` (that is, the diagonal width of the screen is less than six inches), then the runtime selects the MAF taskflow that calls iPhone-specific pages, `Employees_phone_taskflow.xml`. In addition, the Contents folder enables you to select navigation bar and springboard images that display when the runtime selects specific content. If you do not select content-specific images, then MAF instead uses the application feature-level images that are designated in the Feature-level folder.

Note: Images must adhere to the platform-specific guidelines, as described in Section 3.2, "Setting Display Properties for a MAF Application."

Figure 22–2 Setting Content-Level Constraints



22.2 Defining Constraints for Application Features

When setting application feature-level constraints, the `property`, `operator`, and `value` attributes of the `<adfmf:constraint>` element (a child element of `<adfmf:constraints>`) enable you to restrict application usage based on a user, a device, or hardware. An example of defining these attributes, shown in the example below, illustrates defining these attributes to restrict access to an application feature to a Field Rep and to also restrict the application to run only on an iOS-powered device.

This exmaple shows the `<adfmf:constraint>` element.

```
<adfmf:constraints>
      <adfmf:constraint property="user.roles"
                        operator="contains"
                        value="Field Rep"/>
      <adfmf:constraint property="device.model"
                        operator="contains"
                        value="ios"/>
</adfmf:constraints>
```

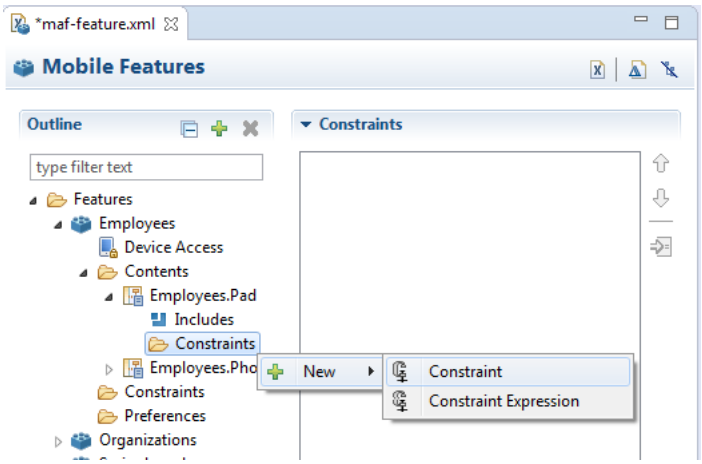
22.2.1 How to Define the Constraints for an Application Feature

You declaratively configure the constraints for a selected application feature using the Constraints folder in the MAF Feature Editor, shown in Figure 22–2.

Defining the constraints for an application feature:

1. Right-click the **Constraints** folder and choose **New** and choose **Constraint**, as shown in Figure 22–3.

Figure 22–3 Creating a New Constraint



2. Select a property and an appropriate operator and then enter a value. For more information on using properties, see Section 22.2.3, "About the property Attribute."

22.2.2 What Happens When You Define a Constraint

Entering the values in the Constraints folder updates the descriptor file's `<adfmf:constraints>` element with defined `<adfmf:constraint>` elements, similar to the example near the start of this section.

22.2.3 About the property Attribute

MAF provides a set of `property` attributes that reflect users, devices, and hardware properties. Using these properties in conjunction with the following operators and an appropriate value determines how an application feature can be used.

- `contains`
- `equal`
- `less`
- `more`
- `not`

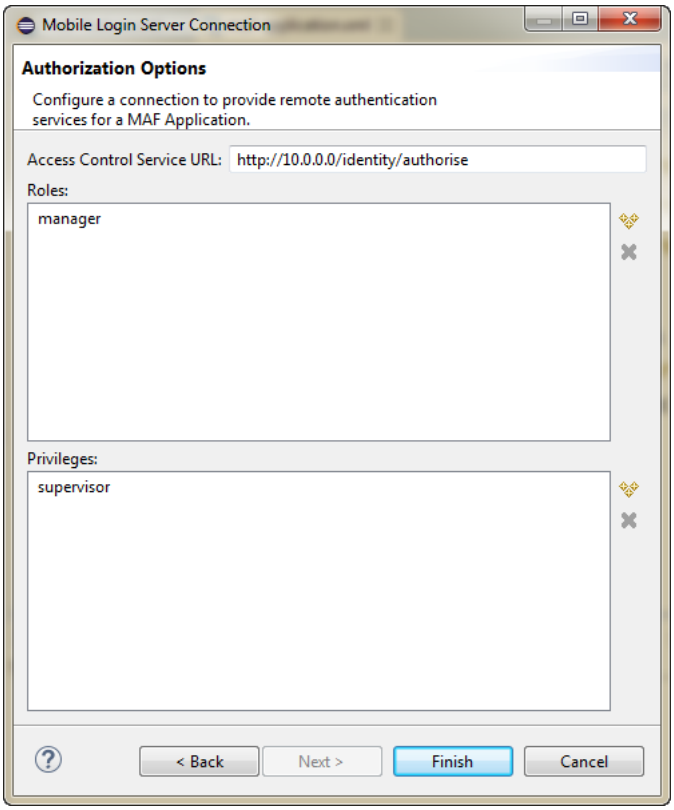
22.2.4 About User Constraints and Access Control

After a user logs into a mobile application, the MAF runtime reconciles the user role-based constraints configured for each application feature against the user roles and privileges retrieved by the Access Control Service (ACS). MAF then presents only the application feature (or application feature content) to end users whose privileges satisfy the constraints. In addition to setting these user privilege and role constraints, you create access control for the mobile application by entering the following in the Mobile Login Server Connection dialog, shown in Figure 22–4 (and described in Section 29.6.2, "How to Designate the Login Page"):

- The URL of the REST web service that transmits a list of user roles and privileges.
- A list the user roles checked by the application feature.
- A list of privileges.

See also Section 29.5.19, "What You May Need to Know About the Access Control Service."

Figure 22–4 Configuring Retrieval of User Roles and Privileges



You control access to application features using constraints based on `user.roles` and `user.privileges`. For example, to allow only a user with the *manager* role to access an application feature, you must add a constraint of `user.roles contains manager` to the definition of the application feature.

The `user.roles` and `user.privileges` use the `contains` and `not` operators as follows:

- `contains`—If the collection of roles or privileges contains the named role or privilege, then the runtime evaluates the constraint to `true`. The next example shows an example of using the `user.roles` property with the `contains` operator. The application feature will appear in the mobile application if the user's roles include the role of employee.

```
<feature ...>
   ...
   <constraints>
      <constraint property="user.roles"
                  operator="contains"
                  value="employee" />
   </constraints>
   ...
</feature>
```

- `not`—If the collection of roles or privileges does not contain the named role or privilege, then the runtime evaluates the constraint to `true`. In the next example,

the application feature is not included if the user's privileges contain the manager privilege.

This example shows how to use the not operator with the `user.privileges` property to restrict access to an application feature.

```
<feature ...>
   ...
   <constraints>
      <constraint property="user.privileges"
                  operator="not"
                  value="manager" />
   </constraints>
   ...
</feature>
```

22.2.5 About Hardware-Related Constraints

The hardware object references the hardware available on the device, such as the presence of a camera, the ability to provide compass heading information, or to store files. These properties use the `equal` operator.

- `hardware.networkStatus`—Indicates the state of the network at the startup of the application. This property can be modified with three attribute values: `NotReachable`, `CarrierDataConnection`, and `WiFiConnection`. The example below illustrates the latter value. As illustrated in this example, setting this value means that this mobile application feature only displays in the mobile application if the device hardware indicates that there is a Wi-Fi connection. In other words, if the device does not have a Wi-Fi connection when the mobile application loads, then this application feature will not display.

  This example shows how to define the `hardware.networkStatus` property.

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.networkStatus"
                  operator="equal"
                  value="WiFiConnection" />
   </constraints>
...
</feature>
```

> **Note:** This constraint is evaluated at startup on iOS-powered devices. If a device does not have a Wi-Fi connection at startup but subsequently attains one (for example, when a user enters a Wi-Fi hotspot), then the application feature remains unaffected and does not become available until the user stops and then restarts the mobile application.

- `hardware.hasAccelerometer`—Indicates whether or not the device has an accelerometer. This property is defined by a `true` or `false` value. The example below shows a `true` value, indicating that this application feature is only available if the hardware has an accelerometer.

  This example shows how to use the `hardware.hasAccelerometer` property.

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.hasAccelerometer"
                  operator="equal"
                  value="true" />
</constraints>
...
</feature>
```

> **Note:** Because all iOS-based hardware have accelerometers, this property must always have a value of `true` for the application feature to be available on iOS-powered devices.

- `hardware.hasCamera`—Indicates whether or not the device has a camera. This constraint is defined using a value attribute of `true` or `false`. In the example below, the value is set to `true`, indicating that the application feature is only available if the device includes a camera.

  This example shows how to use the `hardware.hasCamera` property.

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.hasCamera"
                  operator="equal"
                  value="true" />
</constraints>
...
</feature>
```

> **Note:** Not all iOS-based hardware have cameras. This value is dynamically evaluated at the startup of mobile applications on an iOS-powered device. At this time, the mobile application removes the application features that do not evaluate to `true`.

- `hardware.hasCompass`—Indicates whether the device has a compass. You define this constraint with the attribute value of `true` or `false`, as shown in the example below.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasCompass"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Not every iOS-powered device has a compass. This value is dynamically evaluated at the startup of mobile applications on an iOS-powered device. At this time, the mobile application removes the application features that do not evaluate to `true`.

- `hardware.hasContacts`—Indicates whether the device has an address book or contacts. You define this constraint with the attribute value of `true` or `false`, as shown in the example below.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasContacts"
                  operator="equal"
                  value"=true" />
      </constraints>
   ...
</feature>
```

> **Note:** Because contacts on iOS-based hardware are accessed through Apache Cordova, the `value` attribute is always set to `true` for iOS-powered devices.

- `hardware.hasFileAccess`—Indicates whether the device provides file access. You define this constraint with the attribute value of `true` or `false`, as shown in the example below. The application feature is only available if the runtime evaluates this constraint to `true`.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasFileAccess"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Because file access on iOS-based hardware is accessed through Apache Cordova, the `value` attribute is always `true` for iOS-powered devices.

- `hardware.hasGeoLocation`—Indicates whether or not the device provides geolocation services. You define this constraint with the attribute value of `true` or `false`, as shown in the example below. The application feature is only available if the device supports geolocation.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasGeoLocation"
                  operator="equal"
                  value="true"/>
```

```
   </constraints>
   ...
</feature>
```

> **Note:** Apache Cordova does not provide access to the gelocation service for all iOS-powered devices. Depending on the device, the application feature may not be available when the constraint is evaluated by the runtime.

- `hardware.hasLocalStorage`—Indicates whether the device provides local storage of files. You define this constraint with the `value` attribute of `true` or `false`, as shown in the example below. The application feature only displays if the device supports storing files locally.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasLocalStorage"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Because Apache Cordova provides access to local file storage on all iOS hardware, the `value` attribute is always `true` for iOS-powered devices.

- `hardware.hasMediaPlayer`—Indicates whether or not the device has a media player. You define this constraint with the `value` attribute of `true` or `false`, as shown in the example below. The application feature only displays if the device has a media player.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasMediaPlayer"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** For iOS-powered devices, the `value` attribute is always `true`, because Apache Cordova provides access to media players on iOS-based hardware.

- `hardware.hasMediaRecorder`—Indicates whether or not the device has a media recorder. You define this constraint with the `value` of `true` or `false`, as shown in the example below. The application feature is only included if the device hardware supports a media recorder.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasMediaRecorder"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

Note: Set this value to `true` for all iOS-powered devices because all iOS-based hardware have media recorders which can be accessed through Apache Cordova. Some devices, such as the Apple iTouch, do not have a microphone but can allow end users to make recordings by attaching an external microphone.

- `hardware.hasTouchScreen`—Indicates whether or not the hardware provides a touch screen. You define this constraint with the `value` attribute of `true` or `false`, as shown in the example below. The application feature is only included if the device hardware supports a touch screen.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasTouchScreen"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

Note: Set the `value` attribute to `true` for iOS-powered devices, because all iOS-based hardware provides touch screens.

- `hardware.screen.width`—Indicates the width of the screen for the device in its current orientation. Enter a numerical value that reflects the screen's width in terms of logical device pixels (such as 320 in the example below), not physical device pixels, which represent the actual pixels that appear on a device. The value depends on the orientation of the device.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.width"
                  operator="equal"
                  value="320" />
   </constraints>
   ...
</feature>
```

> **Note:** This value is evaluated at the startup of the mobile application when the runtime evaluates constraints and dismisses application features with constraints that do not evaluate to `true`. If a user rotates the device after the mobile application starts, MAF's runtime does not re-evaluate this constraint because the set of application features is fixed after the mobile application starts.

- `hardware.screen.height`—Indicates the height of screen for the device in its current position. Enter a numerical value that reflects the screen's height in terms of logical pixels, such as 320 or 480, as shown in the example below. The value depends on the orientation of the device.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.height"
                  operator="equal"
                  value="480" />
   </constraints>
   ...
</feature>
```

> **Note:** When the mobile application starts, the MAF runtime evaluates the screen height value for this constraint as part of the process of dismissing application features with constraints that do not evaluate to `true`. If a user changes the orientation of the device after the mobile application starts, the runtime does not re-evaluate this constraint, because the set of application features is fixed after the mobile application starts.

- `hardware.screen.availableWidth`—Indicates the available width of the device's screen in its current orientation. Enter a numerical value that reflects the screen's width in terms of logical pixels, such as 320 or 480, as shown in the example below. The value depends on the orientation of the device.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.availableWidth"
                  operator="equal"
                  value"320" />
   </constraints>
   ...
</feature>
```

- `hardware.screen.availableHeight`—Indicates the available height of the screen for the device in its current position. Enter a numerical value that reflects the screen's width in terms of logical pixels, such as 320 or 480, as shown in the example below. The value depends on the orientation of the device.

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.availableHeight"
                  operator="equal"
```

```
                    value"480" />
      </constraints>
      ...
   </feature>
```

22.2.6 Creating Dynamic Constraints on Application Features and Content

In addition to displaying or hiding an application feature or user interface content based on the static constraints that are defined by the `name`, `operator`, and `value` attributes, you can enable a mobile application to render its application features and content dynamically by defining constraints with EL expressions. The dynamic evaluation of constraints based on EL expressions enables you to write expressions that can call your own bean logic, write complex EL expressions, or even write logic-accessing application preferences. Defining constraints as EL expressions provides flexibility in that the MAF runtime may initially hide an application feature if it evaluates an EL expression as `false`, but may display it at a later point when it evaluates the same EL expression as `true`. The `<adfmf:constraintExpression>` element enables you to define constraints on an application feature using EL expressions, as illustrated by the deferred method expression in the example below, which shows how to define a dynamic constraint with EL expressions.

```
<adfmf:constraints>
   <adfmf:constraint id="c1" property="hardware.screen.dpi" operator="more"
value="120"/>
   <adfmf:constraint id="c2" property="device.model" operator="equal"
value="iPad"/>
   <adfmf:constraintExpression id="c3" value="#{myBean.checkConstraint}"/>
</adfmf:constraints>
```

As also illustrated by the example above, you can nest this element among the static constraints defined within the `<adfmf:constraints>` element of the `maf-feature.xml` file. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

22.2.6.1 About Combining Static and EL-Defined Constraints

While the MAF runtime must evaluate all the criteria of a static constraint to `true` to enable it to display, it likewise displays application features and content when it evaluates the constraint EL expressions as `true`, but hides them when it evaluates the expressions as `false`.

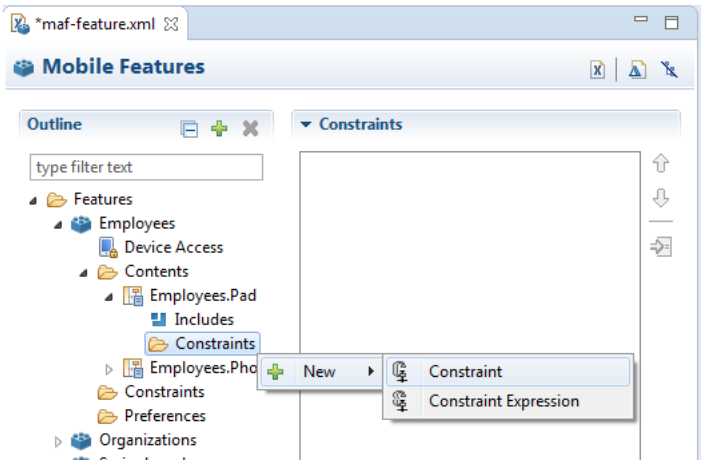
22.2.6.2 How to Define a Dynamic Constraint

Unlike static constraints, you do not create (or update) a dynamic constraint using the `maf-feature.xml` overview editor. Instead, you create an `<adfmf:constraintExpression>` by dragging the Constraint Expression component into either the Source editor or the Structure window and then use the Expression Builder to populate this component with the EL expression.

To define an Constraint Expression component:

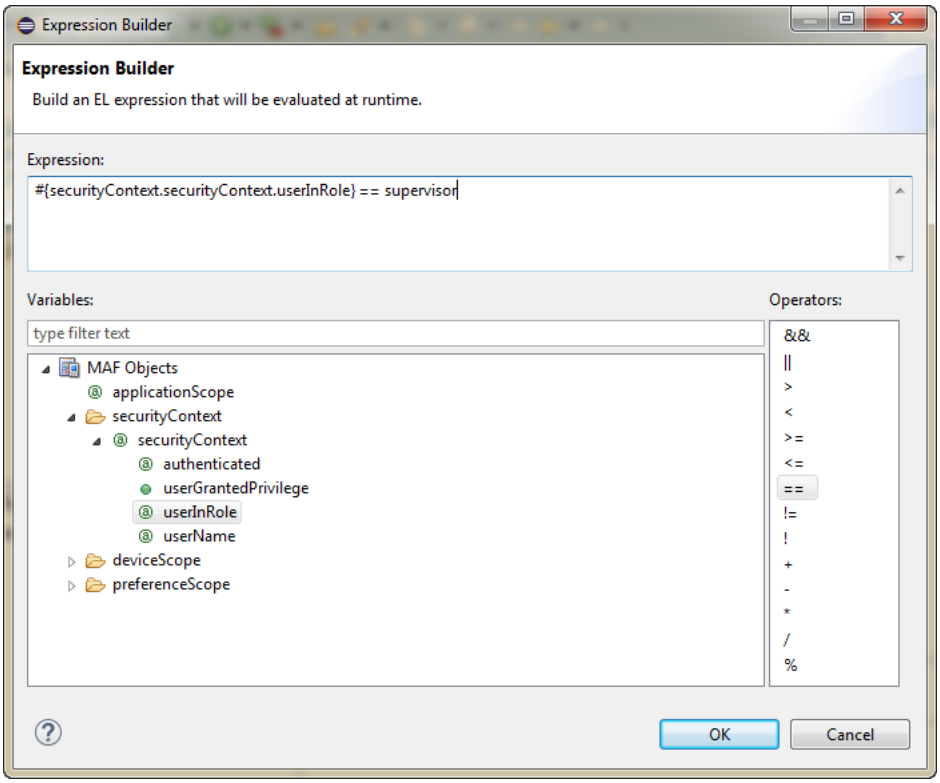
1. Right-click the **Constraints** folder and choose **New** and choose **Constraint Expression**, as shown in Figure 22–5.

Figure 22–5 Creating a Constraint Expression



2. Click to open the Expression Builder dialog as illustrated in Figure 22–6.

Figure 22–6 Building a Constraint's EL Expression



3. In the Expression Builder dialog, create the expression by double-clicking an object in the list, then double-click an operator, then complete the expression.
4. When you are finished, click **OK**.

23

Using AppXray for MAF Artifacts

This chapter introduces AppXRay, a solution provided by OEPE for dependency tracking, validation, visualization, and refactoring support. AppXray is enabled for all MAF artifacts including `maf-application.xml`, `maf-feature.xml`, AMX pages, MAF Task Flows and Data Controls.

This chapter includes the following sections:

- Section 23.1, "Introduction to Using AppXray for MAF Artifacts"
- Section 23.2, "Using AppXray"
- Section 23.3, "Refactoring with AppXray"

23.1 Introduction to Using AppXray for MAF Artifacts

Provided by OEPE, AppXray technology analyzes the MAF artifacts of your application and uses this information to provide validation and consistency checking across many layers of the application.

You can use AppXray with your MAF applications, for which it is enabled by default. AppXray builds its application database that allows it to track artifacts and populate it. MAF artifacts will be detected and added to the database. As you work with your application, AppXray will automatically maintain the application database. If this degrades performance, you can selectively disable automatic maintenance and rebuild the application database as required.

Errors detected by AppXray are displayed in the Problems view, whereas dependencies are graphically displayed in AppXaminer.

AppXray collects information about the following:

- `maf-application.xml`
- `maf-feature.xml`
- AMX pages
- MAF Task Flows
- Data Controls
- Resource bundles
- CSS
- Image files

23.2 Using AppXray

When you use AppXray, the tool for viewing dependencies is called AppXaminer.

23.2.1 How to Open AppXaminer

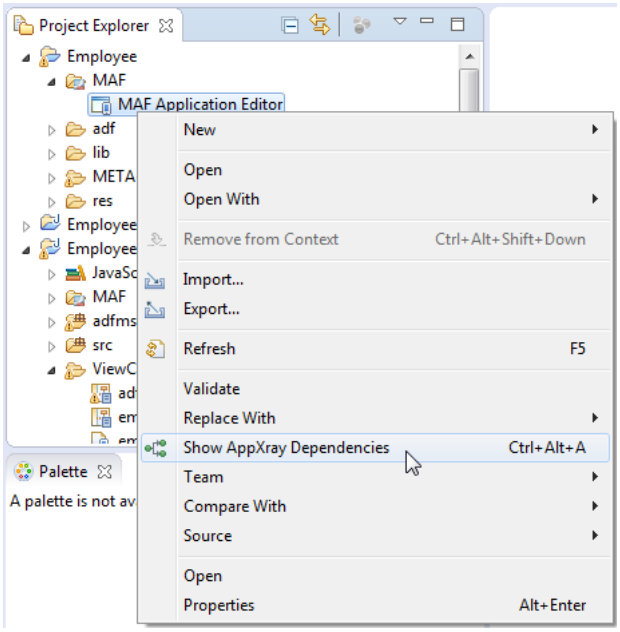
AppXray allows OEPE to provide refactoring support across MAF artifacts. The refactoring options allow you to rename, move, and delete the artifacts that your application uses. These refactoring options synchronize your changes with other parts of the application that are dependent on the changes.

You open AppXaminer from the context menu of a MAF artifact in the Project Explorer.

To view the dependency relationships:

1. In the Project Explorer, expand the assembly project node and navigate to the **MAF >MAF Application Editor**. Right-click on the MAF Application Editor and select Show AppXray Dependencies from the context menu. You can view the dependencies on any editors under the MAF node and also on any file artifacts in the project.
2. Right-click on the **MAF Application Editor** node and choose **Show AppXray Dependencies**, as shown in Figure 23–1.

Figure 23–1 Opening AppXaminer

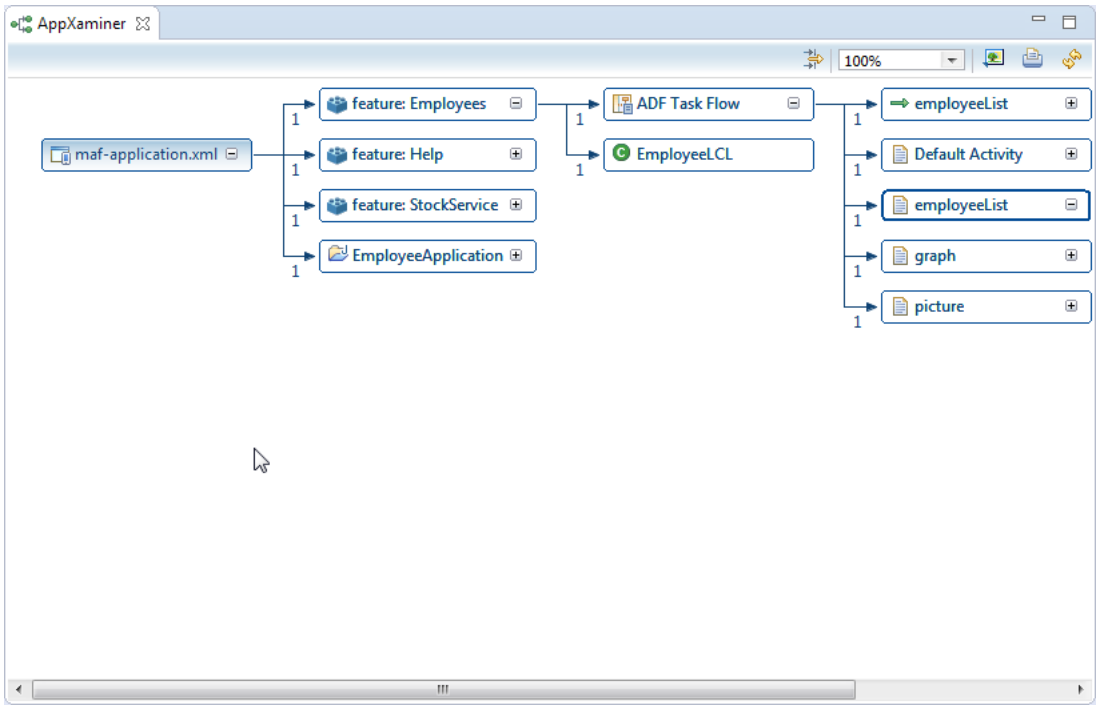


You can choose **Show AppXray Dependencies** from the context menu of any editors under the MAF node, and also on any file artifacts in the project to view dependencies.

23.2.1.1 About AppXaminer

AppXaminer, the UI viewer for AppXray, opens in the editor displaying the relationship the selected artifact has with other components, as shown in Figure 23–2.

Figure 23–2 The AppXaminer Window



23.2.2 Using AppXaminer

AppXaminer allows you to immediately see the relationships between artifacts. As shown in Figure 23–2:

- Numeric values indicate the number of references a component has with another.
- You can expand a node to see the relationship it has with other components.

You can do the following from the context menu of AppXaminer:

- Choose **Show AppXray Dependencies** to show the dependencies of that artifact.
- Choose **Show Reference Detail**. This opens a popup window which displays the detailed components involved.
- Choose **Open** to view the file in the editor.

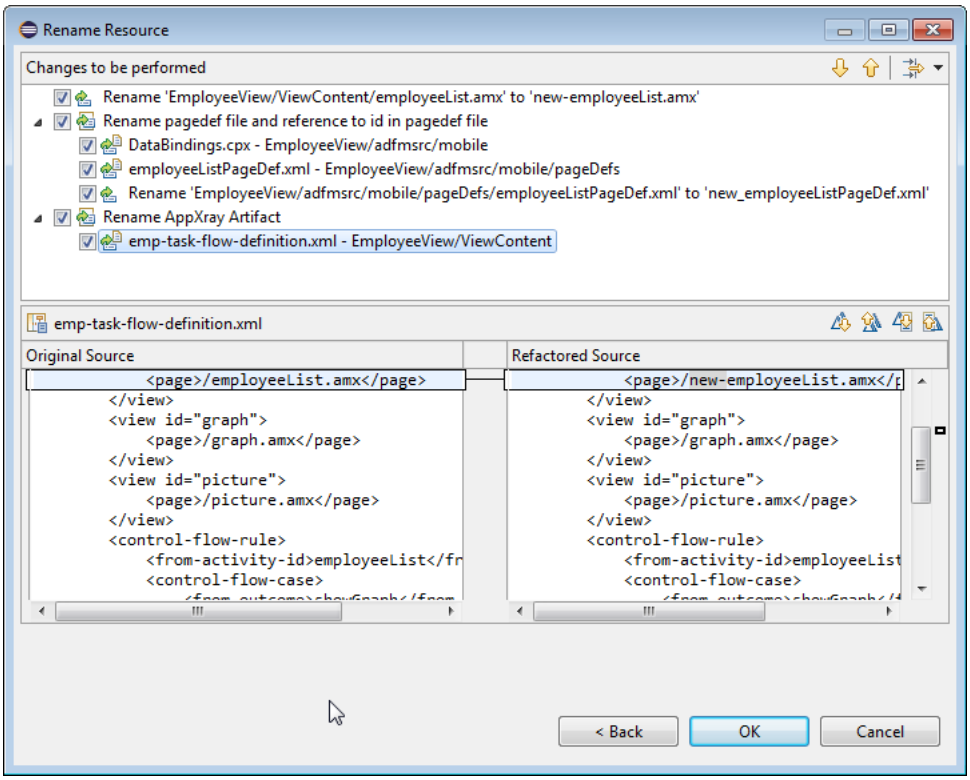
23.3 Refactoring with AppXray

AppXray allows OEPE to provide refactoring support across MAF artifacts, so rename, move, and delete the artifacts that your application uses.

These refactoring options synchronize your changes with other parts of the application that are dependent on the changes.

When you refactor an artifact, AppXRay displays a Rename Resource dialog, in which you can check and confirm the changes you are making. See Figure 23–3

Figure 23–3 Refactoring with AppXray



24

Enabling and Using Notifications

This chapter describes how to enable MAF applications to display local notifications as well as register for, and handle, push notification messages.

This chapter includes the following sections:

- Section 24.1, "Introduction to Notifications"
- Section 24.2, "Enabling Push Notifications"
- Section 24.3, "Managing Local Notifications"

24.1 Introduction to Notifications

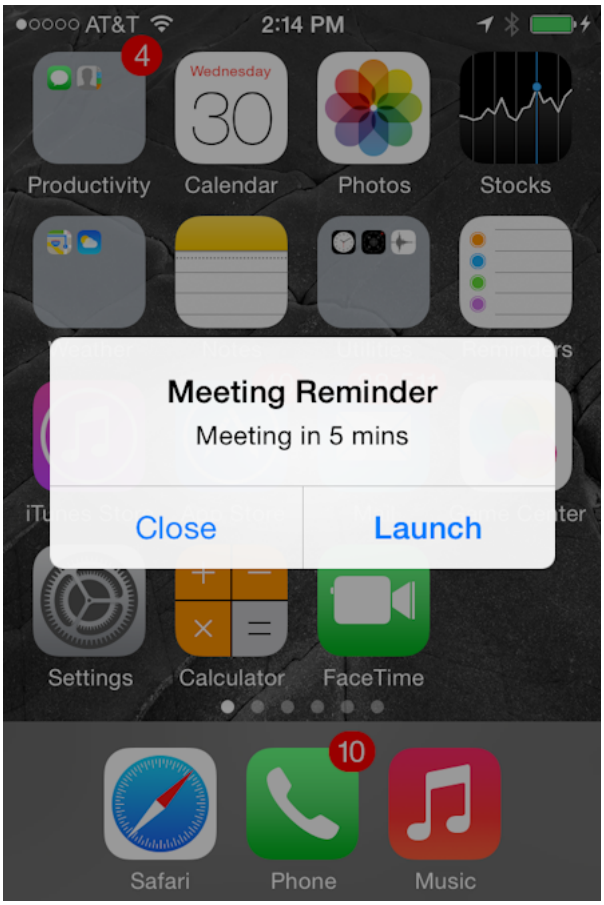
Notifications are signals delivered to the end user outside of a mobile application's regular user interface. These notifications can appear as messages in the form of an alert, or as a banner, depending on the state of the application and user settings. The notifications may be presented visually or as a sound or both.

The following are the two main types of notifications:

- **Push notifications** are sent from an external source, such as a server, to an application on a mobile device. When end users are notified, they can either launch the application or they can choose to ignore the notification in which case the application is not activated.

  Figure 24–1 shows a push notification alert on an iOS-powered device.

Figure 24–1 Push Notification



Applications must register with a notification service to receive push notifications. If the registration succeeds, then the notification service issues a token to the application. The application shares this token with its provider (located on a remote server), and in doing so, enables the provider to send notifications to the application through the notification service. MAF registers on behalf of the application using application-provided registration configuration, described in Section 24.2, "Enabling Push Notifications." Registration occurs upon every start of the MAF application to ensure a valid token. After a successful registration, MAF shares the token obtained from the platform-specific notification service with the provider. On iOS, the notification service is Apple Push Notification Service (APNs). Google Cloud Messaging (GCM) for Android provides the notification service for applications installed on Android-powered devices.

A MAF application can receive push notifications regardless of its state: the display of these messages, which can appear even when the application is not in the foreground, depends on the state of the MAF application and the user settings. Table 24–1 describes how the iOS operating system handles push notifications depending on the state of the MAF application.

Table 24–1 Handling Push Notifications on an iOS-Powered Device

State	Action
The MAF application is installed, but not running.	The notification message displays with the registered notification style (none, banner, or alert). When the user taps the message (if its a banner-style notification) or touches the action button (if the message appears as an alert), the MAF application launches, invoking the application notification handlers.
The MAF application is running in the background.	The notification message displays with the registered notification style (none, banner, alert). When the user taps the message (if it is a banner-style notification), or touches the action button (if the message appears as an alert), the MAF application launches, invoking the application notification handlers.
The MAF application is running in the foreground.	No notification message displays. The application notification handlers are invoked.

On the iOS and Android platforms, if the application is not running in the foreground, then any push notification messages associated with it are queued in a specific location, such as the iOS Notification Center or the notification drawer on Android-powered devices.

- **Local notifications** originate within a mobile application and are received by the same application. The notifications are delivered to the end user through standard mechanisms supported by the mobile device platform (for example, banner, sound).

  Using the Local Notification Abstraction API provided by MAF, you can configure the application to raise a notification immediately or schedule a notification for a future date and time. In addition, you can set a repeat pattern for a notification (for example, daily or weekly) as well as cancel a scheduled notification.

  On both the iOS and Android platform, if the MAF application is running in the foreground, the notification is delivered directly to the application without the end user interaction. If the application is either running in the background or not running at all, the notification is delivered to the application once the end user taps on the notification.

24.2 Enabling Push Notifications

You can enable push notifications by performing the following tasks:

1. Allow the mobile application to receive push notifications by enabling the core plugin **PushPlugin** in the Plugin Enablement section of the MAF Application Editor, and associating the plugin with one or more application features. For more information, see Chapter 9, "Using Plugins in MAF Applications."

   Note: By default, a MAF application does not allow push notifications (nor any other device type of device access).

2. In the application controller project, register an application lifecycle event listener (ALCL) class. For more information, see Section 3.2, "Setting Display Properties for a MAF Application." and Section 10.1, "Introduction to Lifecycle Listeners in MAF Applications."

3. Implement the `oracle.adfmf.application.PushNotificationConfig` interface in the ALCL. This interface provides the configuration required to successfully register with the push notification service.

   Override and implement the `getNotificationStyle` and `getSourceAuthorizationId` methods of the `PushNotificationConfig` interface. The `getNotificationStyle` method enables you to specify the notification styles for which the application registers. The `getSourceAuthorizationId` method enables you to enter the Google Project Number of the accounts authorized to send messages to the mobile application. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

4. In the application controller project, create a push notification event listener class (for example, `NativePushNotificationListener`) that handles push notification events. This class must implement the `oracle.adfmf.framework.event.EventListener` interface that defines an event listener. For more information on the `oracle.adfmf.framework.event.EventListener` interface, see *Java API Reference for Oracle Mobile Application Framework*.

   Override and implement the `onOpen`, `onMessage`, and `onError` methods to register for and receive notification events. After a successful registration with the push notification service, MAF calls the `onOpen` method with the registration token that must be shared with the provider by the application developer. The `onError` method is invoked if there is an error when registering with the notification service, with the error returned by the push notification service encapsulated as an `AdfException`.

   MAF calls the `onMessage(Event e)` method with the notification payload whenever the application receives a notification. The `Event` object can be used to retrieve useful information about notification payload and the application state. To get the notification payload, use the `Event.getPayload` method. To get the application state at the time of notification, use the `Event.getApplicationState` method. For more information, see the `Event` class in *Java API Reference for Oracle Mobile Application Framework*.

5. Get an `EventSource` object in the `start` method of the ALCL class that represents the source of a native push notification event:

   ```
   EventSource evtSource =
   EventSourceFactory.getEventSource(EventSourceFactory.NATIVE_PUSH_NOTIFICATION_
                                                                   REMOTE_EVENT
                                                                  _SOURCE_NAME);
   ```

6. Create and add an object of the push notification events listener class to the event source:

   ```
   evtSource.addListener(new NativePushNotificationListener());
   ```

A MAF sample application called PushDemo demonstrates how to handle push notifications. It is available from **File** > **New** > **MAF Examples**, and described in Appendix E, "MAF Sample Applications."

24.2.1 What You May Need to Know About the Push Notification Payload

MAF respects the following keys for a JSON-formatted payload:

- `alert`: the text message shown in the notification prompt.
- `sound`: a sound that is played when the notification is received.
- `badge`: the number to badge the application icon on iOS.

> **Note:** On Android, the payload can be a JSON object with key-value pairs. The value is always stringified, because the GCM server converts non-string values to strings before sending them to an application. This is not the case with the APNs, which preserves the value types. For more information, refer to the description of the "data" message parameter in the "Implementing GCM Server" section of *Google Cloud Messaging*. This document is available from the Android Developers website at `http://developer.android.com/index.html` or the Android SDK documentation.

24.3 Managing Local Notifications

You can manage local notifications by using the following:

- Java APIs provided by MAF (see Section 24.3.1, "How to Manage Local Notifications Using Java").
- JavaScript APIs provided by MAF (see Section 24.3.2, "How to Manage Local Notifications Using JavaScript").
- Methods of the DeviceFeatures data control that is available to all MAF applications at the application design time (see Section 24.3.3, "How to Manage Local Notifications Using the DeviceFeatures Data Control").

A MAF sample application called LocalNotification demonstrates how to schedule and handle local notifications. It is available from **File** > **New** > **MAF Examples**, and described in Appendix E, "MAF Sample Applications."

24.3.1 How to Manage Local Notifications Using Java

You can schedule local notifications using the following methods of the `oracle.adfmf.framework.api.AdfmfContainerUtilities` class:

- `addLocalNotification(MafNativeLocalNotificationOptions options)`. This method returns a `String` that represents the ID of the notification being scheduled.

  In your Java code, you use this method in a manner similar to the following:

```
try {
  // Configure local notification
  MafNativeLocalNotificationOptions options =
               new MafNativeLocalNotificationOptions();

  options.setTitle("some title text");
  options.setAlert("some alert text");

  // Set the date in UTC
  options.setDate(LocalDateTime.now(Clock.systemUTC()).plusSeconds(5));
  // Set the notification to repeat every minute
  options.setRepeat(MafNativeLocalNotificationOptions.RepeatInterval.MINUTELY);
```

```
  // Set the application badge to be '1' everytime notification is triggered
  options.setBadge(1);
  // Play the default system sound when notification triggers
  options.setSound(MafNativeLocalNotificationOptions.SYSTEM_DEFAULT_SOUND);
  // Vibrate using default system vibration motion when notification triggers
  options.setVibration(
            MafNativeLocalNotificationOptions.SYSTEM_DEFAULT_VIBRATION);

  // Add custom payload that is to be delivered through the local notification
  HashMap<String,Object> payload = new HashMap<String, Object>();

  payload.put("somenumber", 1);
  payload.put("somestring", "value2");
  payload.put("someboolean", true);
  options.setPayload(payload);

  // Schedule local notification
  String notificationID = AdfmfContainerUtilities.
                          addLocalNotification(options);
  System.out.println("Notification added successfully. ID is "+notificationID);
}
catch(Exception e) {
  System.err.println("There was a problem adding notification");
}
```

The notification options' impact on the behavior of your application depends on your target platform. For more information, see Section 24.3.5, "What You May Need to Know About Local Notification Options and the Application Behavior."

- `cancelLocalNotification(String notificationId)`. This method returns a `String` that represents the ID of the successfully canceled notification.

  In your Java code, you use this method in a manner similar to the following:

```
try {
  String cancelledNotificationId = AdfmfContainerUtilities.
         cancelLocalNotification("a83b696d-53e7-4242-ab4d-4a771d8d178f");
  System.out.println("Notification successfully canceled");
}
catch(AdfException e) {
  System.err.println("There was a problem canceling notification");
}
```

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

24.3.2 How to Manage Local Notifications Using JavaScript

MAF allows you to manage local notifications using JavaScript APIs in the `adf.mf.api.localnotification` namespace. The following methods are available:

- `add`, defined as follows:

```
/**
* Schedule a local notification
*
* @param {Object} options - notification options
* @param {string} options.title - notification title
* @param {string} options.alert - notification alert
* @param {Date} options.date - date at which notification is to be triggered
* @param {Number} options.badge - application icon is to be badged by this
*                                 number when notification is triggered
* @param {string} options.sound - set it to 'SYSTEM_DEFAULT' to play the
```

```
*                                 default system sound upon a notification
* @param {string} options.vibration - set it to 'SYSTEM_DEFAULT' for default
*                                    system vibration upon a notification
* @param {Object} options.payload - custom payload to be sent via notification
* @param {successCallback} scb - success callback
* @param {errorCallback} ecb - error callback
*/
adf.mf.api.localnotification.add(options,scb,ecb);

{Object} options : json representing notification options
        {
           "title" : String,  // notification title (Android only)
           "alert" : String,  // notification alert text
           "date" : Date,  // date-time at which notification
                           // should be fired (UTC time zone)
           "repeat" : String,  // either 'minutely', 'hourly', 'daily',
                               // 'weekly', 'monthly', or 'yearly'
           "badge" : Number,  // badge application icon on iOS with this number
           "sound" : "SYSTEM_DEFAULT",  // if set, the default system
                                        // sound is played
           "vibration" : String,  // if set to "SYSTEM_DEFAULT", the default
                                  // vibration is enabled upon
                                  // an incoming notification (Android only)
           "payload" : Object,  // custom JSON data to be passed
                                // through the notification
        }


/**
* Success Callback
*
* @param {Object} request - request
* @param {Object} response - response
* @param {string} response.id - id of the scheduled notification
*/
function scb(request, response) {}


/**
* Error Callback
*
* @param {Object} request - request
* @param {Object} response - response
*/
 function fcb(request, response) {}
```

The notification options' impact on the behavior of your application depends on your target platform. For more information, see Section 24.3.5, "What You May Need to Know About Local Notification Options and the Application Behavior."

- `cancel`, defined as follows:

```
/**
* Cancel a scheduled local notification
*
* @param {string} notificationId - id of the scheduled notification
*                                  that needs to be canceled
* @param {successCallback} scb - success callback
* @param {errorCallback} ecb - error callback
*/
adf.mf.api.localnotification.cancel(notificationId, scb, ecb);
```

```
{var} notificationId : id of notification that is to be canceled.


/**
* Success Callback
*
* @callback successCallback
* @param {Object} request - request
* @param {Object} response - response
* @param {string} response.id - id of the notification
*/


/**
* Error Callback
*
* @callback errorCallback
* @param {Object} request - request
* @param {Object} response - response
*/
```

For more information, see *Oracle Fusion Middleware JSDoc Reference for Oracle Mobile Application Framework*.

24.3.3 How to Manage Local Notifications Using the DeviceFeatures Data Control

You can schedule and cancel a local notification using the `addLocalNotification` and `cancelLocalNotification` methods of the DeviceFeatures data control.

For information about the DeviceFeatures data control, see Section 13.10, "Using the DeviceFeatures Data Control."

24.3.4 How to Handle Local Notifications

To enable handling of local notifications, MAF provides the following:

- The `EventListener` interface that you must implement to create a listener for local notification events. When a notification is triggered, the `onMessage` method is called with the notification details:

```
NativeLocalNotificationListener implements EventListener {
   @Override
   public void onOpen(String id) {
   }

   @Override
   public void onMessage(Event event) {
         //Application state at the time of this notification
         int appState = event.getApplicationState();

         //Get local notification event details
         if (event instanceof NativeLocalNotificationEvent) {
         NativeLocalNotificationEvent localNotificationEvent =
                                    (NativeLocalNotificationEvent) event;
         HashMap<String, Object> notification =
                                    localNotificationEvent.getPayloadObject();

         // do something with the notification payload, such as navigate
         // to an application feature, call a web service, and so on
```

```
    }

    @Override
    public void onError(AdfException error) {
    }
}
```

- The `NativeLocalNotificationEvent` class that extends the `oracle.adfmf.framework.event.Event`.

To receive events related to local notifications, you need to add your listener in the registered `ApplicationLifeCycleEventListener#start` method as follows:

```
EventSource evtSource = EventSourceFactory.getEventSource(
            EventSourceFactory.NATIVE_LOCAL_NOTIFICATION_EVENT_SOURCE_NAME);
evtSource.addListener(new NativeLocalNotificationListener());
```

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

24.3.5 What You May Need to Know About Local Notification Options and the Application Behavior

Table 24–2 lists the local notification options and describes how setting certain values or failing to set values for each option affects the notification behavior of a MAF application.

Table 24–2 Local Notification Options

Option	Value	Behavior on iOS	Behavior on Android
`title`	Either: ■ `null` ■ not specified	NA [1]	The notification title in the notification center appears blank.
`alert`	Either: ■ `null` ■ not specified	If the application is either running in the background or not running at all, the notification is not displayed nor queued in the notification center. If the application is running in the foreground, the notification details are passed directly to the application's local notification listener.	The notification is displayed as a banner with a title but without the alert text.
`date`	Either: ■ `null` ■ not specified ■ time or date in the past	The notification is triggered immediately.	The notification is triggered immediately.
`repeat`	Either: ■ `null` ■ not specified	The notification does not repeat.	The notification does not repeat.

Table 24–2 (Cont.) Local Notification Options

Option	Value	Behavior on iOS	Behavior on Android
`badge`	Either: ■ `null` ■ not specified ■ negative number	The notification does not badge the application icon. Any existing badge is maintained.	NA [2]
`badge`	0	Any existing badge is removed from the application icon.	NA [3]
`sound`	Either: ■ Other than `SYSTEM_DEFAULT_SOUND` ■ not specified	The notification does not play sound.	The notification does not play sound.
`vibration`	Other than `SYSTEM_DEFAULT_VIBRATION`	NA [4]	The notification does not trigger vibration of a mobile device.

[1] The notification is not affected because its title is always the application name.

[2] There is no concept of application badging. The setting is ignored.

[3] There is no concept of application badging. The setting is ignored.

[4] You cannot control vibration. The setting is ignored. However, if you specify that the default system sound should be played upon receipt of a notification and if the end user enables the Vibrate on Ring setting on the mobile device, then the device will also vibrate when the notification is received.

25

Caching Data in a MAF Application

This chapter describes how to cache data in MAF applications.

This chapter includes the following section:

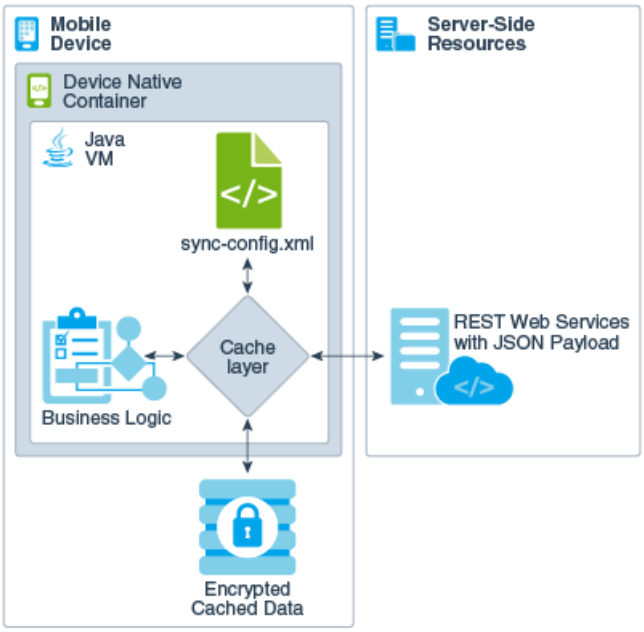
- Section 25.1, "Introduction to Data Caching in MAF Applications"
- Section 25.2, "Enable Data Caching in a MAF Application"
- Section 25.3, "Specifying Cached Resources and Cache Policies in the sync-config.xml File"
- Section 25.4, "Caching Policies Provided by MAF"
- Section 25.5, "Using Configuration Service End Points in the sync-config.xml File"
- Section 25.6, "Encrypting Cached Data in a MAF Application"
- Section 25.7, "Packaging the sync-config.xml File in a FAR"

25.1 Introduction to Data Caching in MAF Applications

Data caching plays a key part providing a positive user experience to the users of mobile applications. Users expect mobile applications to be available and to work, at all times. Unfortunately, mobile application connectivity can be unreliable, as network connectivity may not be available, or it may come and go as connections are established and subsequently dropped. MAF provides a number of capabilities to enable you to develop MAF applications that continue to work even when the end user's device is offline. When network connectivity is unavailable, your MAF application can access locally stored cached data to ensure a seamless user experience.

Figure 25–1 shows how caching works in MAF. The cache layer intercepts REST calls that originate from the Business Logic layer. The cache layer reads the `sync-config.xml` file, and based on the file's entries, stores responses from the REST service in the cached data store. The `sync-config.xml` file is where you specify URI (end point) resources to cache and the cache policies for the URI. The cache layer uses the URI that you specify as the key to identify a specific resource in the cache.

Figure 25–1 Caching Data in a MAF Application



MAF provides a set of policies that you can apply and configure to determine the refresh and expiration frequency of the data in the cache. These policies tell the cache layer how to process requests and, as a result, enable you to improve application performance because it is quicker for applications to use an offline read to fetch data stored on a device than it is to retrieve data from a server. These policies also enable the application caching to maintain an optimal user experience by enabling offline reads when an internet connection becomes unavailable.

Implementing the functionality described in your MAF application requires you to:

- Section 25.2, "Enable Data Caching in a MAF Application"
- Section 25.3, "Specifying Cached Resources and Cache Policies in the sync-config.xml File"
- Section 25.6, "Encrypting Cached Data in a MAF Application"

Note: MAF applications only support the caching of data retrieved by REST services with JSON payloads.

25.2 Enable Data Caching in a MAF Application

To enable data caching, uncomment the following line in the `maf.properties` file (located at *`assembly_project`*`/META-INF`):

```
#java.commandline.argument=-DsyncEnabled=true
```

Note: When you enable data caching, any data that passes over the network will be cached. Once your application is deployed with caching enabled, there is no way to turn it off at runtime.

For information about configuring the resources to cache and the cache policies to use, see Section 25.3, "Specifying Cached Resources and Cache Policies in the sync-config.xml File."

25.3 Specifying Cached Resources and Cache Policies in the sync-config.xml File

OEPE creates the `sync-config.xml` file in the `/adf/META-INF` directory of the ssembly project by default when you create a MAF application. Use this file to specify caching policies to apply when your MAF application makes calls to end points (URIs) that you specify. By default, the `sync-config.xml` file includes a default policy that applies when you enable caching. This default caching policy applies to all end points other than those that you specify in the sync-config.xml file.

Example 25–1 demonstrates how you define two separate policies (`policy1` and `policy2`) for two separate end points (URIs) that originate from the same server. The example also shows a default policy that applies when the MAF application makes REST calls to all other URIs other than those specified for `policy1` and `policy2`. Finally, the example shows how you enables encryption.

You can view another example of a caching policy in use in the WorkBetter sample application's `sync-config.xml` file. For more information about the sample applications, see Appendix E, "MAF Sample Applications".

In Example 25–1, `TaskConn` in the `BaseURI` element of the `sync-config.xml` is a reference to a connection that is defined in the MAF application's `connections.xml` file, as follows:

```
<Reference name="TaskConn"
className="oracle.adf.model.connection.url.HttpURLConnection" xmlns="">
    <Factory className="oracle.adf.model.connection.url.URLConnectionFactory"/>
    <RefAddresses>
      <XmlRefAddr addrType="TaskConn">
<Contents>
  <urlconnection name="TaskConn" url="http://localhost:7101/TaskService/rest/v1"/>
</Contents>
      </XmlRefAddr>
    </RefAddresses>
  </Reference>
```

The benefit of this configuration is that, if connection details change (for example, a change in host name), you only need to make a change in the `connections.xml` file. No change will be required to the `sync-config.xml` file.

Example 25–1 Sample sync-config.xml File Defining Two Policies

```
<Settings xmlns="http://xmlns.oracle.com/sync/config">
   <!-- The connection.xml file defines the URL that the value of the BaseURI
element references. -->
   <BaseUri>TaskConn</BaseUri>

   ...
   <EnableEncryption>true</EnableEncryption>
   <Policies>
<ServerGroup id="tasklist" baseUri="TaskConn">
   <Policy id="policy1">
<Path>/taskservice1/rest/v1/TaskList/*</Path>

 <!-- This caching policy applies to a REST service accessed by a call to an end
```

```
point constructed from
      a concatenation of the baseURI value for TaskConn and the <Path> element's
value. That is,
      http://localhost:7101/TaskService/rest/v1/taskservice1/rest/v1/TaskList/*

      When the end point above is used by the business logic layer in the MAF
application, the cache layer checks sync-config.xml
        to see if there is a cache policy defined for the end point. If it finds
the policy, it applies the policy. -->

<FetchPolicy>FETCH_FROM_SERVICE_ON_CACHE_MISS_OR_EXPIRY</FetchPolicy>
<UpdatePolicy>QUEUE_IF_OFFLINE</UpdatePolicy>
<ExpirationPolicy>EXPIRE_AFTER</ExpirationPolicy>
<ExpireAfter>30</ExpireAfter>
<EvictionPolicy>EVICT_ON_EXPIRY_AT_STARTUP</EvictionPolicy>
   </Policy>
</ServerGroup>

<ServerGroup id="taskdetail" baseUri="TaskConn">
    <Policy id="policy2">
<Path>/taskservice1/rest/v1/TaskDetail/*</Path>

 <!-- This caching policy applies to a different REST service accessed by a call
to an end point constructed from
      a concatenation of the baseURI value for TaskConn and the <Path> element's
value. That is,
      http://localhost:7101/TaskService/rest/v1/taskservice1/rest/v1/TaskDetail/*

      When the end point above is used by the business logic layer in the MAF app,
the cache layer checks sync-config.xml
      to see if there is a cache policy defined for the end point. If it finds the
policy, it applies the policy. -->

   <FetchPolicy>FETCH_FROM_SERVICE_IF_ONLINE</FetchPolicy>
   <UpdatePolicy>UPDATE_IF_ONLINE</UpdatePolicy>
   <ExpirationPolicy>EXPIRE_AFTER</ExpirationPolicy>
   <ExpireAfter>30</ExpireAfter>
     <EvictionPolicy>EVICT_ON_EXPIRY_AT_STARTUP</EvictionPolicy>
   </Policy>
</ServerGroup>

  <DefaultPolicy>
<FetchPolicy>FETCH_FROM_SERVICE_IF_ONLINE</FetchPolicy>
<UpdatePolicy>UPDATE_IF_ONLINE</UpdatePolicy>
<ExpirationPolicy>NEVER_EXPIRE</ExpirationPolicy>
<EvictionPolicy>MANUAL_EVICTION</EvictionPolicy>
   </DefaultPolicy>
   </Policies>
</Settings>
```

25.4 Caching Policies Provided by MAF

You define caching policies for a mobile application in the `sync-config.xml` file. Combine policies to provide appropriate content for any mobile application.

For information about configuring data storage policy settings, see Section 25.3, "Specifying Cached Resources and Cache Policies in the sync-config.xml File.".

The policies that enable you to configure your application's caching behavior fall under the following groups:

- **Fetch Policies**—Tells the cache layer to fetch resources (from server, from local cache, a combination of the two, and so on). See Table 25–1 for a list of these policies.
- **Expiration Policies**—Sets the time period (in seconds) after which cache layer notes the resources stored in the local cache as out-dated or stale and therefore requiring either updating per the update policies (described in Table 25–4) or deletion from the local cache per the eviction policies (see Table 25–3). For more information on the expiration policies, see Table 25–2.
- **Eviction Policies**—Designates when the cache layer deletes resources stored in the local cache. The eviction policies apply only to the data in the local cache, not to server-side resources. See Table 25–3 for a list of policies.
- **Update Policies**—Defines when expired resources stored in the local cache will be updated. See Table 25–4 for a list of policies.

Table 25–1 Fetch Policies

Policy	Description
Fetch from Cache	Instructs the cache layer to fetch the data from cache only, not from the server. If the cache layer can't find the data in the cache, it returns an error or a null object. Because the cache layer retrieves data directly from the cache when it applies this policy, it can carry out this policy when the client application is online or offline.
Fetch from Service	Instructs the cache layer to fetch the data directly from the server only, not from the cache. The cache layer can only apply this policy when the client application is online. Otherwise, it returns an error or a null object to the client application.
Fetch from Service, if Online	Instructs the cache layer to fetch the data from the server when the client application is online, or to fetch data from the cache when the application is offline.
Fetch from Service on Cache-Miss	Instructs the cache layer to fetch data from the cache. If it can't find the requested data, the cache layer fetches it from the server.
Fetch from Service on Cache-Miss or Expiry	Instructs the cache layer to fetch data in the cache if it exists in the cache and is not stale (expired). Otherwise, the cache layer fetches the request data from the server.
Fetch from Cache, Schedule Refresh	Instructs the cache layer to fetch data from the cache and schedule a background refresh to update cache from the server's latest copy. If there's a cache-miss (meaning that the cache doesn't have the requested data), the cache layer returns a null object to the client application.
Fetch with Refresh	Instructs the cache layer to: ■ Fetch data from the cache if the requested data exists and has not expired. ■ Schedule a background refresh to update the cache from the server's latest copy. If there's a cache-miss (meaning that the cache doesn't have the requested data) or if the data has expired, the cache layer fetches the data directly from the server as it does for the Fetch from Service policy.

Table 25–2 Expiration Policies

Policy	Description
Expire on Restart	Instructs the cache layer to note the data for any URI as expired when the client application restarts, or to update the local data with the server copy the next time it's called by the client application.
Expire After	Instructs the cache layer to expire the data after a specified time (in seconds) that is set in the Expire After Duration policy. Use this policy for data that refreshed on a regular basis.
Expire After Duration	The number of seconds after which the cache layer notes that the data in the cache has expired.
Never Expire	Instructs the cache layer that it can't designate the local data as expired.

Table 25–3 Eviction Policies

Policy	Description
Evict on Expiry at Startup	Instructs the cache layer to delete the expired data from cache when the client application restarts, or to update the local data with the server copy the next time it's called by the client application.
Manual Eviction	The cache layer can't remove data from the local cache automatically. To evict data manually, use an API.

Table 25–4 Update Policies

Policy	Description
Update if Online	Instructs the cache layer to update the local cache with server-side data only when the client application is online. Otherwise, the cache layer returns an error.

25.5 Using Configuration Service End Points in the sync-config.xml File

The `BaseUri` element and `baseUri` attribute on the `ServerGroup` element in `sync-config.xml` can refer to end points defined in `connections.xml`. To take advantage of this functionality, replace the values to point to a valid connection reference rather than a URL as follows:

```
baseUri="<connection_reference_name_in_connections_xml>"
```

If an end point is changed at runtime using connection overrides, the cache policies remain the same for the new URL. For more information, see Chapter 14, "Configuring End Points Used in MAF Applications." The WorkBetter sample application demonstrates an implementation of this configuration. For information about how to access this and other sample applications, see Appendix E, "MAF Sample Applications."

25.6 Encrypting Cached Data in a MAF Application

MAF provides you with the ability to encrypt data that the MAF application caches.

Once enabled, all data cached by the MAF application is encrypted. By default, it is disabled. You configure the sync-config.xml file to enable encryption, as shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<Settings xmlns="http://xmlns.oracle.com/sync/config">
  <BaseUri>TaskConn</BaseUri>
  ...
  <EnableEncryption>true</EnableEncryption>
  <Policies>
  ...
</Settings>
```

25.7 Packaging the sync-config.xml File in a FAR

The `sync-config.xml` file is included in the Feature Archive file when the view controller project is deployed as a FAR. Like the `connections.xml` file, MAF merges the contents of the `sync-config.xml` file in the FAR (`jar-sync-config.xml`) with those of the consuming application's `sync-config.xml` file after you add the FAR to the application. Because the `sync-config.xml` file describes the web service endpoints used by the mobile application, you can update the endpoints for all of the web services used by the application features that comprise a mobile application by adding a FAR as described in Section 8.2, "Using FAR Content in a MAF Application."

After you add the FAR to the application, MAF logs messages that prompt you to verify and, if needed, modify the application's `sync-config.xml` and `connections.xml` files. These messages reflect the state of the `sync-config.xml` file in the consuming application.

If the consuming application lacks the `sync-config.xml` file, then MAF adds the file to the application and writes a message similar to the following:

```
oracle.adfmf.framework.dt.deploy.features.deployers.SyncConfigMerger _logNoSyncConfigInAppUsingFar
WARNING: The application does not contain a synchronization file, "sync-config.xml". Creating one
containing the synchronization configuration in the Feaure Archive.
```

MAF writes a log message similar to the following that requests that you verify (or create) a connection if the `sync-config.xml` file's `<ServerGroup>` elements do not have corresponding `<Reference>` elements defined in the consuming application's `connections.xml` file:

```
oracle.adfmf.framework.dt.deploy.features.deployers.SyncConfigMerger _logAddedServerGroups
WARNING: The following server groups were added sync-config.xml by the Add to Application
operation:
{
  ServerGroup1 - there is no existing application connection defined for this server group.
Please create the connection.

  ServerGroup2 - verify its configuration.
}
```

If the `<ServerGroup>` definitions in the consuming application's `sync-config.xml` file duplicate those of the counterpart `sync-config.xml` file included in the FAR, then MAF writes the following `SEVERE`-level message to the log:

```
oracle.adfmf.framework.dt.deploy.features.deployers.SyncConfigMerger _logDuplicateServerGroups
SEVERE: Cannot merge the server groups from the Feature Archive because the following definitions
already exist:
ServerGroup1
ServerGroup2
```

26

Displaying Error Messages in MAF Applications

This chapter describes how to use the `AdfException` class to handle errors and how to localize error messages.

This chapter includes the following sections:

- Section 26.1, "Introduction to Error Handling in MAF Applications"
- Section 26.3, "Localizing Error Messages"
- Section 26.2, "Displaying Error Messages and Stopping Background Threads"

26.1 Introduction to Error Handling in MAF Applications

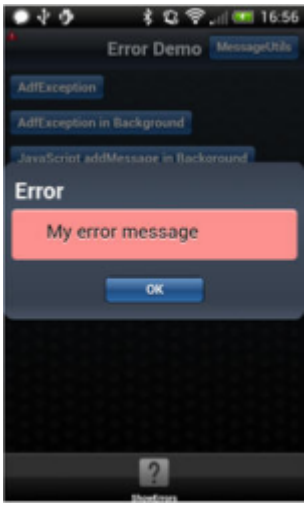
Errors arising from mobile applications might be unexpected, such as a failed connection to a remote server, or expected, such as a violation of an application business rule. Errors or exceptions might occur in the primary request thread or in a secondary thread that runs a background task. If the application supports multiple languages, then it must display the error message in the user's language.

To enable a mobile application to throw an exception, use `oracle.adfmf.framework.exception.AdfException` class. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

The following code enables MAF to handle an exception gracefully. A popup message, similar to the one illustrated in Figure 26–1 displays within the application and shows the message severity and explanatory text.

```
throw new AdfException("My error message",AdfException.ERROR);
```

Figure 26–1 An Error Message



Note: Similar error messages display within application when the exception is thrown within a managed bean or a data control bean.

26.2 Displaying Error Messages and Stopping Background Threads

The `MessageUtils` class, illustrated in the example below, enables an application to stop a thread and display an error by first making a JavaScript call (`invokeContainerJavaScriptFunction`) and then throwing an exception. The `addMessage` method enables the error to display. For more information, see Section 26.2.1, "How Applications Display Error Message for Background Thread Exceptions." See also Section B.2.19, "invokeContainerJavaScriptFunction."

The `MessageUtils` class uses the `BundleFactory` and `Utility` methods for retrieving the resource bundle and the error message and dynamically checks if a thread is running in the background. Using this class, you can move code from the main thread to the background thread.

```
package oracle.errorhandling.demo.mobile;

import java.util.ResourceBundle;

import oracle.adfmf.framework.api.AdfmfContainerUtilities;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.framework.exception.AdfException;
import oracle.adfmf.util.BundleFactory;
import oracle.adfmf.util.Utility;

public class MessageUtils{  public static void handleError(AdfException ex)  {
handleMessage(ex.getSeverity(), ex.getMessage());  }  public static void
handleError(String message)  {    handleMessage(AdfException.ERROR, message);  }
public static void handleError(Exception ex)  {
handleMessage(AdfException.ERROR, ex.getLocalizedMessage());  }  public static
void handleMessage(String severity, String message)  {    if
(AdfmfJavaUtilities.isBackgroundThread())    {
AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.
                                                          getFeatureName(),
                                                          "adf.mf.api.amx.ad
dMessage",
```

```
                                                                    new Object[]
{severity,
                                                                                mes
sage,
                                                                                nul
l,
                                                                                nul
l});

      if (AdfException.ERROR.equals(severity))      {        // we still need to
throw execption to stop background thread processing        throw new
AdfException(message,severity);             }    }    else    {      throw new
AdfException(message,severity);    }     }}
    public static void addJavaScriptMessage(String severity, String message) {

AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.
                                                                  getFeatureName()
,
                                                                  "adf.mf.api.amx.
addMessage",
                                                                   new Object[]
{severity,
                                                                                m
essage,
                                                                                n
ull,
                                                                                n
ull });


    }
}
```

26.2.1 How Applications Display Error Message for Background Thread Exceptions

Applications do not display error messages when exceptions are thrown for background threads. To enable error messages to display under these circumstances, applications call the `addMessage` method. The `addMessage` method takes the following parameters:

- The severity of the error
- The summary message
- The detail message
- a `clientComponentId`.

The example below illustrates how you can enable the application to alert the user when an error occurs in the background by using the `addMessage` method.

```
Runnable runnable = new Runnable()
{
  public void run()
  {

AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.getFe
atureName(),
      "adf.mf.api.amx.addMessage", new Object[] {AdfException.ERROR,
                                                  "My error message for background
```

```
thread",
                                                    null,
                                                    null });
  }
};
Thread thread = new Thread(runnable);
thread.start();
```

Because the `adf.mf.api.amx.addMessage` JavaScript function is the same method that is used when the application throws `AdfException` in the primary request thread, users receive the same popup error message whether the error message is referring to exceptions in the main thread or from a background thread.

> **Note:** As illustrated in the example above, the detail message and the `clientComponentId` can be a `Null` value. A detail message displays on a new line in the same font size as the summary message.

However, you can prevent an error message from appearing if you place the code within a piece of Java logic that runs in a background thread, as illustrated in the following example. Using the method illustrated in the first example of Section 26.3, "Localizing Error Messages" enables the background thread to stop silently without notifying the user.

```
Runnable runnable = new Runnable() {
   public void run() {
      // this exception will be lost because no popup error
      // message will display in the MAF application
      throw new AdfException("My (lost) error message in background",
                             AdfException.ERROR);
   }
};
Thread thread = new Thread(runnable);
thread.start();
```

26.3 Localizing Error Messages

MAF uses standard Java resource bundles to display an exception error message in the language of the application user. As illustrated in the example below, the resource bundle name (the `.xlf` file) and bundle message key is passed to the `AdfException` constructor method to enable the error message to be read from a resource bundle.

```
private static final String XLF_BUNDLE_
NAME="oracle.errorhandling..mobile.ViewControllerBundle";
   throw new AdfException(AdfException.ERROR, XLF_BUNDLE_NAME,
                                             "MY_ERROR_MESSAGE",
                                              null);
```

To ensure that the application does not throw an `MissingResourceException` error, use the `oracle.adfmf.util.BundleFactory` method to retrieve the resource bundle and then use the `oracle.adfmf.util.Utility` method to retrieve the error message, as illustrated in the example below.

```
ResourceBundle bundle = BundleFactory.getBundle(XLF_BUNDLE_NAME);
String message = Utility.getResourceString(bundle, "MY_ERROR_MESSAGE",null);
throw new AdfException(message,AdfException.ERROR);
```

The next example illustrates using the `adf.mf.api.amx.addMessage` JavaScript function to display the localized error message when an exception is thrown from a background thread.

```
ResourceBundle bundle = BundleFactory.getBundle(XLF_BUNDLE_NAME);
String message = Utility.getResourceString(bundle, "MY_ERROR_MESSAGE_BG",null);
AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.
                                        getFeatureName(),
                                        "adf.mf.api.amx.addMessage",
                                         new Object[] {AdfException.ERROR,
                                                       message,
                                                       null,
                                                       null });
```

27

Deploying MAF Applications

This chapter describes how to deploy mobile applications for testing and for publishing.

This chapter includes the following sections:

- Section 27.1, "Introduction to Deployment of MAF Applications"
- Section 27.2, "Working with Deployment Configurations"
- Section 27.3, "Deploying an Android Application"
- Section 27.4, "Deploying an iOS Application"
- Section 27.5, "Deploying a MAF Application to the Universal Windows Platform"
- Section 27.6, "Deploying Feature Archive Files (FARs)"
- Section 27.7, "Creating a Mobile Application Archive File"
- Section 27.8, "Creating Unsigned Deployment Packages"
- Section 27.9, "Deploying with Oracle Mobile Security Suite"

27.1 Introduction to Deployment of MAF Applications

Before you can publish an application for distribution to end users, you must test it on a device or virtual device to assess its behavior and ease of use. You create deployment configurations which you use to deploy your MAF application to the device or virtual device (emulator or simulator) where you want to test your MAF application.

MAF uses the deployment configuration to execute the deployment of an application by copying a platform-specific template application to a temporary location, updating that application with the code, resources, and configuration defined in the MAF project. MAF then builds and deploys the application using the tools of the target platform. You can deploy a mobile application as the platform-specific package which you can make available from a download site or application marketplace (for example, the Apple App Store). For testing and debugging, you can deploy to a device or virtual device. You can reuse the application features by deploying the view projects as a feature archive (FAR). You also have the option to reuse the entire mobile application by deploying it as a Mobile Application Archive (`.maa`) file.

Each MAF deployment configuration (Android, iOS, Windows) includes a set of different libraries that are specific to the type of deployment (release or debug) in combination with the deployment target (simulators or actual devices). In addition, each set of these libraries includes a JVM JAR file. The application binding layer resides within this virtual machine, which is a collection of Objective-C libraries. For example, MAF deploys a JVM JAR file and a set of libraries for a debug deployment

targeted at an iOS simulator, but deploys a different JVM JAR file and set of libraries to a debug deployment targeted to an actual iOS-powered device. The libraries that you declare for the project are included in the deployment artifacts for the project.

27.2 Working with Deployment Configurations

You create a deployment configuration for the platform you want the application to run on (Android, iOS, Windows). A deployment configuration defines how an application is packaged into the archive that will be deployed to the platform device (for example, an Android-powered device or Android emulators). The deployment configuration does the following:

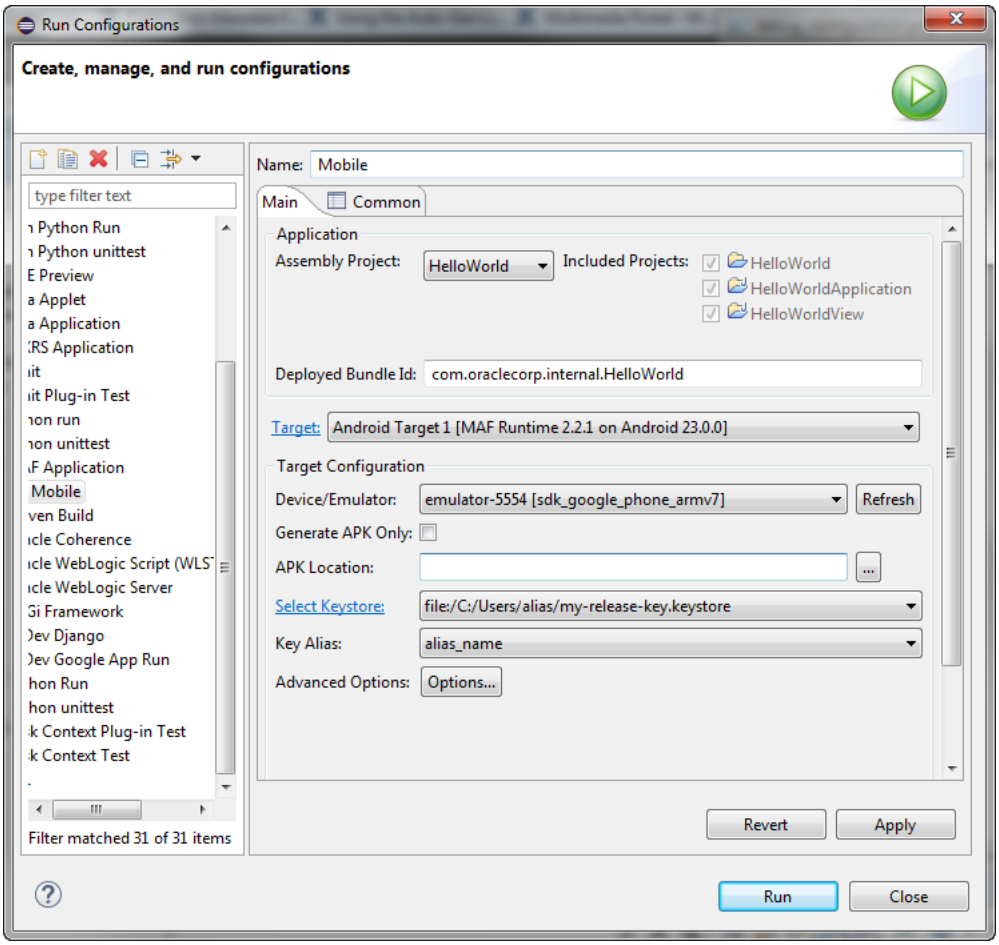
- Select the assembly project to package and deploy.
- Select the target to deploy to. The target is a combination of a platform (for example, Android) and a specific version of the MAF runtime.
- Selects the device profile, that is the emulator or device, to deploy to.
- Selects platform-specific advanced options.
- Specifies the application to deploy.

In addition to the platform-specific deployment configurations, MAF also enables you to package the MAF application as a MAF Application Archive (`.maa`) file. Using this file, you can create a new MAF application using a pre-existing application that has been packaged as an `.maa` file. By default, this deployment file bears the name of the MAF application followed by `_archive`. For more information, see Section 27.7, "Creating a Mobile Application Archive File."

27.2.1 About Deployment Configurations

You can use deployment configurations to deploy a MAF application immediately after creating it using the **Run** button, as shown in Figure 27–1.

Figure 27–1 Creating a Run Configuration



27.2.2 Differences Between Run Configurations and Debug Configurations

The Run Configurations and Debug Configurations are both invoked from the Run menu, and they differ in that Debug Configurations has an additional tab, the Debug tab, where you set debug options. For more information, see Chapter 30, "Testing and Debugging MAF Applications.".

- Debug—Select this option for development builds. Designating a debug build results in the inclusion of debugging symbols.

  During development, you should choose debug in order to use the default keystore.

  > **Note:** Debug is not available for UWP applications.

  At runtime, MAF indicates that an application has been deployed in the debug mode by overlaying a debugging symbol that is represented by an exclamation point within a red triangle.

- Run—Select to compile the build ready for release, with libraries, and so on. release bits and libraries, and so on.

> **Tip:** Use the run mode, where the application is compiled for release, not the debug mode, to test application performance.

27.2.3 How to Create a Deployment Configuration

When you create an application you choose the deployment targets for the application. Later, when you create the deployment configuration you choose the deployment targets to use. You can deploy an application using these configurations, edit them, or construct new ones using the MAF-specific configurations dialog. The Run configurations dialog and Debug configurations dialog enable you to create default deployment configurations. You can create as many deployment configurations as you want.

Before you begin:

To enable OEPE to deploy mobile applications, you must designate the SDKs for the target platforms as described in the "Configuring Mobile Application Framework" section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

To enable OEPE to deploy mobile applications, you must designate the SDKs for the target platforms using the MAF Preferences page.

> **Tip:** For iOS deployments, run iTunes and the iOS Simulator at least once before you configure their directory locations in the MAF Platforms preferences page.

To create a deployment configuration:

1. For development, choose **Run** > **Debug Configurations**.

   For production, choose **Run** > **Run Configurations**
2. Navigate to **MAF Application**, and right-click and choose **New** to create a new configuration (see Figure 27–1).
3. Do the following:
   - Accept the default name for the configuration or enter a new one.
   - Choose the Assembly project.
   - Choose the target from the list of those available and the device or emulator from the list of those available. If necessary, refresh the list.
   - If necessary, select the keystore and key alias for the device you are deploying to.
4. If you want to deploy immediately, click **Run**. To save the configuration click **Apply**. Alternatively click **Close** and in the Save Changes dialog, click **Yes**.

27.2.4 What Happens When You Create a Deployment Configuration

When you deploy an application by clicking **Run** or **Debug**, it triggers the packaging and deployment of the application. OEPE creates a deployment directory and related subdirectory. It also creates Feature Archive files (FARs) for the view projects (which must have different names) and assembly project. In addition to these two FARs, OEPE creates copies of any FARs that were imported into the project. Finally, the package is deployed to the device or emulator.

To clean the deploy artifacts, use the Clean dialog to just one set of projects, or all projects. You can clean:

- Just one set of projects. This is the fastest way to do a clean-rebuild.

  Select **Clean projects selected below** then select the assembly project you want to clean, and click **OK**.

- All projects. This ensures that all user artifacts get rebuilt before being re-staged and deployed.

  Choose **Clean all projects**, and click **OK**.

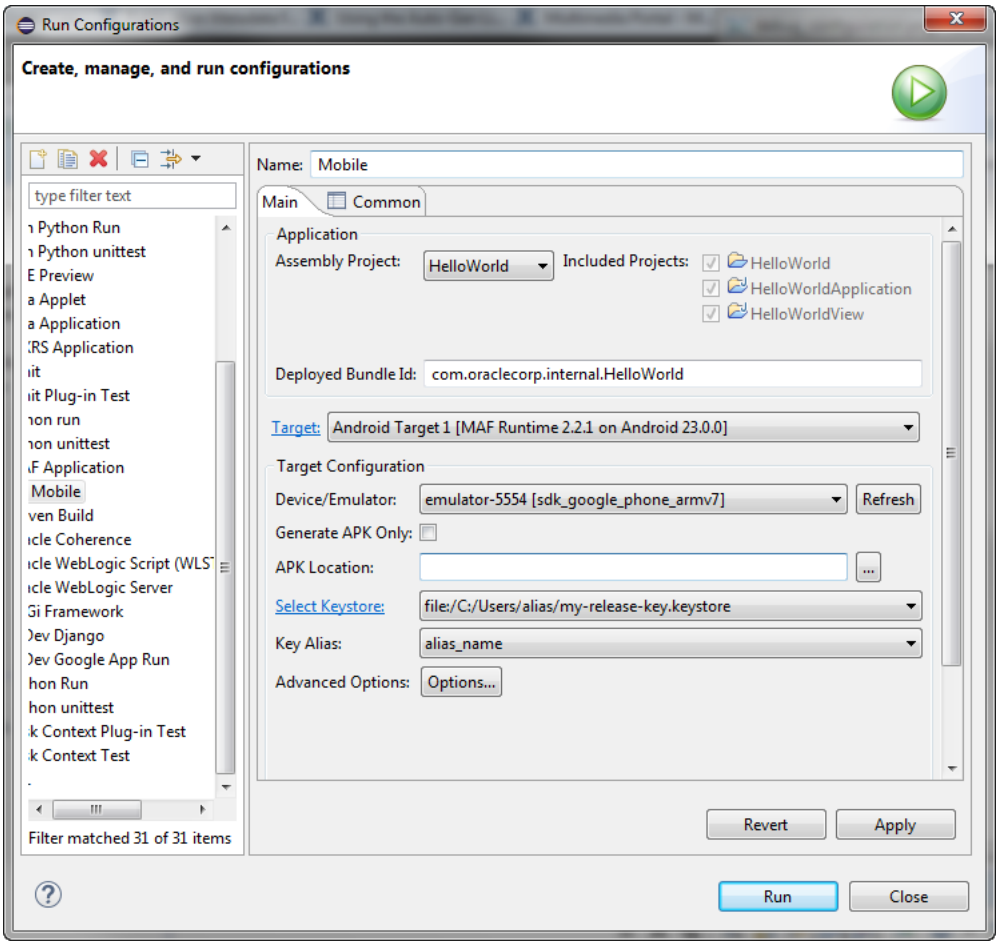
Once you have created a run configuration, you can select it by clicking the green Run arrow from the menu bar. Clicking the arrow will select the last-used run configuration. You can select a different run configuration by clicking the drop-down selector next to the green arrow and choosing the configuration you wish to run from the context menu.

27.3 Deploying an Android Application

After you define the deployment configuration, you can deploy a mobile application to the Android platform from the run configuration dialog or debug configuration dialog, as shown in Figure 27–2.

Using debug configuration dialog, you can deploy the completed application to an Android emulator or to an Android-powered device for testing. After you have tested and debugged the application, you can use the run configuration dialog to bundle the mobile application as an Android application package (`.apk`) file so that it can be published to end users through an application marketplace, such as Google Play.

Figure 27–2 Deployment Configuration Dialog for Android Applications



27.3.1 How to Create an Android Deployment Configuration

To create the deployment configuration for Android, you must define the signing options for the application, the behavior of the `javac` compiler, and if needed, override the default Oracle images used for application icons with custom ones.

Before you begin:

Install and download the Android SDK as described in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

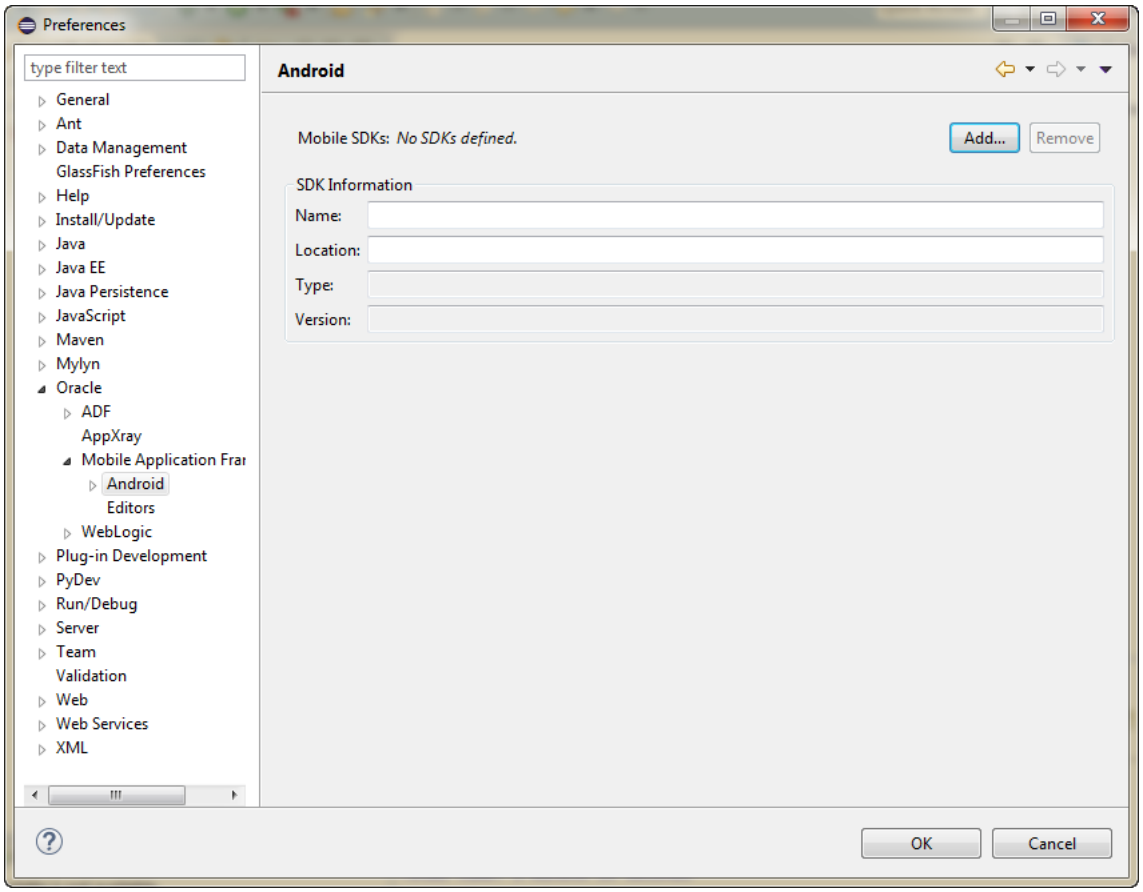
If you deploy to an Android emulator, you must create a virtual device for each emulator instance using the Android Virtual Device Manager, as described in the "Managing Virtual Devices" document, available from the Android Developers website (`http://developer.android.com/tools/devices/index.html`).

You must also set the MAF preferences for the locations for the SDK and platform, which are part of the Android SDK package download

To set the MAF preferences for the Android platform SDKs

1. Open the Preferences dialog from the Window menu (Eclipse menu on MacOS) and navigate to **Oracle** > **Mobile Application Framework** > **Android** to open the Android preferences page, as show in Figure 27–3.

Figure 27–3 Setting Android Preferences



2. Click **Add** to open the Select SDK dialog, and navigate to the location of the Android SDK. Click **OK**.

The SDK information for Android 4.4.2 (API 19), which is illustrated in Figure 27–4, differs from earlier versions.

Note: Push notifications require devices and emulators running Android 2.2 platform (or later). The Google Play store must be installed on these devices. The Google API must be installed in the SDK to enable push notifications on emulators. Users must create a Google account (and be logged in) on devices running platforms earlier than 4.0.3 (API 15).

See also "GCM Architectural Overview" chapter in *Google Cloud Messaging for Android*, available from the Android Developers website (`http://developer.android.com/index.html`) and Section 27.3.1, "How to Create an Android Deployment Configuration."

To create the configuration

1. Select **Run** > **Run Configurations** to open the Configurations dialog.
2. In the Configurations dialog, shown in Figure 27–4, enter a name for the configuration.
3. Choose the assembly Project from the list of those available.

4. If you want to change the application ID, enter a new one in **Deployed Bundle Id**.

   Note: To ensure that an application deploys successfully to an Android-powered device or emulator, the ID must begin with a letter, not with a number or a period. For example, an ID comprised of a wholly numeric value, such as 925090 (*com.company.925090*) will prevent the application from deploying. An ID that begins with letters, such as hello925090 (*com.company.hello925090*) will enable the deployment to succeed.

5. Choose the target, which is combination of the platform version and the MAF runtime version.
6. Choose the target configuration, that is, the device or emulator where the application is to be deployed.

   Note: Sometimes OEPE cannot detect an Android emulator because the daemon associated with the adb server is not running. When the daemon is not running, OEPE displays an error message:

   ```
   Either the target device is not set or the selected device
   is invalid. Click Refresh if your device doesn't show in the
   dropdown. If you do not see your device, run the command
   'adb devices' to check the status of your device.
   ```

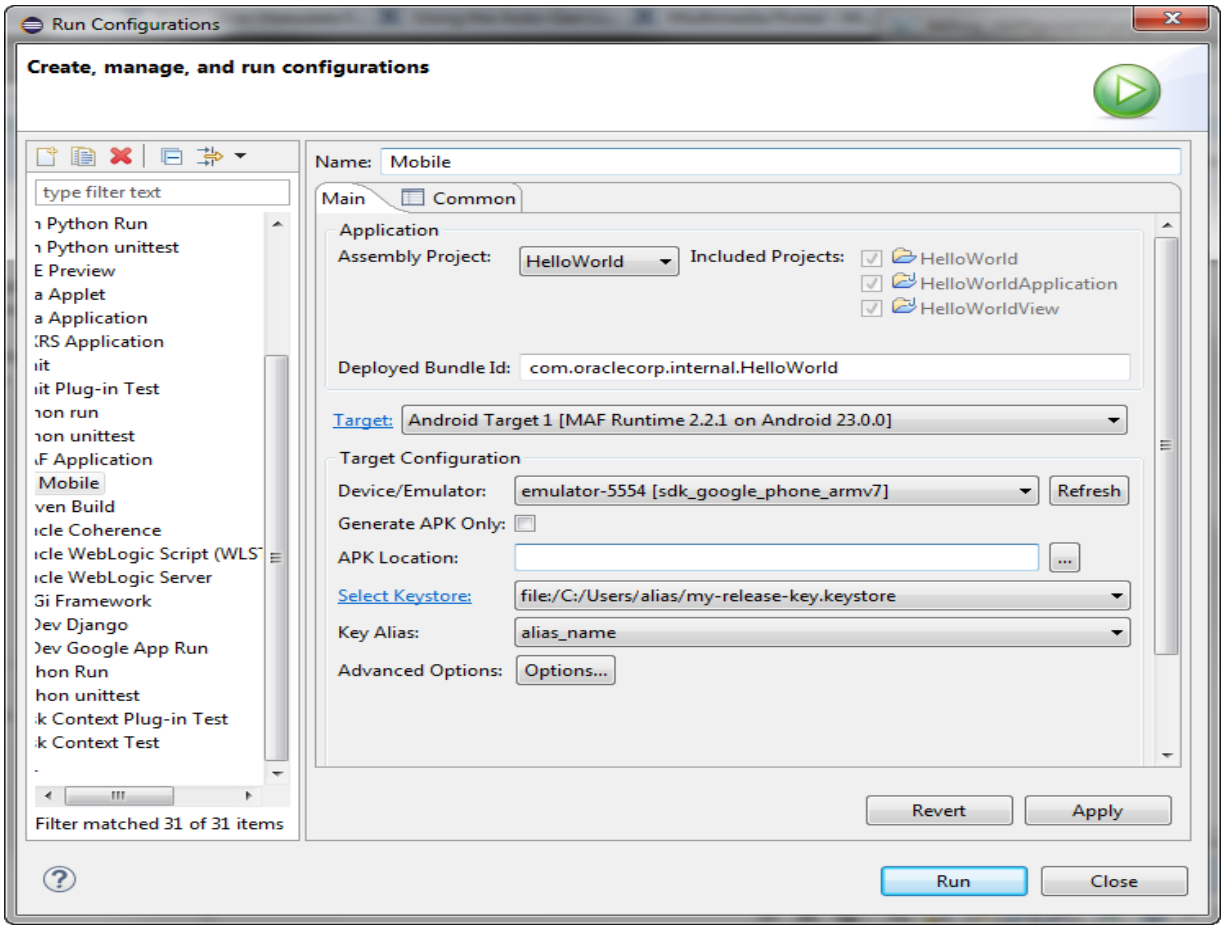
   Running `adb devices` from the SDK/platform-tools directory starts the daemon as below:

   ```
   <sdk-install>\adt-bundle-windows-x86_
   64-20130717\sdk\platform-tools>adb devices
   * daemon not running. starting it now on port 5037 *
   * daemon started successfully *
   List of devices attached
   emulator-5554 device
   ```

   OEPE will then detect the emulator.

7. Enter the keystore. If necessary, click Select Keystore which opens the Android Keystores page of the Preferences dialog where you can add a keystore. For more information, see Section 27.3.1.3, "Defining the Android Signing Options." See also the application publishing information in the "Signing Your Applications" document, available from the Android Developers website (`http://developer.android.com/tools/publishing/app-signing.html`).

Figure 27–4 Setting the Android SDK and Signing Properties



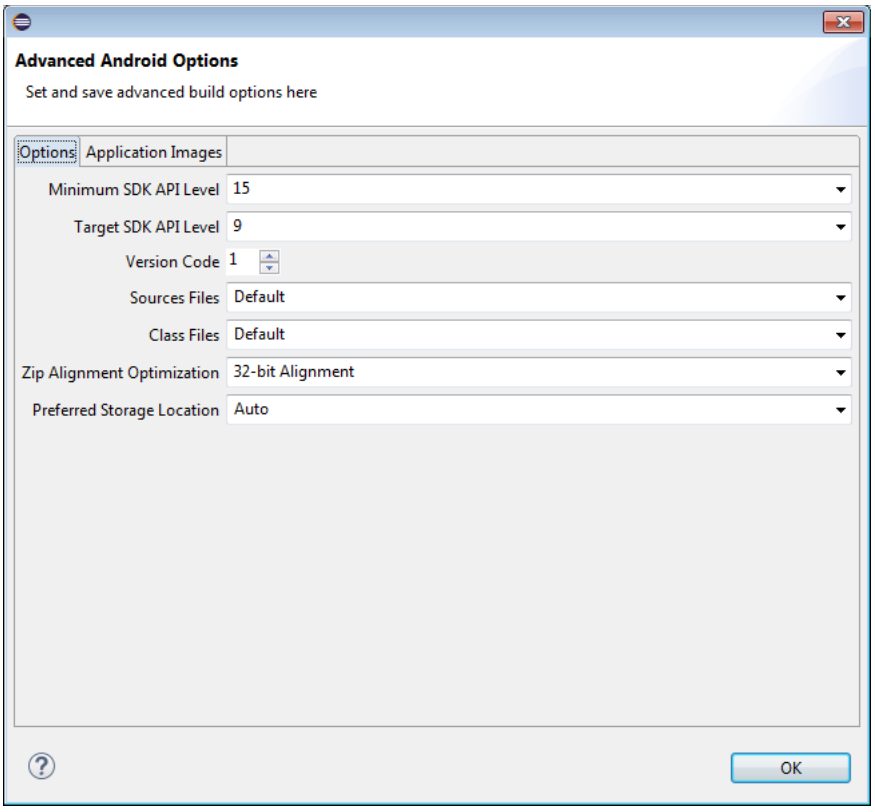
Note: To deploy an application to an Android emulator, you must install API 14 or later (that is, Platform 4.0.*n*)

27.3.1.1 Setting Advanced Options

The Advanced Android Options page, shown in Figure 27–4, enables you to do the following:

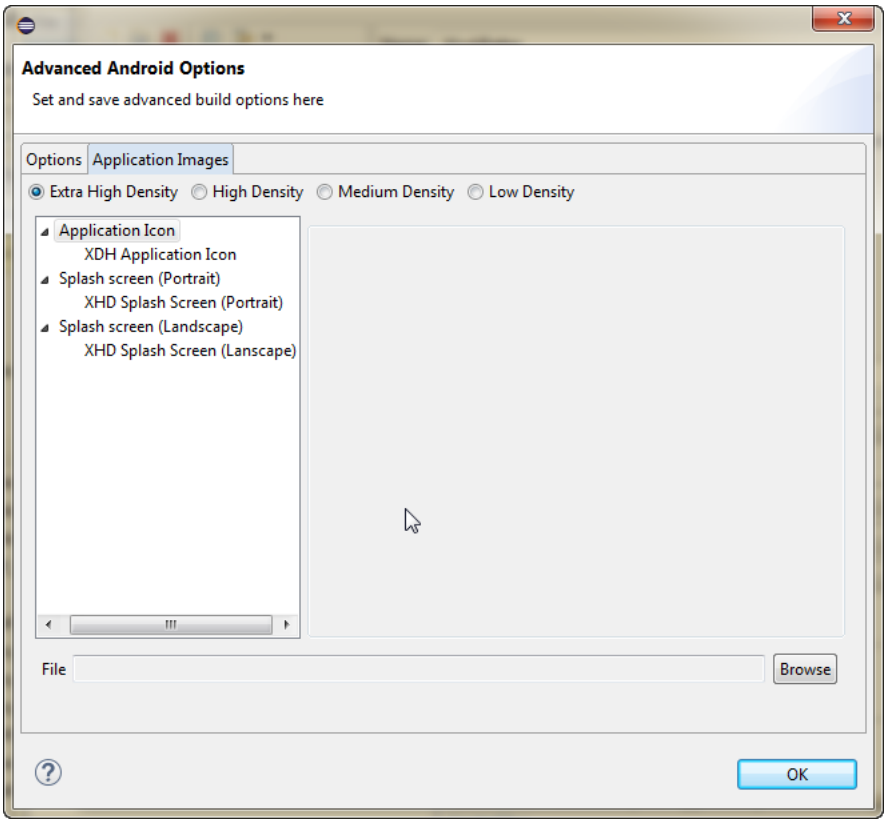
- Set a number of advanced options, including the minimum SDK API level, the target SDK API level, the version code, source files, class files, zip alignment, and preferred storage location, as shown in Figure 27–5

Figure 27–5 Setting SDK Levels



- Change the default application images, as shown in Figure 27–6.

Figure 27–6 Setting Application Images



27.3.1.2 Setting Deployment Options

The Main and Common tabs of the deployment configurations dialog, and the Advanced Android Options dialog, invoked from the Advanced Options button, enable you to set values that are passed in by the `javac` compiler tool options, and also the Android API revisions.

To set the JDK-Compatibility level for the `R.java` and `.class` files:

1. In the deployment configuration dialog, click *Options* to open the Advanced Options dialog.
2. Select the minimum API Level on which the application is able to run from the **Minimum SDK API Level** dropdown list. The minimum and default value is 15, which corresponds to Android 4.0.3 platform.
3. Select the intended API Level on which the application is designed to run from the **Target SDK API Level** dropdown list. The minimum (and default) value is API Level 9, which corresponds to the Android 2.3.*n* platform. For more information, refer to the description of the `<uses-sdk>` attribute in the document entitled "The AndroidManifest.xml File," available through the Android Developers website (`http://developer.android.com/guide/topics/manifest/manifest-intro.html`).

27.3.1.3 Defining the Android Signing Options

An application must be signed before it can be deployed to an Android device or emulator. Android does not require a certificate authority; an application can instead be self-signed.

Before you begin:

OEPE creates a keystore if one does not exist. In addition, you can create one using the keytool utility, as illustrated in the following example.

An application must be signed before it can be deployed to an Android device or emulator. Android does not require a certificate authority; an application can instead be self-signed.

Defining how the deployment signs a mobile application is a two-step process: within the MAF Platforms preference page, you first define release properties for a key that is used to sign Android applications. You only need to configure the release signing properties once. After you define these options, you configure the deployment configuration to designate if the application should be deployed in the release mode.

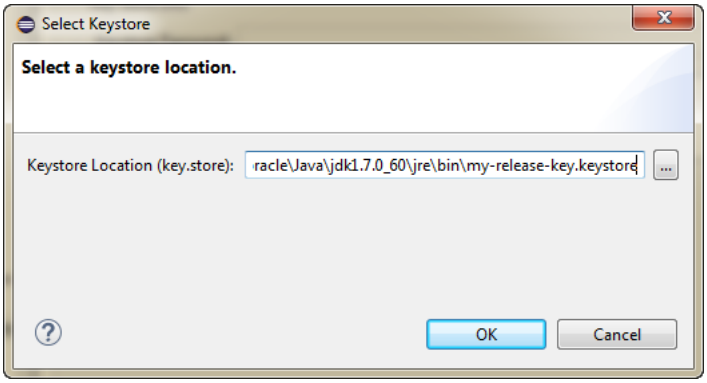
```
keytool -genkey
        -v
        -keystore c:\oepe\workspace\releasesigning.keystore
        -alias releaseKeyAlias
        -keyalg RSA
        -keysize 2048
        -validity 10000
```

In this example, the keystore contains a single key, valid for 10,000 days. As described in the "Signing Your Applications" document, available from the Android Developers website (http://developer.android.com/tools/publishing/app-signing.html), the keytool prompts you to provide passwords for the keystore and key, and to provide the Distinguished Name fields for your key before it generates the keystore. Refer to Java SE Technical Documentation (https://download.oracle.com/javase/index.html) for information on how to use the keytool utility.

To configure the key options:

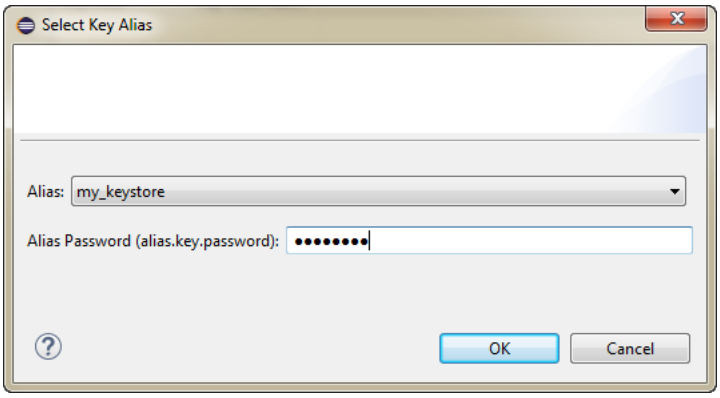
1. Choose **Window** > **Preferences** > **Oracle** > **Mobile Application Framework** > **Android Keystore**.
2. Click and in the Select Keystore dialog, navigate to the location of the keystore, as shown in Figure 27–7 and click **OK**.

Figure 27–7 Selecting the Keystore Location



3. Enter the password for the keystore. When you enter a correct password, the **Add** button is enabled.
4. Click **Add** to open the Select Key Alias dialog, as shown in Figure 27–8.

Figure 27–8 Selecting the Key Alias



5. Enter the password for the key alias and click **OK**.

   The keystore and key alias are now available to be used in the deployment configuration dialog.

To set the Android build mode:

1. Do one of the following:

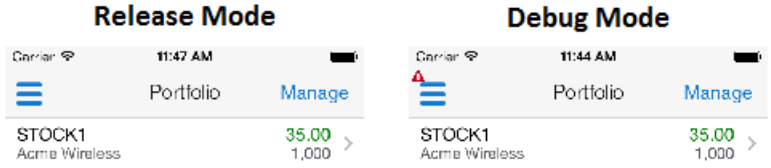
   - From the Run menu, select Debug Configurations. Select **Debug** for developing and testing an application (such as Java and JavaScript debugging). This option enables you to deploy an application on the Android platform without having to provide a private key. Use this option when deploying an application to an Android emulator or to an Android-powered device for testing. See also Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript."

     Note: You cannot publish an application signed with the debug keystore and key; this keystore and key are used for testing purposes only and cannot be used to publish an application to end users.

   - From the Run menu, select Run Configurations. When the application is ready to be published, select **Release**. Use this option when the application is ready to be published to an application marketplace, such as Google Play.

2. In the deployment configuration dialog, after the `.apk` file is signed in either debug or release mode, you can deploy it to a device or to an emulator. At runtime, MAF indicates that an application has been deployed in debug mode by overlaying a debugging symbol that is represented by an exclamation point within a red triangle, as shown in Figure 27–9.

Figure 27–9 Deployment Modes



27.3.1.4 What You May Need to Know About Credential Storage

Credentials for keystore access are stored in Eclipse Secure Storage, which stores data outside the workspace. The location is usually in your home directory, although the specific location varies depending on the OS platform.

To find the Secure Storage Location

1. Open the Preferences dialog from the Window menu (Eclipse menu on MacOS) and navigate to **General** > **Security** > **Secure Storage** to display the Secure Storage page.
2. Click the Contents tab. The storage location is displayed at the bottom of the dialog, for example, *user-home*`/.eclipse/org.elipse.equinox.security/secure_storage`.
3. You can see the passwords that are stored by expanding nodes **oracle > eclipse > tools > oepe > maf > android > keystores**.

   There is a node for each keystore that is declared in the **Oracle > Mobile Application Framework > Android > Android Keystores** page of the Preferences dialog.

   Click on the node to show the data (the password is obfuscated).

> **Note:** Because iOS keys are Mac-specific, there is no need to store them in Eclipse. The iOS XcodeBuild will extract them directly from the Mac Keychain based on the mobileprovision that is specified at launch (configured from the **Oracle >Mobile Application Framework > Android > Android Keystores** page of the Preferences dialog).

27.3.1.5 How to Add a Custom Image to an Android Application

Enabling MAF application icons to display properly on Android-powered devices of different sizes and resolutions requires low-, medium-, and high-density versions of the same images. MAF provides default Oracle images that fulfill these display requirements. However, if the application requires custom icons, you can use the Application Images page, shown in Figure 27–10, to override default images by selecting PNG-formatted images for the application icon and for the splash screen. For the latter, you can add portrait and landscape images. If you do not add a custom image file, then the default Oracle icon is used instead. To create custom images, refer to the "Iconography" document, available from the Android Developers website (`http://developer.android.com/design/style/iconography.html`).

Figure 27–10 Setting Custom Images for an Android Application

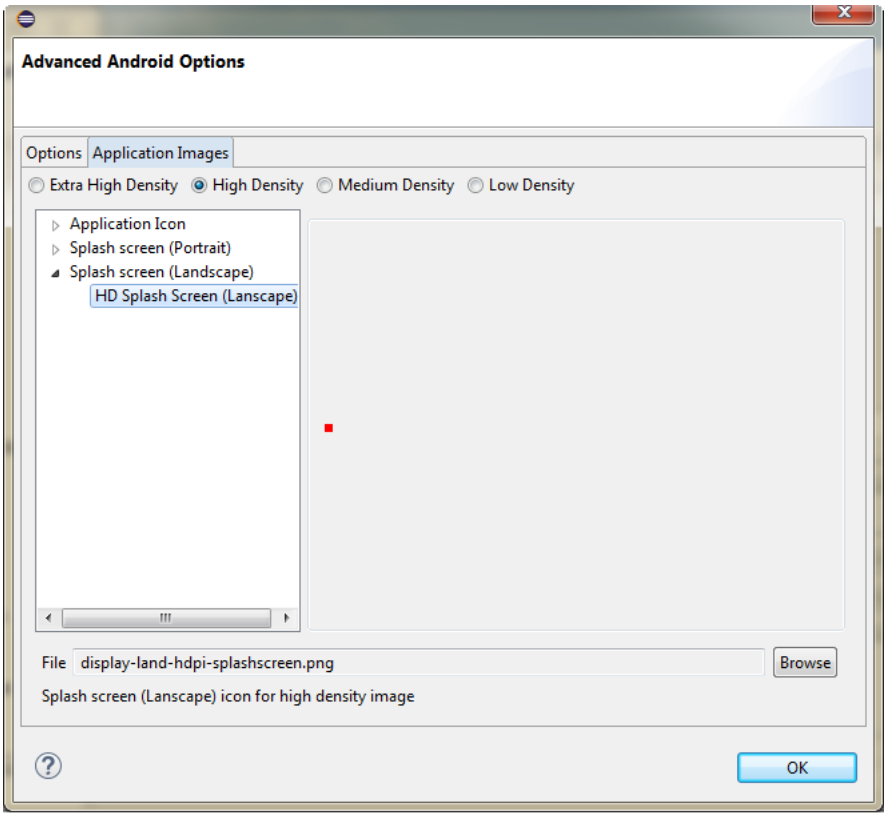


Figure 27–10 shows selecting images for application icons and portrait orientation splash screen images that applications use for displaying on devices with low-, medium-, high- and extra-high density displays.

Before you begin:

Obtain the images in the PNG, JPEG, or GIF file format that use the dimensions, density, and components that are appropriate to Android theme and that can also support multiple screen types. The default location for images in *assembly directory*\res\android For more information, see "Supporting Multiple Screens" document, available from the Android Developers website (http://developer.android.com/guide/practices/screens_support.html).

To add custom images:

1. Click **Run** > **Run Configurations** to open the deployment configuration dialog.
2. Click **Options** to open the Advanced Android Options dialog, and then click the Application Images tab.

   Choose the density of the image you want to change, then expand the appropriate node to choose the image you want to change.
3. Click **Browse** to select the new image you want to use and click **OK**.

27.3.1.6 What Happens When OEPE Deploys Images for Android Applications

As shown in Table 27–1, each image file is copied to a subdirectory called drawable, named for the drawable object, described on the Android Developers website (http://developer.android.com/reference/android/graphics/drawable/Drawable.html). Each drawable directory matches the image density (ldpi, mdpi, hdpi, and

xhdpi) and orientation (port, land). Within these directories, OEPE renames each icon image file as adfmf_icon.png and each splash screen image as adfmf_loading.png.

> **Note:** In order to view the contents of *assembly project*\.main.android, you must change the default view of the Project Explorer. For more information, see Section 2.3.2, "What Happens When You Create an MAF Application."

Table 27–1 Deployment File Locations for Seeded Application Images

Source File (...\res\android)	Temporary Deployment File (...\.main.android\build\release\res)
display-ldpi-icon.png	drawable-ldpi\adfmf_icon.png
display-mdpi-icon.png	drawable-mdpi\adfmf_icon.png
display-hdpi-icon.png	drawable-hdpi\adfmf_icon.png
display-xhdpi-icon.png	drawable-xhdpi\adfmf_icon.png
display-port-ldpi-splashscreen.png	drawable-port-ldpi\adfmf_loading.png
display-port-mdpi-splashscreen.png	drawable-port-mdpi\adfmf_loading.png
display-port-hdpi-splashscreen.png	drawable-port-hdpi\adfmf_loading.png
display-port-xhdpi-splashscreen.png	drawable-port-xhdpi\adfmf_loading.png
display-land-ldpi-splashscreen.png	drawable-land-ldpi\adfmf_loading.png
display-land-mdpi-splashscreen.png	drawable-land-mdpi\adfmf_loading.png
display-land-hdpi-splashscreen.png	drawable-land-hdpi\adfmf_loading.png
display-land-xhdpi-splashscreen.png	drawable-land-xhdpi\adfmf_loading.png

For custom images, OEPE copies the set of application icons from their specified location to the corresponding density and orientation subdirectory of the temporary deployment location.

27.3.2 How to Deploy an Android Application to an Android Emulator

You can deploy the mobile application directly to an Android emulator.

Before you begin:

Deployment to an Android emulator requires the following:

- Configure the keystore password and key alias details in the Android Platform preference page (accessed by choosing **Window** > **Preferences** > **Oracle** > **Mobile Application Framework** > **Android** > **Android Keystore**).

  > **Note:** You must install the Android 4.0.*n* platform (API 14 or later).

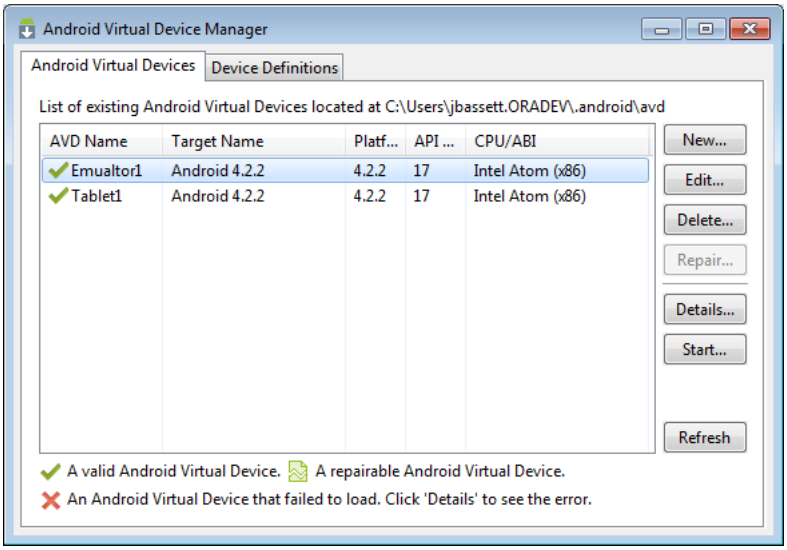
- Ensure that the Android Virtual Device instance configuration reflects the ARM system image.
- Start the Android emulator before you deploy an application.

  You can start the emulator using the Android Virtual Device Manager, as illustrated in Figure 27–11, or from the command line by first navigating to the

`tools` directory (located in `Android\android-sdk`) and then starting the emulator by first entering `emulator -avd` followed by the emulator name (such as `-avd AndroidEmulator1`).

Note: You can run only one Android emulator at a time.

Figure 27–11 Starting an Emulator Using Android Virtual Device Manager

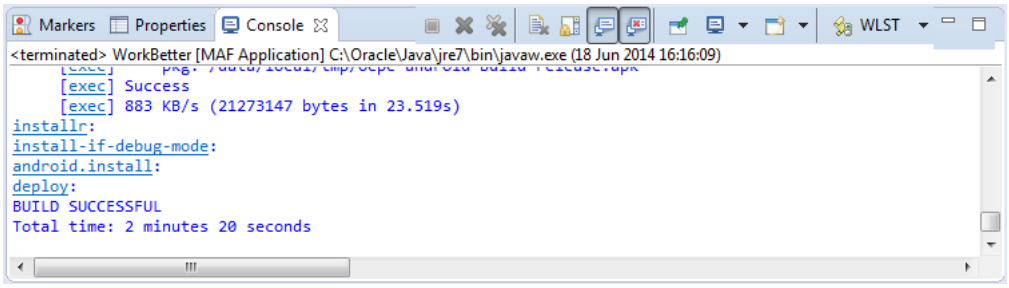


To deploy an application to an Android emulator:

1. Choose either **Run**, then **Run Configurations** or **Run**, then **Debug Configurations**. Under the MAF Applications node select an Android deployment configuration.
2. Check that the target is the one you want to use, and click either **Run** or **Debug**.

   Occasionally the deployment process may hang. In this case, kill the adb server then restart it, and retry the deployment.
3. Review the deployment log in the console, as shown in Figure 27–12. The deployment log notes that the deployer starts the Android Debug Bridge server when it detects a running instance of an Android emulator.

Figure 27–12 The Deployment Log



27.3.3 How to Deploy an Application Locally as an APK File

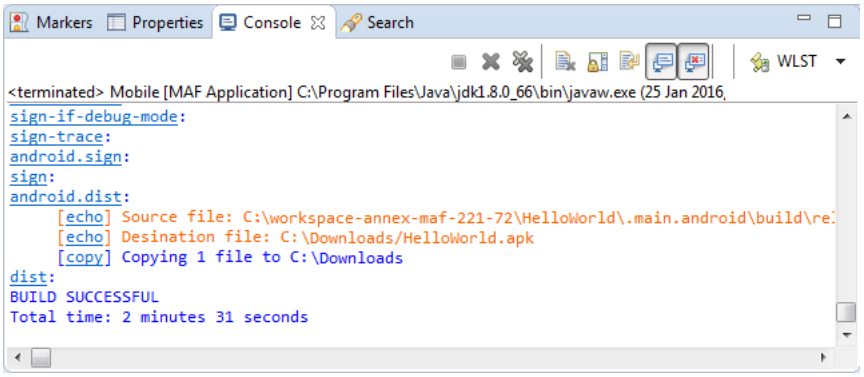
You can deploy an application as an .apk file on the local filesystem.

Note: You must configure the signing options in the Android Platform preference page (accessed by choosing **Window** > **Preferences** > **Mobile Application Framework**) as described in Section 27.3.1.3, "Defining the Android Signing Options."

To deploy an Android application to a local file:

1. Choose **Run**, then **Run Configurations** to open the Run Configurations dialog. Under the MAF Applications node select an Android deployment configuration.
2. Under Target Configuration, select **Generate APK Only** and enter a location where the APK file will be generated.
3. click **Run**
4. Review the deployment log in the console, as shown in Figure 27–12. The deployment log notes the source of the application, and shows the destination.

Figure 27–13 The Deployment Log



27.3.4 How to Deploy an Application to an Android-Powered Device

You can deploy a mobile application directly to an Android-powered device that runs on a platform of 2.*n* (API Level 9) or later.

Before you begin:

Connect the device to the development computer that hosts OEPE, as described in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

Ensure that the debug signing credentials are configured in the Android Platform preference page, shown in Figure 27–4.

To deploy an application to an Android device:

1. Choose **Run**, then **Run Configurations** to open the Run Configurations dialog. Under the MAF Applications node select an Android deployment configuration.
2. Check that the target is the one you want to use, and click **Run**.
3. Review the deployment log in the console, as shown in Figure 27–12. The deployment log notes that the deployer starts the Android Debug Bridge server when it detects a running instance of an Android emulator.

Figure 27–14 The Deployment Log

1. Choose either **Run**, then **Run Configurations** or **Run**, then **Debug Configurations**. Under the MAF Applications node select an Android deployment configuration.
2. Check that the target is the one you want to use, and click either **Run** or **Debug**.

27.3.5 How to Publish an Android Application

After you have tested and debugged the application, as described in Chapter 30, "Testing and Debugging MAF Applications," you can publish it to an application marketplace (such as Google Play) by following the instructions provided on the Android Developers website (`http://developer.android.com/tools/publishing/publishing_overview.html`).

Before you begin:

Create a Run Configuration.

> **Note:** You must configure the signing options in the Android Platform preference page (accessed by choosing **Window** > **Preferences** > **Mobile Application Framework**) as described in Section 27.3.1.3, "Defining the Android Signing Options."

To deploy an application as an .apk file:

1. Choose **Run**, then **Run Configurations** to create a run configuration. Under the MAF Applications node select an Android deployment configuration.
2. Check that the target is the one you want to use, and click **Run**.
3. Review the deployment log in the console.
4. Publish the application to an application marketplace.

27.3.6 What Happens in OEPE When You Create an .apk File

Deploying an application results in the following being deployed in an `.apk` file.

- The content in the `adfmsrc`
- The content in the `.adf` folder
- `maf-application.xml` and `maf-feature.xml` files
- `logging.properties` file
- The JVM 1.4 files

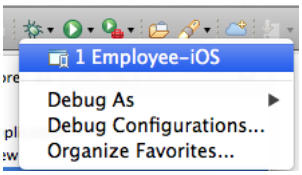
Table 27–2 Contents of the .apk File

Content	Location Within the .apk File
The content in the `.adf` folder	The root folder of the Android application file (`[apkRoot]\.adf`)
The content in the `adfmsrc` folder	The deployment packages the content in the `adfmsrc` folder into the default JAR file, which is located in a folder called `user` (`[apkRoot]\user`). This JAR file is added to the `.apk` using the Android Asset Packaging Tool (AAPT) and has a default name of the form `ANDROID_MOBILE_NATIVE_archiveN` where `N` is the nth Android created profile (you can override this name when creating the profile). This JAR file contains the following: ▪ Any `.class` files generated from `.java` files that are added to the view controller project, as well as the `adfmsrc` content. The `.java` files are compiled using the JVM 1.4 JDK `javac` tool. ▪ Contains data binding and pagedef metadata files. This JAR file is not processed by the Dalvik virtual machine. Because the `.class` files run in the JDK, they do not need to be converted into the Dalvik bytecode format (`.dex`).
`adfmf_application.xml` and `adfmf_feature.xml` files	Located in a file called Configuration (`[apkRoot]\Configuration`).
`logging.properties` file	Located in the root of the application file.
JVM 1.4 files	The JVM files are packaged into two separate folders: ▪ The library file (`\framework\build\java_res\libs\`) in the template will be packaged into a `lib` folder in the `APK - [apkRoot]\lib`. ▪ The `\framework\build\java_res\assets\storage` file is packaged in the `assets\storage` directory in the `APK - [apkRoot]\assets\storage`.

27.3.7 Selecting the Most Recently Used Deployment Configurations

After you select a deployment action, OEPE creates a shortcuts on the Debug and Run buttons on the main toolbar, as shown in Figure 27–15. Click the down-arrow, and choose the configuration that enables you to easily redeploy the application using that same deployment action.

Figure 27–15 Choosing a Recently Used Configuration



27.3.8 Viewing the Device/Emulator Log in the Console

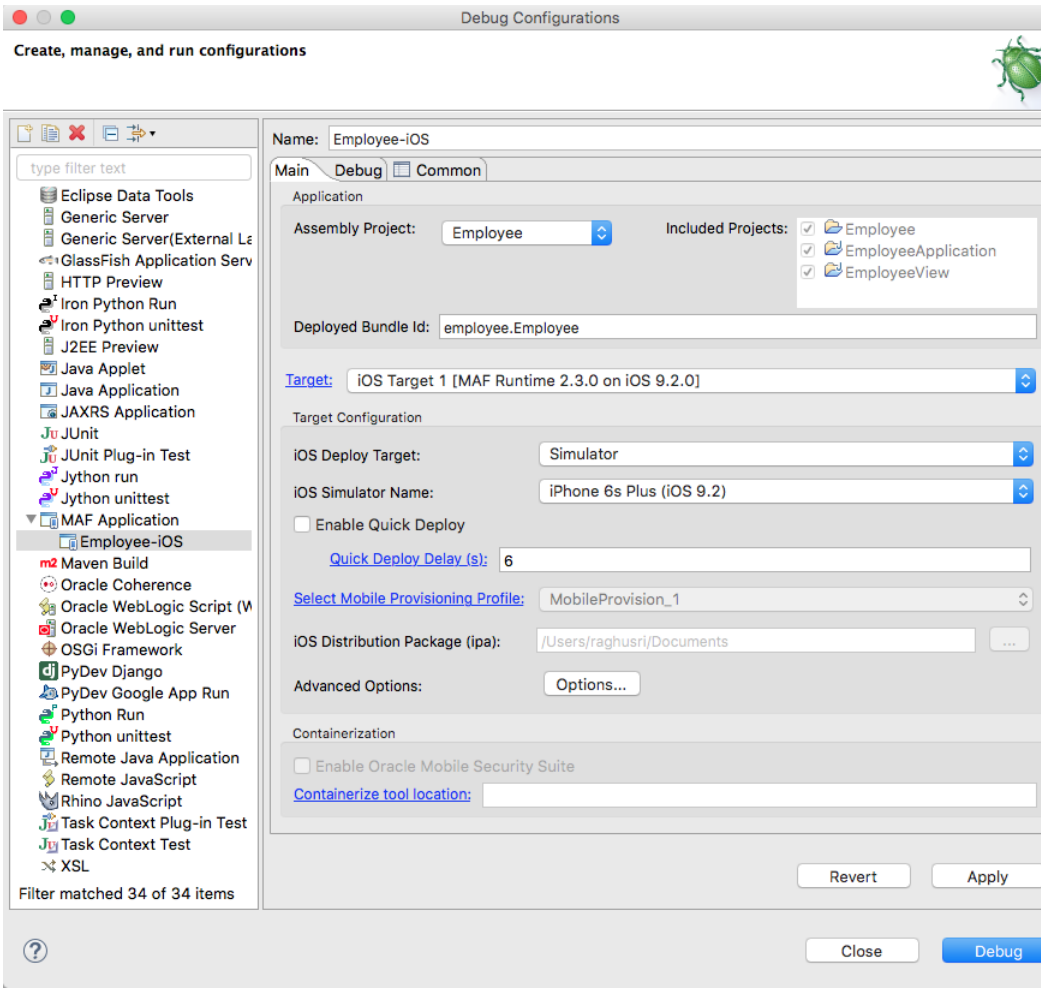
In OEPE, the log output from the device or emulator is streamed inside the console, as shown in Figure 27–16. This is very useful for debugging. For more information, see Section 30.6.1, "How to Configure Logging Using the Properties File."

Figure 27–16 Viewing the Device or Emulator Log

27.4 Deploying an iOS Application

The deployment configuration dialog, shown in Figure 27–17, enables you to deploy an iOS application directly to an iOS simulator or to a device through iTunes. You can only deploy an iOS application from an Apple computer. Deployment to the iOS simulator does not require membership to either the iOS Developer Program or the iOS Developer Enterprise Program; registration as an Apple developer, which provides access to versions of Xcode that are not available through the App Store, will suffice. For more information on iOS developer programs, which are required for deployment to iOS-powered devices (and are described at Section 27.4.7, "How to Deploy an Application to an iOS-Powered Device," and Section 27.4.10, "How to Distribute an iOS Application to the App Store"), see `https://developer.apple.com/programs/`.

Figure 27–17 The Deployment Configuration Dialog (for iOS Applications)



27.4.1 How to Create an iOS Deployment Configuration

For iOS, use the Debug Configuration dialog to define the iOS application build configuration as well as the locations for the splash screen images and application icons.

Before you begin:

Download Xcode (which includes the Xcode IDE, performance analysis tools, the iOS simulator, the Mac OS X, and the iOS SDKs) to the Apple computer that also runs OEPE.

> **Tip:** Refer to the Certification and Support Matrix on Oracle Technology Network (`http://www.oracle.com/technetwork/developer-tools/maf/documentation/index.html`) for the minimum supported version required to compile applications.

Because Xcode is used during deployment, you must install it on the Apple computer before you deploy the mobile application from OEPE.

Tip: While the current version of Xcode is available through the App Store, you can download prior versions through the following site:

`https://developer.apple.com/xcode/`

Access to this site requires an Apple ID and registration as an Apple developer.

After you download Xcode, you must enter the location of its xcodebuild tool and, for deployment to iOS simulators, the location of the iOS simulator's SDK, in the iOS Platform preference page. For more information, see *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

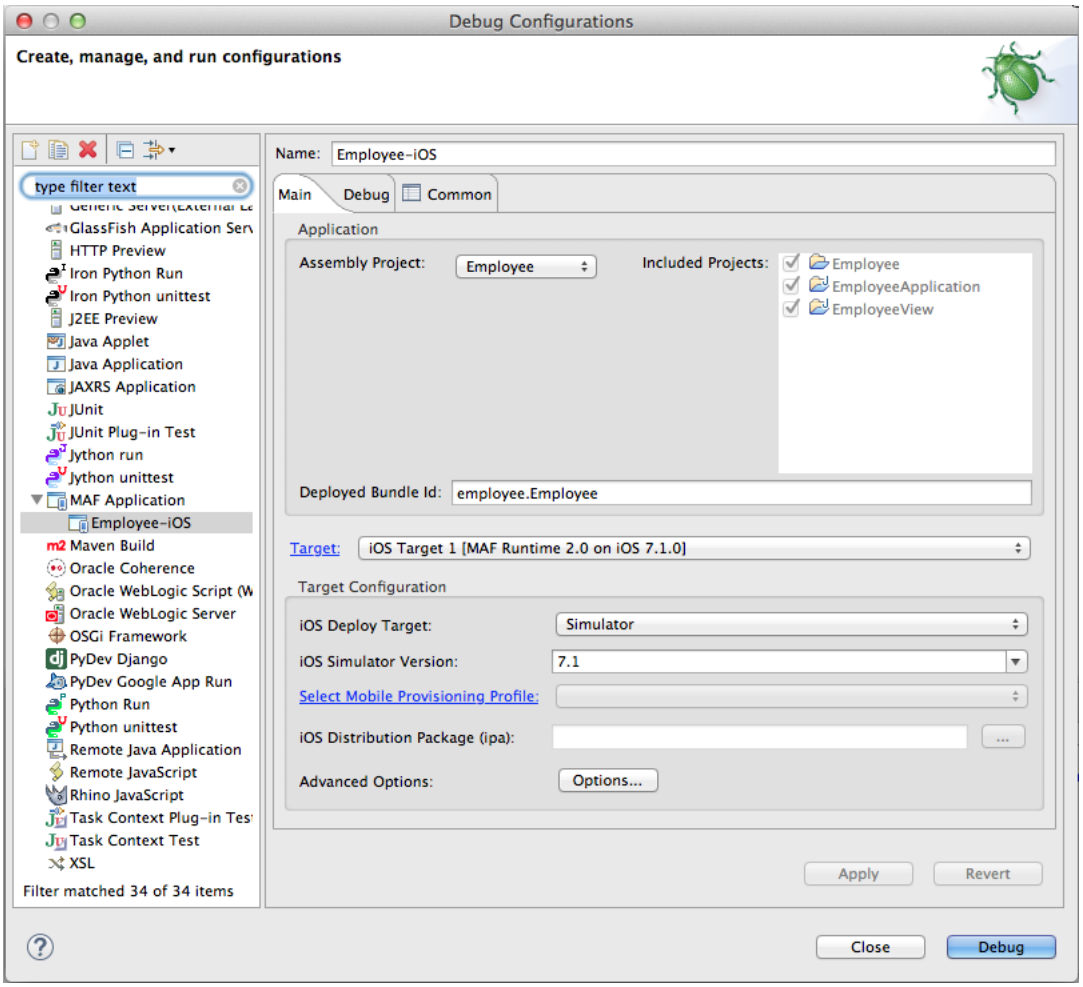
Note: Run both iTunes and the iOS simulator at least once before entering their locations in the iOS Platform preference page.

To deploy a mobile application to an iOS-powered device (as opposed to deployment to an iOS simulator), you must obtain both a provisioning profile and a certification from the iOS Provisioning Profile as described in Section 27.4.2, "Setting the Device Signing Options."

To create a deployment configuration:

1. Choose **Run > Debug Configurations**, as shown in Figure 27–18.

Figure 27–18 Setting the iOS Options



2. Accept the default values, or define the following:

   - **Assembly Project**—Choose from the list of those available.
   - **Deployed Bundle Id**—If needed, enter a Bundle Id to use for this application that identifies the domain name of the company. The deployed bundle Id must be unique for each application installed on an iOS device and must adhere to reverse-package style naming conventions (that is, *com.<organization name>.<company name>*). For more information, see the *App Distribution Guide*, which is available through the iOS Developer Library at `http://developer.apple.com/library/ios/navigation/`). For information on obtaining the Bundle Seed Id using the iOS Provisioning Portal, see Section 27.4.9.3, "Registering an Application ID." See also Section 3.2, "Setting Display Properties for a MAF Application."

     Note: The deployed bundle Id cannot contain spaces.

     Because each deployed bundle Id is unique, you can deploy multiple mobile applications to the same device. Two applications can even have the same name as long as their deployed bundle Ids are different. Mobile applications deployed to the same device are in their own respective sandboxes. They are

unaware of each other and do not share data (they have only the Device scope in common).

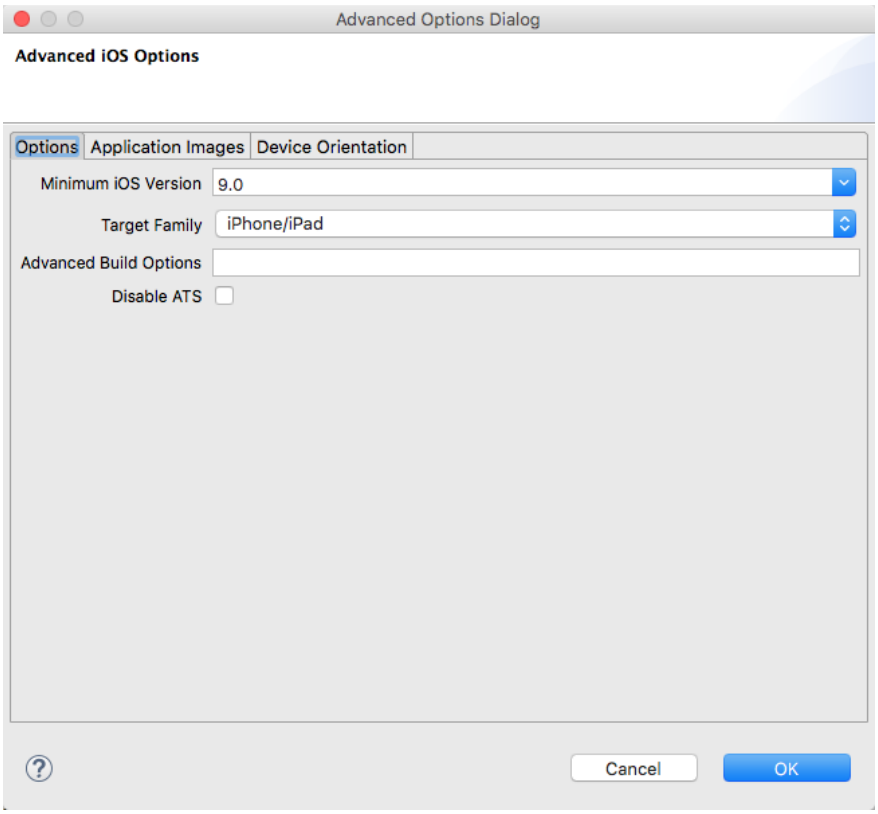
- **Target**—Choose from the list of targets configured in the Preferences dialog, or click **Target** to add a new target.
- **Target Configuration**
  - **iOS Deploy Target**—Choose device or simulator.
  - **iOS Simulator Version**—When you select `Simulator` as the deploy target, select the version of the emulator to which you are deploying the application. To find the available target versions, select **Hardware** and then **Version** on an iPhone simulator. The minimum version is 6.0. The default setting is **<Highest Available>**. For more information, see the *iOS Simulator User Guide*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

Note: Older versions of the iOS target version are usually available in the simulator for testing.

  - **Select Mobile Provisioning Profile**—
  - **iOS Distribution Package (ipa)**—If needed, enter the name for the `.ipa` file or the `.app` file. MAF creates an `.ipa` file when you select either the **Deploy to distribution package** or **Deploy to iTunes for synchronization to device** options in the Deployment Action dialog, shown in Figure 27–17. It creates an `.app` file when you select the **Deploy application to simulator** option. Otherwise, accept the default name. For more information, see Section 27.4.7, "How to Deploy an Application to an iOS-Powered Device" and Section 27.4.10, "How to Distribute an iOS Application to the App Store."

    By default, MAF bases the name of the `.ipa` file (or `.app` file) on the application `id` attribute configured in the `maf-application.xml` file. For more information, see Section 3.2, "Setting Display Properties for a MAF Application."
  - **Advanced Options**—Click Options to open the Advanced Option dialog, where you can set various options, as shown in Figure 27–19.

Figure 27–19 Setting the Minimum iOS Version



Minimum iOS Version—Indicates the earliest version of iOS to which you can deploy the application. The default value is the current version. The version depends on the version of the installed SDK.

- **Target Family**—Select the family of iOS products on which the application is intended to run. The default option is for both iPad and iPhone.
- **Advanced Build Options**—Specify additional arguments that Xcode can use when it builds the MAF application.
- **Disable Application Transport Security**—App Transport Security (ATS) is a security policy that restricts network requests from the application to use only approved secured transport protocols. MAF enables ATS by default. Select Disable Application Transport Security to deploy your MAF application without ATS enabled.

27.4.2 Setting the Device Signing Options

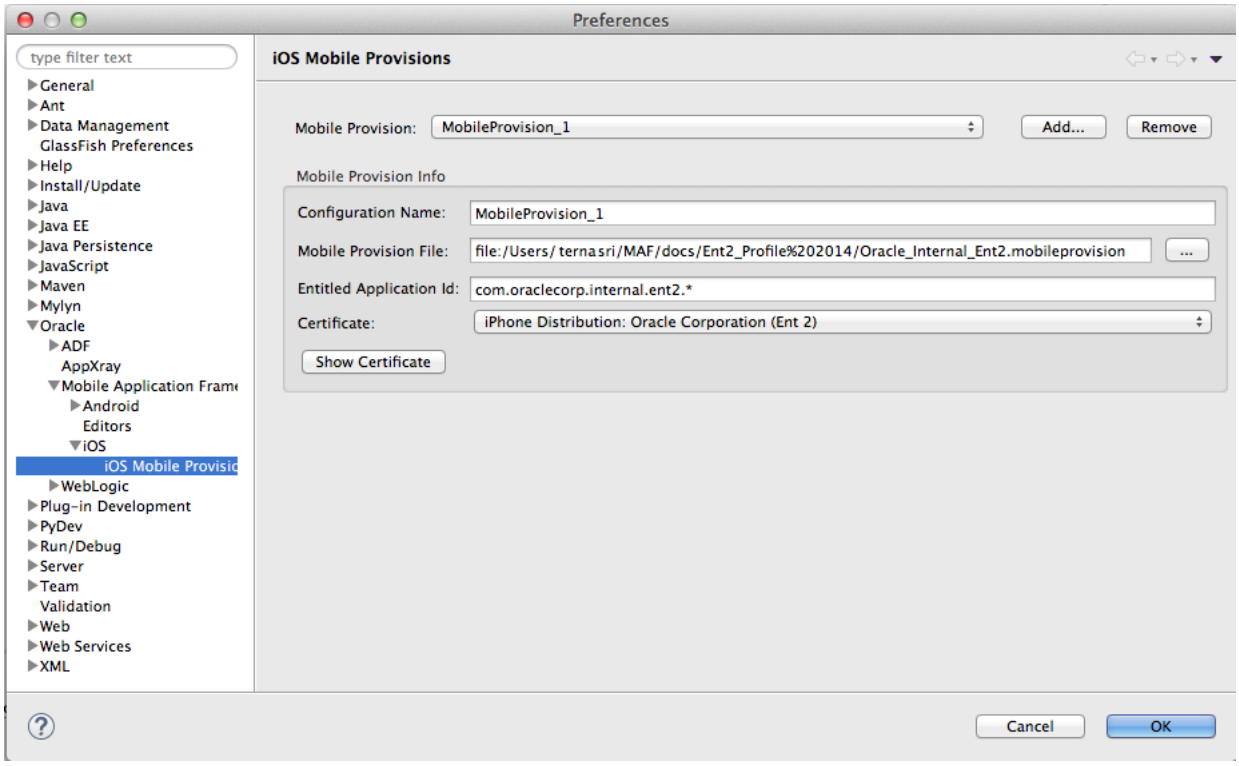
The iOS Platform preference page for iOS includes fields for the location of the provisioning profile on the development computer and the name of the certificate. You must define these parameters if you deploy an application to an iOS device or as a MAF Application Archive.

Note: Neither a certificate nor a provisioning profile are required if you deploy a mobile application to an iOS simulator.

To set the signing options:

1. Choose **Window**, then **Preferences**, and then **Mobile Application Framework**.
2. Choose **iOS** and then choose **iOS Mobile Provisions**.
3. In the Mobile Provision Info section of the page, shown in Figure 27–20, enter the location of the provisioning profile in **Mobile Provisioning File**.
4. OEPE enters the **Certificate**.

Figure 27–20 The Device Signing Section of the iOS Platform Preference Page

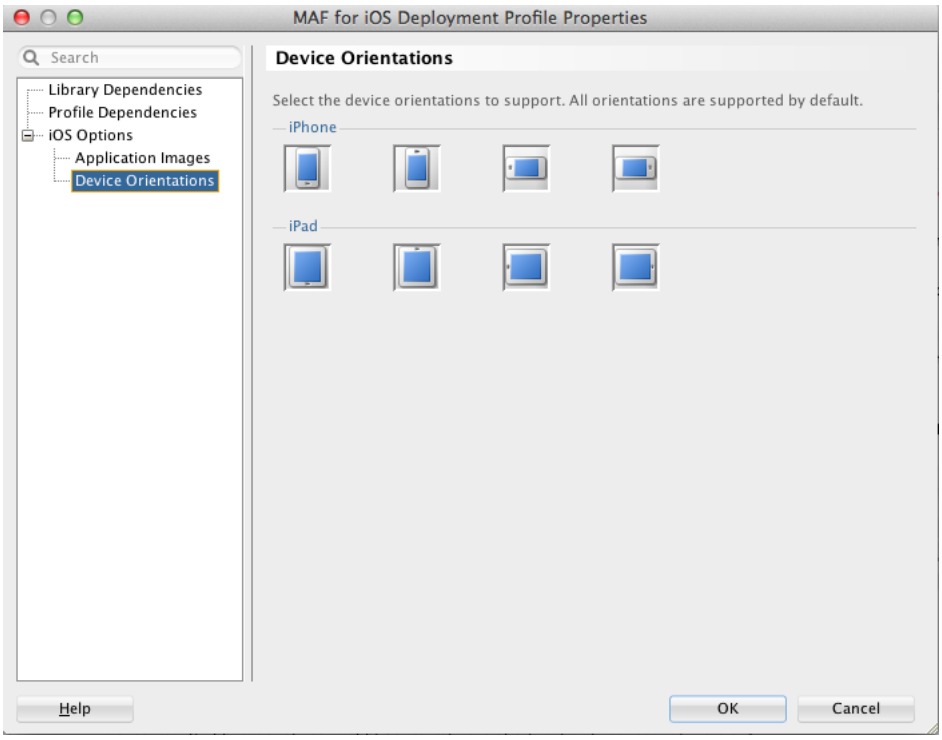


Note: There are provisioning profiles used for both development and release versions of an application. While a provisioning profile used for the release version of an application can be installed on any device, a provisioning profile for a development version can only be installed on the devices whose IDs are embedded into the profile. For more information, see the *App Distribution Guide*, which is available from the iOS Developer Library (http://developer.apple.com/library/ios/navigation/).

27.4.3 How to Restrict the Display to a Specific Device Orientation

By default, MAF supports all orientations for both iPhone and iPad. If, for example, an application must display only in portrait and in upside-down orientations on iPads, you can limit the application to rotate only to these orientations using the Device Orientation page, shown in Figure 27–21

Figure 27–21 Select a Device Orientation



To limit the display of an application to a specific device orientation:

1. Choose **Device Orientations**, as shown in Figure 27–21.
2. Clear all unneeded orientations from among those listed in Table 27–3. By default, MAF deploys to all of these device orientations. By default, all of these orientations are selected.

Table 27–3 iPhone Device Orientations

Icon	Description
	iPad, portrait—The home button is at the bottom of the screen.
	iPad, upside-down—The home button is at the top of the screen.
	iPad, landscape left—The home button is at the left side of the screen.
	iPad, landscape right—The home button is at the right side of the screen.

Table 27–3 (Cont.) iPhone Device Orientations

Icon	Description
	iPhone, portrait—The home button is at the bottom of the screen.
	iPhone, upside-down—The home button is at the top of the screen.
	iPhone, landscape left—The home button is at the left side of the screen.
	iPhone, landscape right—The home button is at the right side of the screen.

3. Click **OK**.

27.4.4 What Happens When You Deselect Device Orientations

Deselecting a device orientation updates the source `.plist` file.

27.4.5 Using Images with iOS Applications

By default, mobile applications deployed to an iOS device through iTunes, or deployed as an archive (`.ipa` file) for download, use the default Oracle image unless otherwise specified.

By selecting an iTunes artwork image as the icon for the deployed application, you override the default image. You can use an image to differentiate between versions of the application. Figure 27–22 illustrates the difference between the default image and a user-selected image, where Application4 is displayed with the default image and Application6 is displayed with a user-selected image (the Oracle icon, scaled to 512 x 512 pixels).

During deployment, MAF ensures that the icon displays in iTunes by adding the iTunes artwork image to the top-level of the `.ipa` file in a file called *iTunesArtwork*.

Figure 27–22 Custom and Default Application Icons

For more information on iOS application icon images, see the "Icon and Image Design" section in *iOS Human Interface Guidelines*. This document is available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

You can rebrand an application by overriding the default Oracle image used for application icons and artwork with custom images. There are three ways you can use your own custom images in an application:

- The simplest way is to override the default Oracle image as part of the Run/Debug Configuration, see Section 27.4.5.1, "Overriding the Default Oracle Images in the Configuration."
- You can overwrite the default file for each custom image with your own. see Section 27.4.5.2, "Replacing the Default Images.".
- You can copy your custom icons to the same folder as the default icons and modify the properties file for the device profile for each image, see Section 27.4.5.3, "Add Custom Images."

> **Note:** All images must be in the PNG format.

For each application, there is an image folder which contains the image files and a properties file which maps predefined keys to a specific image in that folder.

- The images folder is *application*`/res/ios`.
- The properties file is in the same folder, `ios-resources.properties`.

The properties file configures how custom images are deployed, for example:

```
# iPad (76 x 76 px)
# Appears on home screen on iPad (later than iOS 6.1)
# deploys to Icon-76.png
oepe.adfmf.res.source.icon76=Icon-76.png

# iPad (152 x 152 px)
# Appears on home screen on iPad with retina display (later than iOS 6.1)
# deploys to Icon-76@2x.png
oepe.adfmf.res.source.icon76@2x=Icon-76@2x.png
```

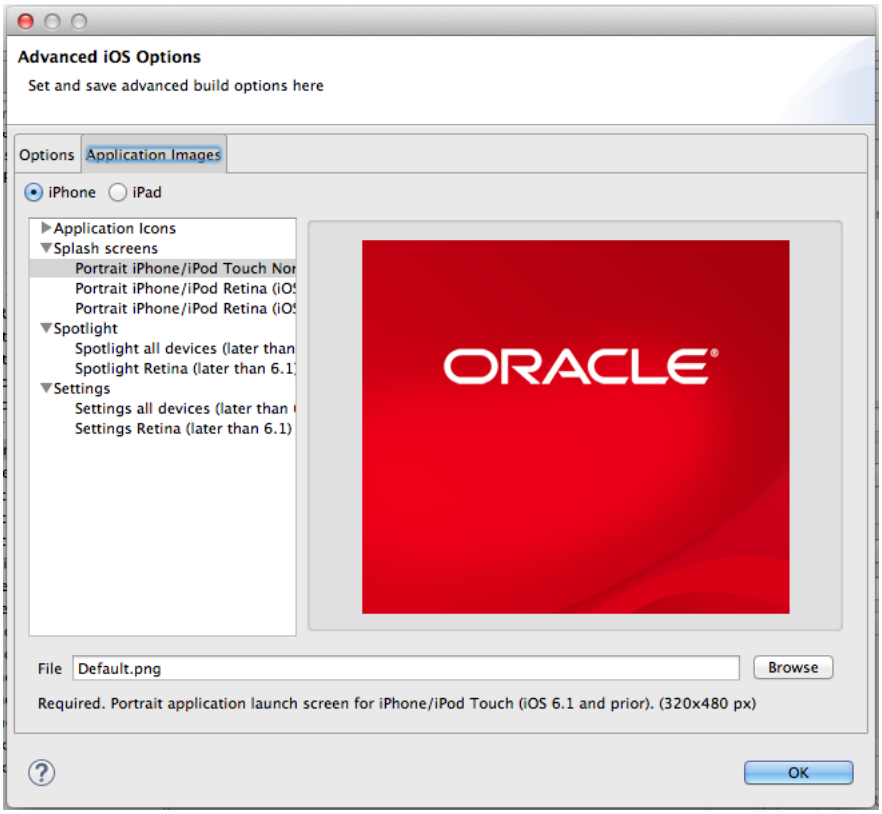
```
# All devices  (40 x 40 px)
# Spotlight on all devices (later than iOS 6.1)
# deploys to Icon-40.png
oepe.adfmf.res.source.icon40=Icon-40.png
```

27.4.5.1 Overriding the Default Oracle Images in the Configuration

The Application Images page enables you to rebrand an application by overriding the default Oracle image used for application icons and artwork with custom images. The options in this page, shown in Figure 27–23, enable you to enter the locations of custom images used for different situations, and device resolutions.

> **Note:** When you change an icon it is changed for all the devices the icon is used for.

Figure 27–23 Overriding the Default Images



To override the default images:

1. Open the deployment configuration by choosing **Run** > **Debug Configurations**.
2. Click **Options** to open the Advanced iOS Options dialog.
3. Click the Application Images tab, and choose whether you want to change the image for iPhone or iPod.
4. Choose the image you want to change, and **Browse** to select the image file you want to use. This image file must exist within the current application.

During deployment, OEPE copies the custom image file into the deployment configuration and renames it to match the name of the default image.

5. Click **OK**.

27.4.5.2 Replacing the Default Images

If you are happy to overwrite the default images, you can overwrite them in the image directory with your own custom images.

To overwrite images:

1. Navigate to the folder containing the images for your application, *application*/res/ios.
2. Identify the images you want to replace and check the size of the image and where it is used in the properties file in the same folder, ios-resources.properties.
3. Replace the default images with your custom images.

27.4.5.3 Add Custom Images

You can add your custom images to the folder containing the images for your application, and edit the properties file to modify each key/value pair to match the device profile for each image.

To add custom images:

1. Add the custom images to the folder containing the images for your application, *application*/res/ios.

   For example, if you have a custom 57x57 pixel image for iPhone called myicon.png, add it to this folder.

2. Open the properties file in the same folder, ios-resources.properties and edit it to add your custom icon.

   For example, to be able to use myicon.png, change the key as follows:

   ```
   oepe.adfmf.res.source.icon57@2x=myicon.png
   ```

27.4.6 How to Deploy an iOS Application to an iOS Simulator

The Deployment Actions dialog enables you to deploy an iOS application directly to an iOS simulator.

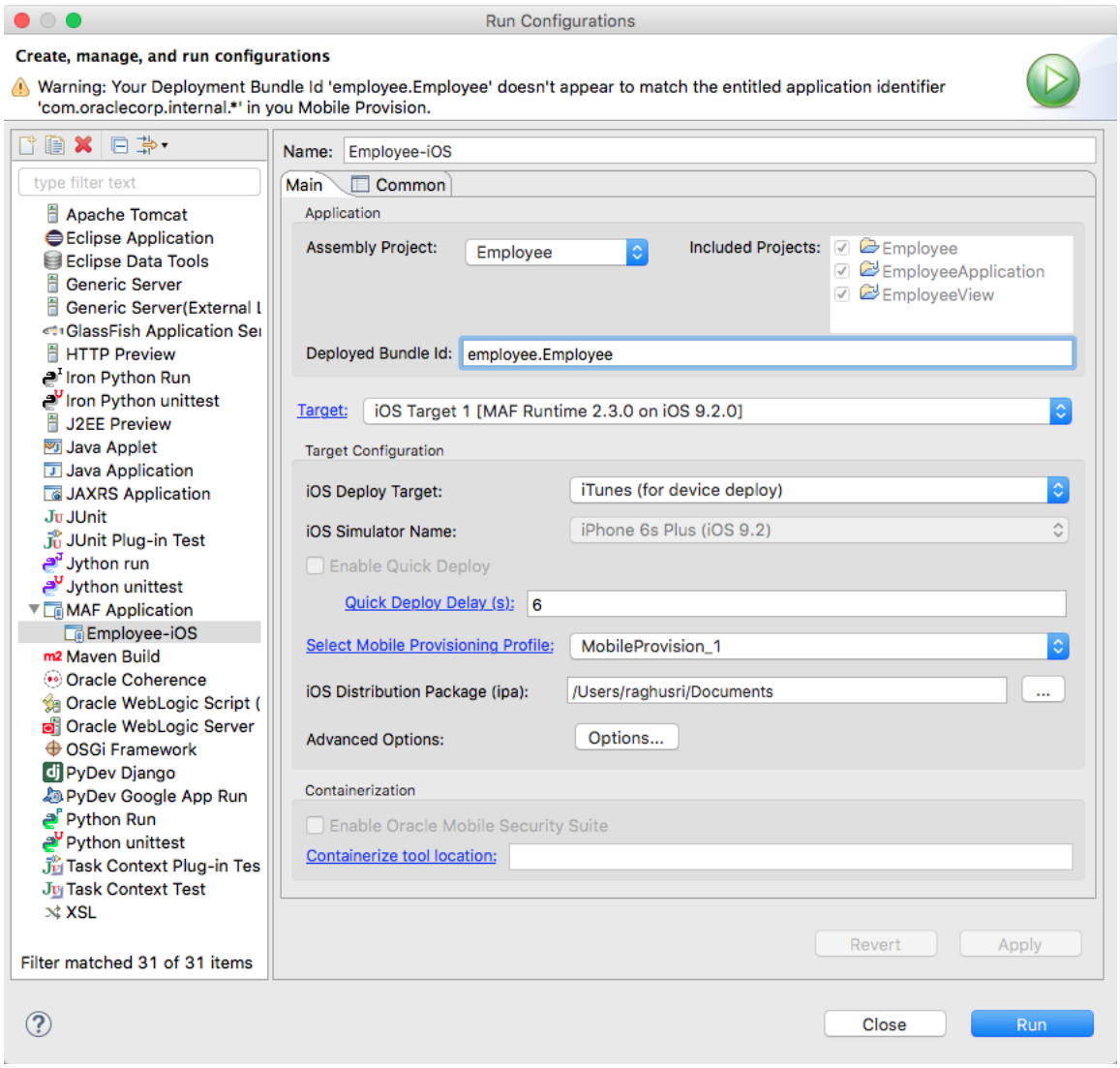
Before you begin:

To enable deployment to an iOS simulator, you must perform the following tasks:

To deploy an application to an iOS simulator:

1. Choose **Run > Run Configurations**, to open the Run Configurations dialog, as shown in Figure 27–24.

Figure 27–24 Run Configurations Dialog



2. Accept the default values, or define the following:

   - **Assembly Project**—Choose from the list of those available.
   - **Deployed Bundle Id**—If needed, enter a Bundle Id to use for this application that identifies the domain name of the company. The deployed bundle Id must be unique for each application installed on an iOS device and must adhere to reverse-package style naming conventions (that is, *com.<organization name>.<company name>*). For more information, see the *App Distribution Guide*, which is available through the iOS Developer Library at `http://developer.apple.com/library/ios/navigation/`). For information on obtaining the Bundle Seed Id using the iOS Provisioning Portal, see Section 27.4.9.3, "Registering an Application ID." See also Section 3.2, "Setting Display Properties for a MAF Application."

   Note: The deployed bundle Id cannot contain spaces.

Because each deployed bundle Id is unique, you can deploy multiple mobile applications to the same device. Two applications can even have the same name as long as their deployed bundle Ids are different. Mobile applications deployed to the same device are in their own respective sandboxes. They are unaware of each other and do not share data (they have only the Device scope in common).

- **Target**—Choose from the list of targets configured in the Preferences dialog, or click **Target** to add a new target.
- **Target Configuration**
  - **iOS Deploy Target**—Choose `Simulator`.
  - **iOS Simulator Version**—Select the version of the emulator to which you are deploying the application. To find the available target versions, select **Hardware** and then **Version** on an iPhone simulator. The minimum version is 6.0. The default setting is **<Highest Available>**. For more information, see the *iOS Simulator User Guide*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

Note: Older versions of the iOS target version are usually available in the simulator for testing.

  - **Enable Quick Deploy**—Select to have OEPE automatically redeploy an application to an iOS simulator after a build. For more information, see Section 27.4.6.1, "Using Quick Deploy."
  - **Select Mobile Provisioning Profile**—
  - **iOS Distribution Package (ipa)**—If needed, enter the name for the `.app` file. MAF creates an `.app` file when you select the **Deploy application to simulator** option. Otherwise, accept the default name.

    By default, MAF bases the name of the `.ipa` file on the application `id` attribute configured in the `maf-application.xml` file. For more information, see Section 3.2, "Setting Display Properties for a MAF Application."
  - **Advanced Options**—Click Options to open the Advanced Option dialog, where you can set various options.

    Minimum iOS Version—Indicates the earliest version of iOS to which you can deploy the application. The default value is the current version. The version depends on the version of the installed SDK.

27.4.6.1 Using Quick Deploy

OEPE can automatically redeploy an application to an iOS simulator. It does this by keeping track of the changes that you make in the application and then redeploying after a build.

The updates occur when the application builds, and not when resources change. If you do not have automatic builds set, then you should select **Save automatically before build** in the General > Workspace page of the Preferences dialog so that all resources are committed before building and redeploying.

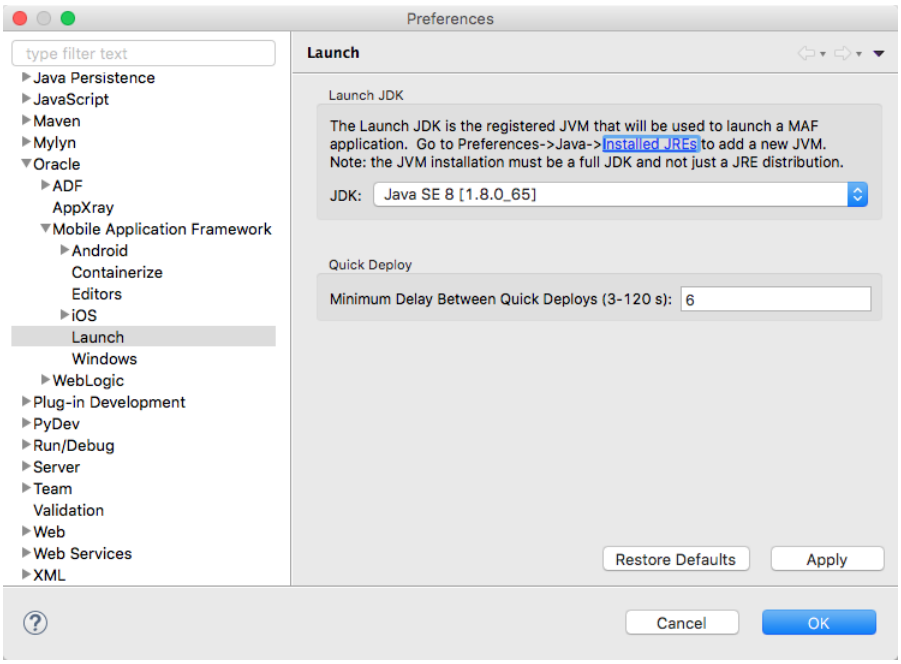
There is a delay of a few seconds before the application restarts so that the application is not redeployed on every build if builds are close together. The default is 6 seconds,

but you can change this in OEPE preferences. If you find that the application is restarting too often, you can change the minimum time before quick deploy.

To change the minimum quick deploy delay:

1. Open the Preferences dialog from the Window menu (Eclipse menu on MacOS) and navigate to **Oracle > Mobile Application Framework > Launch** to open the Launch preferences page, as show in Figure 27–25.

Figure 27–25 Quick Deploy Delay



2. Enter the number of seconds between 3 and 120 before an application should be redeployed when Quick Deploy has been set. The default is 6 seconds.

27.4.7 How to Deploy an Application to an iOS-Powered Device

The **Deploy to iTunes for Synchronization to device** option enables you to deploy a mobile application to an iOS-powered device for debugging and testing. Deployment to an iOS-powered device or to a distribution site requires membership to either the iOS Developer Program or the iOS Developer Enterprise Program. For more information, see `https://developer.apple.com/programs/`.

Before you begin:

You cannot deploy an application directly from OEPE to a iOS device; an application must instead be deployed from the Applications folder in Apple iTunes. To accomplish this, you must perform the following tasks:

- Download Apple iTunes to your development computer and run it at least once to create the needed folders and directories.
- Set the location of the `Automatically Add to iTunes` folder (the location used for application deployment) in the iOS Platform preference page, shown in Figure 27–26.

Tip: Although your user home directory (`/User/<username>/Music/iTunes/iTunes Media/Automatically Add to iTunes.localized`) is the default directory for the iTunes Media folder, you can change the location of this folder as follows:

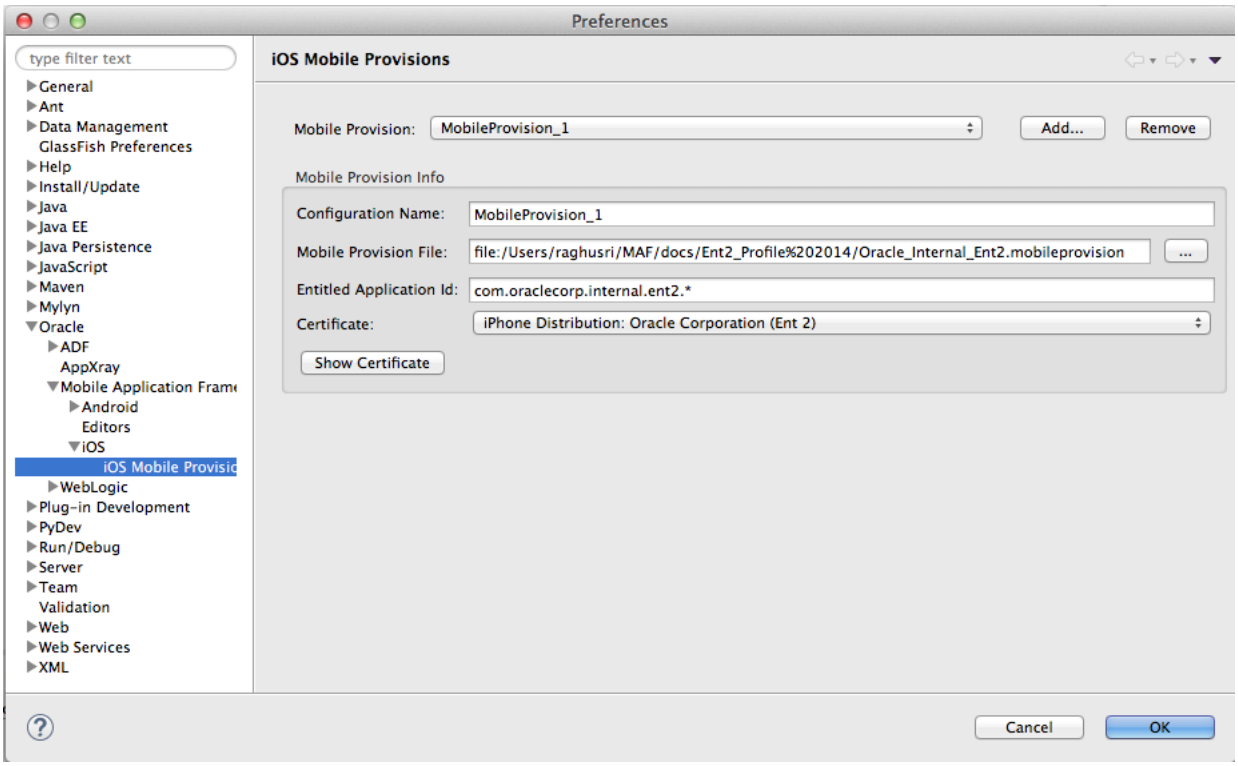
1. In iTunes, select **Edit**, **Preferences**, then **Advanced**.
2. Click **Change** and then browse to the new location.
3. Consolidate the library.
4. Delete the original iTunes Media folder.

For instructions, refer to Apple Support (`http://support.apple.com`).

You must also update the location in the iOS Platform preference page.

- Set the location of the Xcode folder where the xcodebuild tool is invoked, such as `/Developer/usr/bin` in Figure 27–26.

Figure 27–26 Setting the Locations for the iTunes Media Folder and the xcodebuild System



- Enter the name of the certificate and the location of the provisioning profile in the iOS Platform preference page. The OS Provisioning Portal generates the certificate and provisioning profile needed for deployment to iOS devices, or for publishing `.ipa` files to the App Store or to an internal download site.

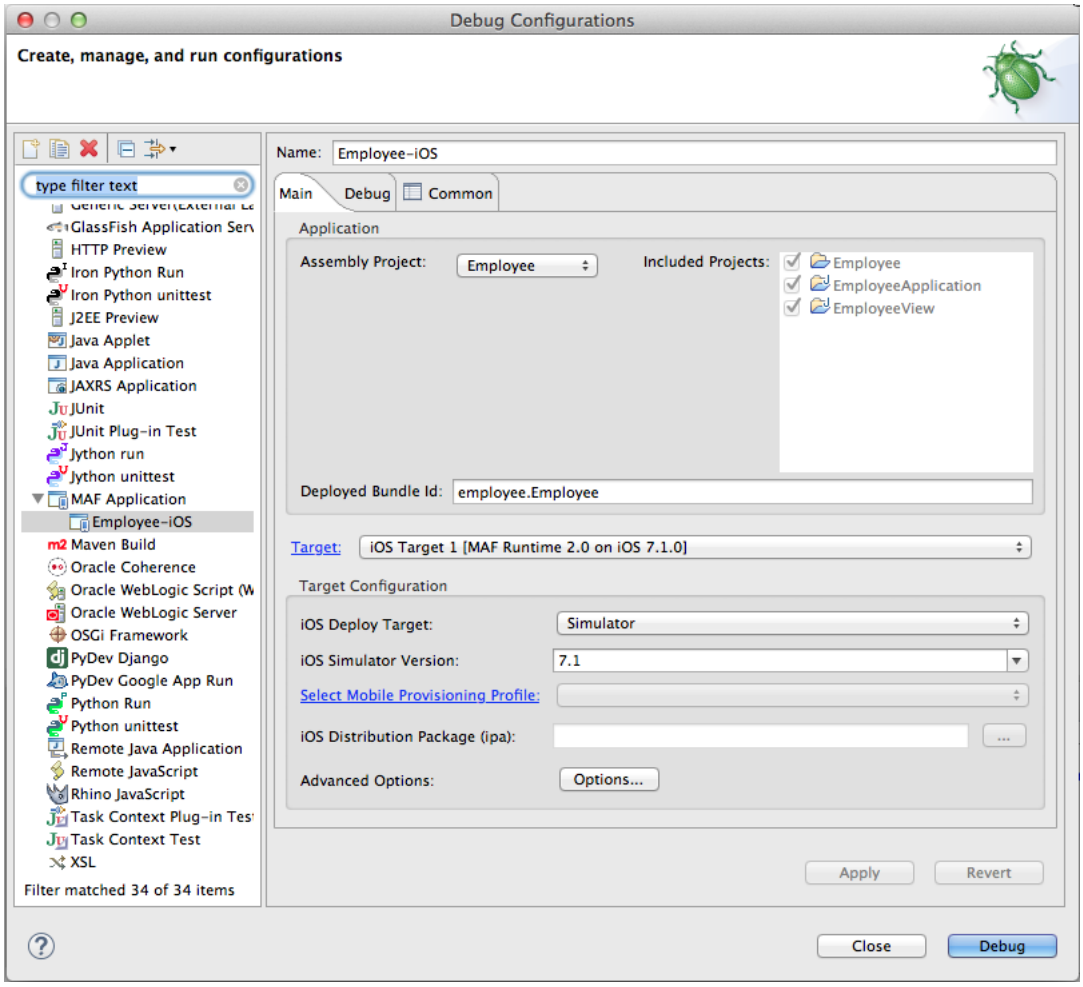
Note: The deployment will fail unless you set the iOS provisioning profile and certificate to deploy to a device or to an archive. MAF logs applications that fail to deploy under such circumstances. For more information, see Section 27.4.9, "What You May Need to Know About Deploying an Application to an iOS-Powered Device."

- In the iOS Options page of the deployment configuration, select **Run** as the build mode and then **OK**.
- Refer to the *App Distribution Guide*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

To deploy an application to an iOS-powered device:

1. Choose **Run > Debug Configurations**, as shown in Figure 27–18.

Figure 27–27 Setting the iOS Options



2. Accept the default values, or define the following:
   - **Assembly Project**—Choose from the list of those available.
   - **Deployed Bundle Id**—If needed, enter a Bundle Id to use for this application that identifies the domain name of the company. The deployed bundle Id must be unique for each application installed on an iOS device and must adhere to reverse-package style naming conventions (that is, *com.<organization name>.<company name>*). For more information, see the *App Distribution Guide*, which is available through the iOS Developer Library at `http://developer.apple.com/library/ios/navigation/`). For information on obtaining the Bundle Seed Id using the iOS Provisioning Portal, see Section 27.4.9.3, "Registering an Application ID." See also Section 3.2, "Setting Display Properties for a MAF Application."

Note: The deployed bundle Id cannot contain spaces.

Because each deployed bundle Id is unique, you can deploy multiple mobile applications to the same device. Two applications can even have the same name as long as their deployed bundle Ids are different. Mobile applications deployed to the same device are in their own respective sandboxes. They are unaware of each other and do not share data (they have only the Device scope in common).

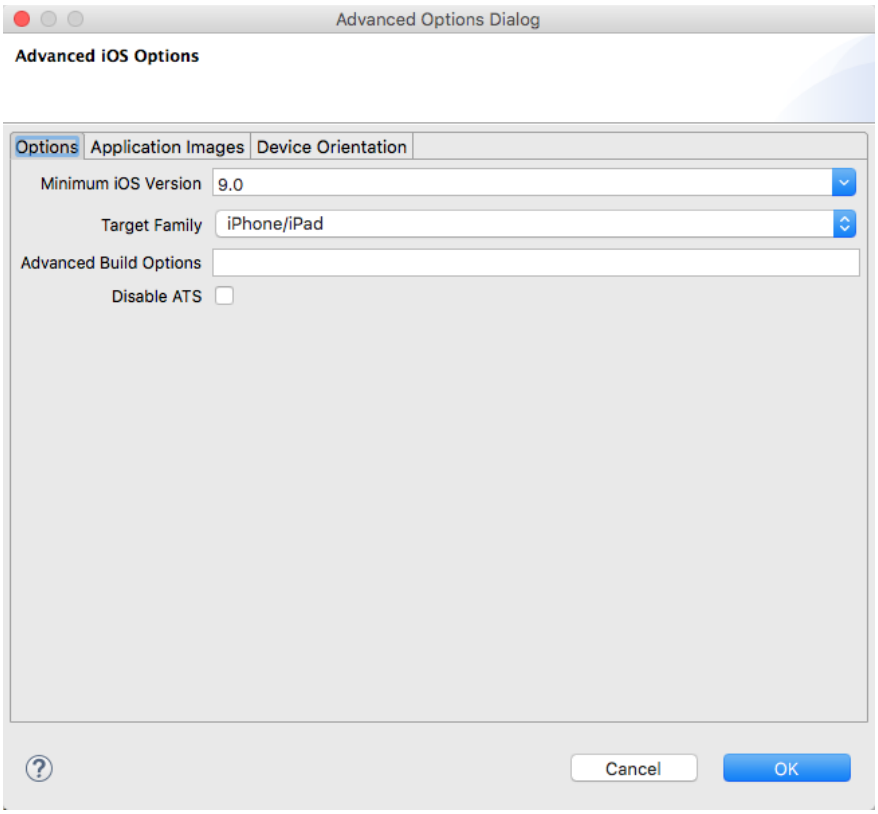
- **Target**—Choose from the list of targets configured in the Preferences dialog, or click **Target** to add a new target.
- **Target Configuration**
  - **iOS Deploy Target**—Choose device or simulator.
  - **iOS Simulator Version**—When you select `Simulator` as the deploy target, select the version of the emulator to which you are deploying the application. To find the available target versions, select **Hardware** and then **Version** on an iPhone simulator. The minimum version is 6.0. The default setting is **<Highest Available>**. For more information, see the *iOS Simulator User Guide*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

Note: Older versions of the iOS target version are usually available in the simulator for testing.

  - **Select Mobile Provisioning Profile**—
  - **iOS Distribution Package (ipa)**—If needed, enter the name for the `.ipa` file or the `.app` file. MAF creates an `.ipa` file when you select either the **Deploy to distribution package** or **Deploy to iTunes for synchronization to device** options in the Deployment Action dialog, shown in Figure 27–17. It creates an `.app` file when you select the **Deploy application to simulator** option. Otherwise, accept the default name. For more information, see Section 27.4.7, "How to Deploy an Application to an iOS-Powered Device" and Section 27.4.10, "How to Distribute an iOS Application to the App Store."

    By default, MAF bases the name of the `.ipa` file (or `.app` file) on the application `id` attribute configured in the `maf-application.xml` file. For more information, see Section 3.2, "Setting Display Properties for a MAF Application."
  - **Advanced Options**—Click Options to open the Advanced Option dialog, where you can set various options, as shown in Figure 27–19.

Figure 27–28 Setting the Minimum iOS Version



Minimum iOS Version—Indicates the earliest version of iOS to which you can deploy the application. The default value is the current version. The version depends on the version of the installed SDK.

27.4.8 What Happens When You Deploy an Application to an iOS Device

The application appears in the iTunes Apps Folder, similar to the one illustrated in Figure 27–22 after a successful deployment.

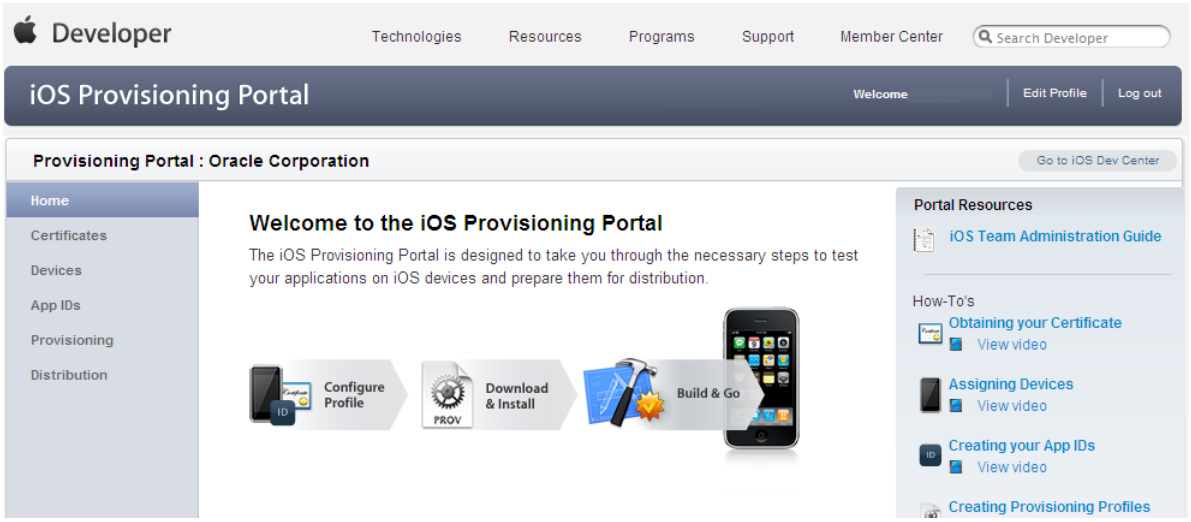
27.4.9 What You May Need to Know About Deploying an Application to an iOS-Powered Device

You cannot deploy an iOS application (that is, an `.ipa` file) to an iOS-powered device or publish it to either the App Store or to an internal hosted download site without first creating a provisioning profile using the iOS Provisioning Portal, which is accessible only to members of the iOS Developer Program. You enter the location of the provisioning profile and the name of the certificate in the Options page as described in Section 27.4.2, "Setting the Device Signing Options."

As noted in the *App Distribution Guide*, (which is available through the iOS Developer Library at `http://developer.apple.com/library/ios/navigation/`), a provisioning profile associates development certificates, devices, and an application ID. The iOS Provisioning Portal enables you to create these entities as well as the provisioning profile.

Tip: After you download the provisioning profile, double-click this file to add it to your `Library/MobileDevice/Provisioning Profile` directory.

Figure 27–29 The iOS Provisioning Portal



27.4.9.1 Creating iOS Development Certificates

A certificate is an electronic document that combines information about a developer's identity with a public key and private key. After you download a certificate, you essentially install your identity into the development computer, as the iOS Development Certificate identifies you as an iOS developer and enables the signing of the application for deployment. In the iOS operating environment, all certificates are managed by the Keychain.

Using the Certificates page in the iOS Provisioning Portal, you log a CSR (Certificate Signing Request). The iOS Provisioning Portal issues the iOS Development Certificate after you complete the CSR.

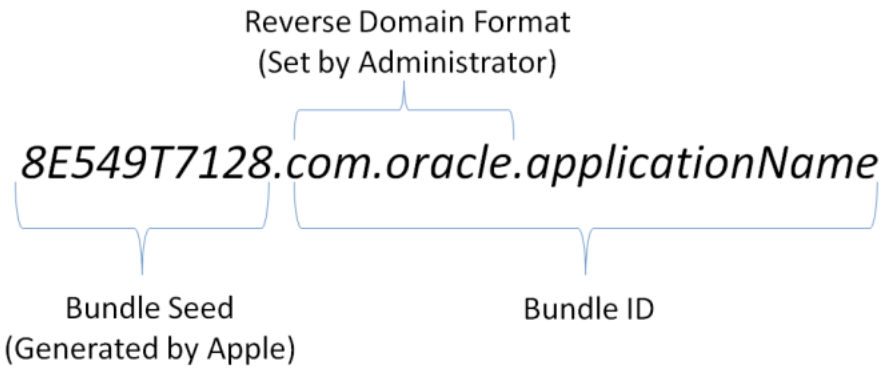
27.4.9.2 Registering an Apple Device for Testing and Debugging

After you install a certificate on your development computer, review the Current Available Devices tab (located in the iOS Provisioning Portal's Devices page) to identify the Apple devices used by you (or your company) for testing or debugging. The application cannot deploy unless the device is included in this list, which identifies each device by its serial number-like Unique Device Identifier (UDID).

27.4.9.3 Registering an Application ID

An application ID is a unique identifier for an application on a device. An application ID is comprised of the administrator-created reverse domain name called a Bundle Identifier in the format described in Section 3.2, "Setting Display Properties for a MAF Application" prefixed by a ten-character alpha-numeric string called a bundle seed, which is generated by Apple. Figure 27–30 illustrates an application ID that is unique, one that does not share files or the Keychain with any other applications.

Figure 27–30 An Explicit Application ID



Using a wildcard character (*) for the application name, such as *8E549T7128.com.oracle.**, enables a suite of applications to share an application ID. For example, if the administrator names *com.oracle.MAF.** on the iOS Provisioning Portal, it enables you to specify different applications (*com.oracle.MAF.application1* and *com.oracle.MAF.application2*).

Note: For applications that receive push notifications, the application ID must be a full, unique ID, not a wildcard character; applications identified using wildcards cannot receive push notifications. For more information, see the "Provisioning and Development" section of *Local and Push Notification Programming Guide*, available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`)

When applications share the same prefix, such as *8E549T7128*, they can share files or Keychains.

Note: The Bundle Id must match the application ID set in the Options page of the deployment configuration.

27.4.10 How to Distribute an iOS Application to the App Store

After you test and debug an application on an iOS device, you can distribute the application to a wider audience through the App Store or an internal download site. To publish an application to the App Store, you must submit the `.ipa` file to iTunes Connect, which enables you to add `.ipa` files to iTunes, as well as update applications and create test users.

Before you begin:

Before you distribute the application, you must perform the following tasks:

- In the iOS Platform preference page, shown in Figure 27–26, enter the location of the `Automatically Add to iTunes` directory.

  Tip: Run iTunes at least once before entering this location. See also Section 27.4.7, "How to Deploy an Application to an iOS-Powered Device."

- Test the application on an actual iOS device. See Section 27.4.7, "How to Deploy an Application to an iOS-Powered Device."
- Obtain a distribution certificate through the iOS Provisioning Portal.

> **Note:** Only the Team Agent can create a distribution certificate.

- Obtain an iTunes Connect account for distributing the `.ipa` file to iTunes. For information, see "Prepare App Submission" in the iOS Development Center's App Store Resource Center. Specifically, review the *App Store Review Guidelines* to ensure acceptance by the App Review Team.
- You may want to review both the and *iTunes Connect Developer Guide*. These guides are both available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).
- In the iOS Options page of the deployment configuration, select *App Distribution Guide* **Release** as the build mode and then click **OK**.

To distribute an iOS application to the App Store:

1. Choose **Run** > **Run Configurations**, and then select an iOS deployment configuration.
2. Choose **Deploy to Distribution Package**.
3. Review the Summary page, which displays the following values. Click **Finish**.
   - **Deployed Bundle Id**—The unique name that includes a Java language-like package name (*com.<organization name>.<application name>*) prefixed with the Bundle Seed that is generated from the iOS Provisioning Portal.
   - **File**—The file name of the final image deployed to an iOS target.
   - **Certificate**—The application's author. If this value has not been configured in the iOS Platform preference page of the deployment configuration, then the Summary page displays *<Not Specified>*.
   - **Provisioning Profile**—The name of the provisioning profile that associates one or more development certificates and devices with an application ID. If this value is not configured in the iOS Platform preference page, then the Summary page displays *<Not Specified>*.

   > **Note:** The Certificate and Provisioning Profile values cannot be noted as *<Not Specified>*; you must specify these values in the Options page to enable the `.ipa` file to be accepted by iTunes.

4. Log in to iTunes Connect.
5. Submit the `.ipa` file to iTunes Connect for consideration using the Manage Your Applications module and the Application Loader described in the "Adding New Apps" and "Using Application Loader" sections in iTunes Connect Developer Guide.
6. After the application has been approved, refer to the "Creating Test Users" section in *iTunes Connect Developer Guide* for information on using the *Manage Users* module. For testing multi-language applications, create a test user account for the regions for which the application is localized.

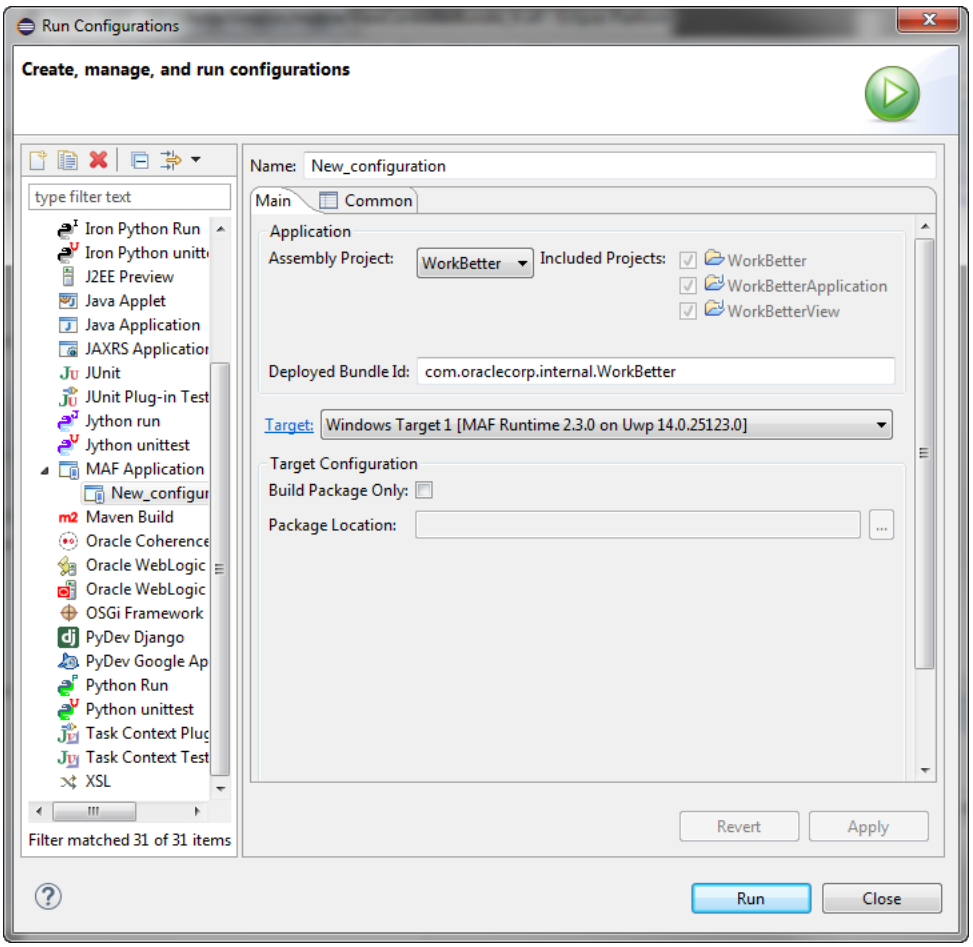
7. Refer to the "Editing and Updating App Information" section in *iTunes Connect Developer Guide* for information on updating the binary using the *Managing Your Application* module.

27.5 Deploying a MAF Application to the Universal Windows Platform

MAF applications can be deployed to the local machine where you develop the application or to an installation package that you use to install the application on a supported UWP device.

> **Note:** Debug for UWP is not supported in this release.

Figure 27–31 Deployment Configuration Dialog for UWP Applications



27.5.1 How to Create a Deployment Configuration for Universal Windows Platform

Release ready applications cannot be published to the Windows Store. Choose another mechanism to distribute your application. MAF provides a PowerShell script in the installation package that, when executed, installs the application on the UWP device.

Perform the following setup and configuration tasks before you attempt to deploy an application to the UWP:

1. Verify that your development machine meets the requirements to deploy a MAF application to the UWP. Among these requirements are that your machine be a UWP device. That is, it runs the Windows 10 operating system. You must enable Developer mode on this machine.

   For more information, see the What You Need to Develop an Application for the Universal Windows Platform section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

2. Install the Visual Studio software from Microsoft. This software contains the Windows SDK that enables deployment to the UWP.

   For more information, see the Setting Up Development Tools for the Universal Windows Platform section in Installing *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

3. Create a certificate (a personal information exchange file) to digitally sign the application you want to deploy.

   For more information, see the How to Create a PFX File for MAF Applications section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

4. Enter the following values in the Windows page of the Preferences dialog shown in Figure 27–32:

   - The location of the Windows SDK that is installed with the Visual Studio software from Microsoft.
   - (On the Windows Certificate page) The location and password (if required) for the certificate that you use to sign the application.

   You access the Preferences dialog from the Window menu (Eclipse menu on MacOS), as shown in Figure 27–32.

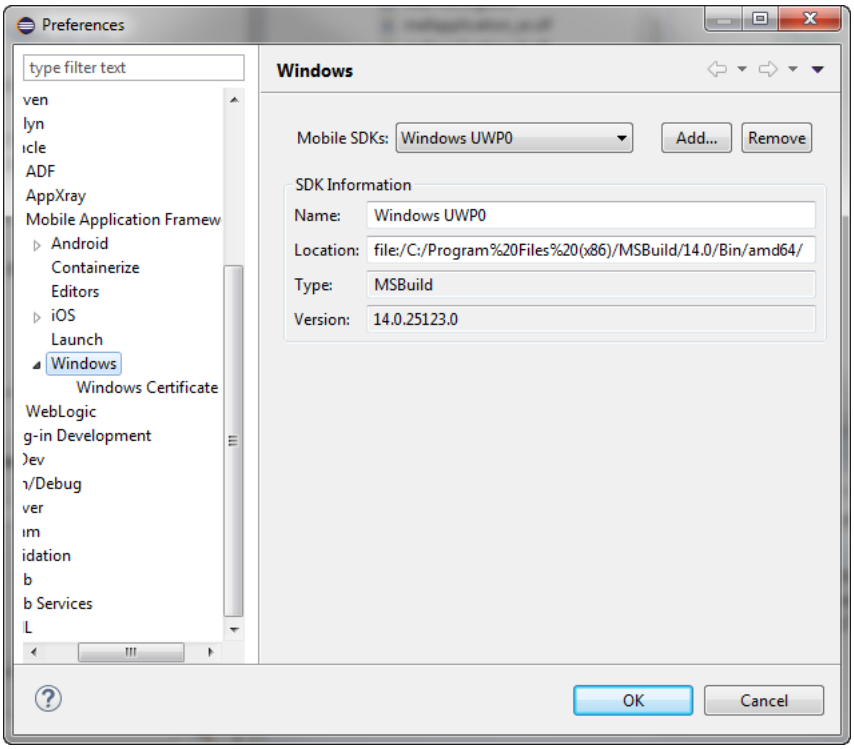
Before you deploy your application, note the following issues that can prevent deployment:

- Microsoft Windows imposes a maximum length for the path to directories and files. MAF application deployment fails if the path for a MAF application exceeds the maximum path length. To work around this issue, place the MAF application in a location where the directory path length is less than the maximum length mandated by Windows. For more information about the maximum path length limitation, see Microsoft's documentation.
- Ensure no other process uses the MAF application's data folder before you deploy the MAF application using the Deploy application to local machine option. The application data folder is typically located at `C:\Users\[userName]\AppData\Local\Packages\[appBundleId]_[id]`. An example of a scenario where another process uses this folder is if you have a file in the folder open with an application, such as Notepad. If you cannot determine what application process is using the directory/file(s), reboot your machine to resolve the issue.

To set the MAF preferences for the UWP platform SDKs:

1. Open the Preferences dialog from the Window menu (Eclipse menu on MacOS).
2. Navigate to **Oracle > Mobile Application Framework > Windows** to open the Windows preferences page, as show in Figure 27–32.

Figure 27–32 Setting UWP Preferences



3. Click **Add** to open the Select SDK dialog, and navigate to the location of the UWP SDK. Click **OK**.

To create the configuration:

1. Select **Run > Run Configurations** to open the Configurations dialog.
2. In the Configurations dialog, shown in Figure 27–31, enter a name for the configuration.
3. Choose the Assembly Project from the list of those available.
4. If you want to change the application ID, enter a new one in **Deployed Bundle Id**.
5. Choose the target, which is combination of the platform version and the MAF runtime version.
6. **Build Package Only**
   - Leave unselected if you want to deploy to the local machine.
   - Select if you want to build a package to deploy to other machines or tablets.

27.5.2 Defining the Windows Platform Signing Options

An application must be signed before it can be deployed to a Windows device and you do this by providing the location of a Windows Personal Information Exchange (`.pfx`) file.

For more information, see the How to Create a PFX File for MAF Applications section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

To specify the location of the `.pfx` file:

1. Open the Preferences dialog from the Window menu (Eclipse menu on MacOS) and navigate to **Oracle > Mobile Application Framework > Windows > Windows Certificate**.
2. Enter the certificate location and, if necessary, a password.

27.5.3 How to Deploy a MAF Application to the Universal Windows Platform

Deploy the application using a MAF for Windows deployment configuration that deploys the application to the UWP.

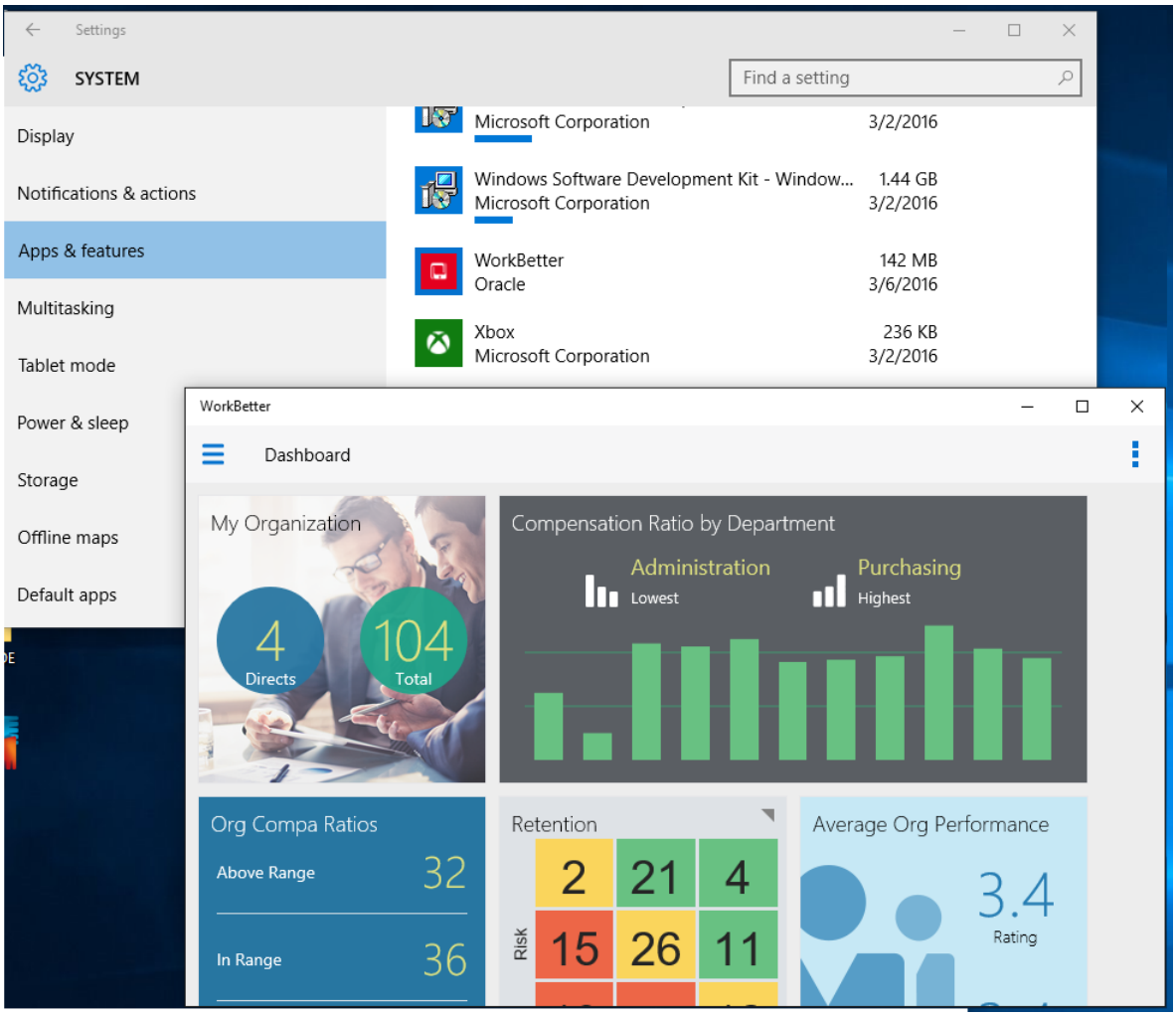
To deploy a MAF application to the UWP:

1. Choose **Run**, then **Run Configurations**. Under the MAF Applications node select an UWP deployment configuration.
2. Check that the target is the one you want to use, and click **Run**.
3. Review the deployment log in the console.

27.5.4 What Happens When You Deploy a MAF Application to the Universal Windows Platform

MAF deploys the application to the machine that you are using and installs the application there. In the foreground of Figure 27–33 is an instance of the WorkBetter sample application that has been deployed to a local machine in release mode. In the background of Figure 27–33, you can view the WorkBetter sample application listed among the installed applications on the Windows 10 machine's Apps & features page.

Figure 27–33 MAF Application Deployed in Release Mode on Windows Local Machine



27.5.5 How to Deploy an Application Locally as a Package

The default deployment launch configuration setting will always deploy application to the local machine. But there is also a way to deploy to other UWP compatible machines and tablets. For this you must change launch configuration to build a package, as described in Section 27.5.6, "What Happens When You Deploy Locally as a Package."

To deploy as a package:

1. Choose **Run**, then **Run Configurations**. Under the MAF Applications node select an UWP deployment configuration.
2. Select **Build Package Only** and enter a location for the generated package.

27.5.6 What Happens When You Deploy Locally as a Package

The package is generated to the location in the deployment configuration. The directory contains another directory (`MafTemplate_*_Test`). Distribute the contents of this latter directory to the end users with supported UWP devices who want to install your MAF application. The directory includes a PowerShell script (`Add-AppDevPackage.ps1`) that end users execute to install the application. In addition

to the script, the directory contains the application package, dependent packages, and the certificate that signed the application. The following example lists the contents:

```
Add-AppDevPackage.ps1
Add-AppDevPackage.resources
Dependencies
MafTemplate_1.0.0.0_x64.appx
MafTemplate_1.0.0.0_x64.appxsym
MafTemplate_1.0.0.0_x64.cer
```

The name of the `MafTemplate_*_Test` directory and files depends on the Version number.

For example, if you specify `3` as the Version number and select the Release build mode, the directory name is `MafTemplate_3.0.0.0_x64_Test`.

27.6 Deploying Feature Archive Files (FARs)

To enable re-use by MAF view controller projects, application features— typically, those implemented as MAF AMX or Local HTML— are bundled into an archive known as a Feature Archive (FAR). A FAR is a JAR file that contains the application feature artifacts that can be consumed by mobile applications. A FAR may contain Java classes, though these classes must be compiled. The example below illustrates the contents of a FAR, which includes a single `maf-feature.xml` file and a `connections.xml` file.

```
connections.xml (or some form of connection metadata)

  META-INF
     jar-connections.xml
     jar-adf-config.xml
     adfm.xml
     maf-feature.xml
     MANIFEST.MF
     task-flow-registry.xml

  oracle
    application1
      mobile
        DataControls.dcx
         DataBindings.cpx
         pageDefs
           view1PageDefs

  model
     Class1.class

  public_html
     adfc-mobile-config.xml
       index.html
       navbar-icon.html
       springboard-icon.html
     view1.amx
     task-flow-definition.xml
```

Working with Feature Archive files involves the following tasks:

1. Creating a Feature Archive file—You create a Feature Archive by exporting the view project as a FAR.

2. Using the Feature Archive file when creating a mobile application—This includes registering a FAR with the application, and then registering the features in the FAR with the application.
3. Deploying a mobile application that includes features from FARs—This includes unpacking the FAR to a uniquely named folder within the deployment template.

> **Note:** MAF generates FARs during the deployment process. You only need to deploy a view project if you use the FAR in another application.

27.6.1 How to Create a Mobile Feature Archive File

Use the OEPE Export wizard to create the `.far` file.

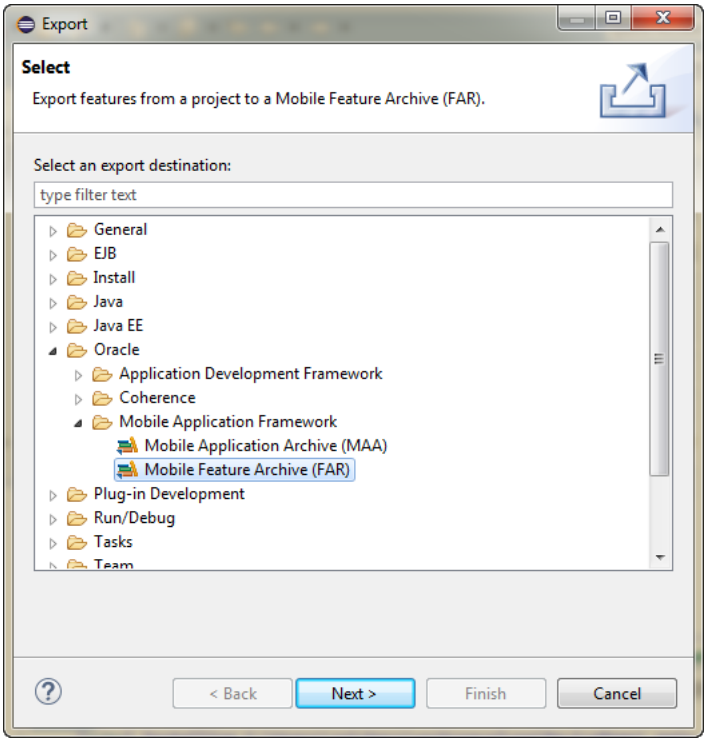
Before you begin:

Create the appropriate connections for the application. Because FARs may be used in different MAF applications with different connection requirements, choose a connection name that represents the connection source or the actual standardized connection name.

How to package mobile features as a Feature Archive File:

1. Click **File** then choose **Export**.
2. In the Export wizard, expand Oracle, then Mobile Application Framework, and choose Mobile Feature Archive (FAR), as shown in Figure 27–34.

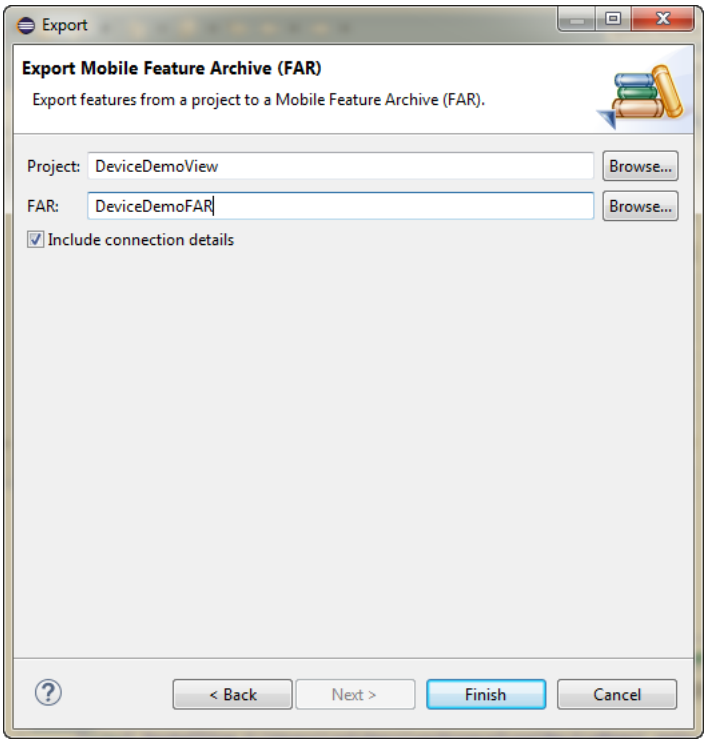
Figure 27–34 Creating a Mobile Feature Archive File



3. Click **Next**, and choose the view project containing the features you want to archive, and enter a name for the archive file, as shown in Figure 27–35.

Note: Name the profile appropriately. Otherwise, you may encounter problems if you upload more than one application feature with the same archive name.

Figure 27–35 Entering a Name and Path for the Mobile Feature Archive File



4. Click **Finish**. The Summary page, shown in Figure 27–36, displays the full path of where the Feature Archive file's JAR path is deployed.

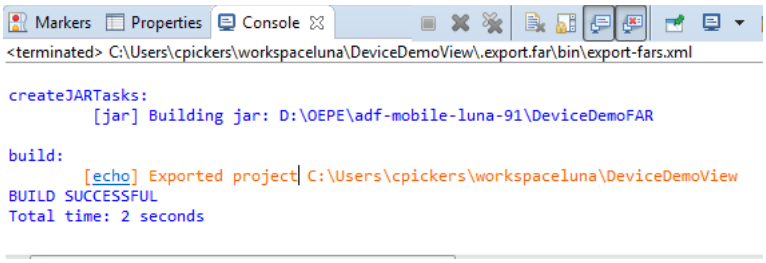
27.6.2 How to Deploy the Feature Archive

The Deployment Actions dialog enables you to deploy the FAR as a JAR file. This dialog includes only one deployment option, **Deploy to feature archive JAR file**.

How to deploy the Feature Archive deployment profile:

1. Right-click the view project and then select the Feature Archive deployment profile.
2. Click **Finish**. The Summary page, shown in Figure 27–36, displays the full path of where the Feature Archive file's JAR path is deployed.

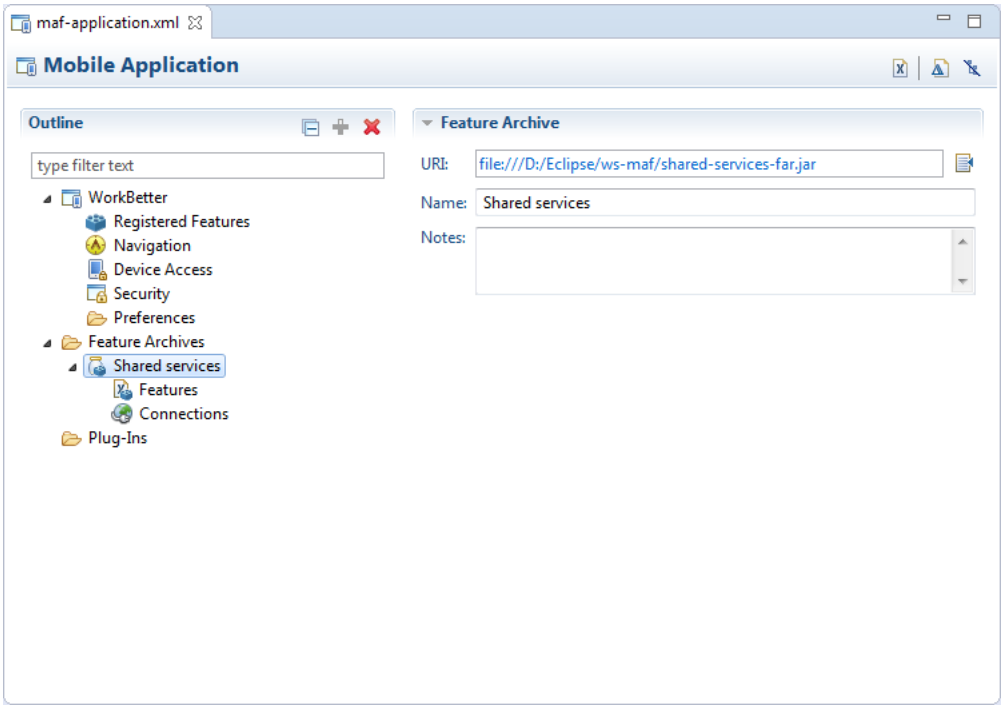
Figure 27–36 File Export Summary Page



27.6.3 What Happens When You Create a Feature Archive File

To use a FAR from an external application, you need to register the FAR in your application's `maf-application.xml` file. Figure 27–37 shows Feature Archives that can be made available to a mobile application through a file system connection.

Figure 27–37 Deployed Feature Archive JARs in the MAF Application Editor



27.7 Creating a Mobile Application Archive File

The MAF Application Archive (`.maa`) file format enables you to provide third-parties with an unsigned mobile application. By deriving a mobile application from an imported `.maa` file, you enable various customizations, which include:

- Giving an application a unique application ID (to enable push notifications, for example).
- Signing an application with a company-specific credential or certificate.
- Replacing the resources with customized splash screens and application icons.

You can create a new mobile application from a n existing mobile application by first packaging the original mobile application as a Mobile Application Archive (`.maa`) file

and then by deriving a new mobile application from this file. An .maa file can be used by third parties, as described in Section 27.8, "Creating Unsigned Deployment Packages."

An .maa file preserves the structure of the mobile application. Table 27–4 describes the contents of this file.

Table 27–4 Contents of a Mobile Application Archive File

Directory	Description
adf	Contains the META-INF directory, which contains the metadata files, including: ■ The adf-config.xml file ■ The maf-application.xml file ■ The maf-config.xml file ■ Other applicable application-level files, such as the connections.xmlfile
Projects	Contains a JAR file for each project in the workspace. For example, a ViewController.jar file and a ApplicationController.jar file are located in this directory when you deploy a default mobile application to an .maa file. The Projects directory of the .maa file does not include the .java files from the original project. Instead, the .java files are compiled and the resulting .class files are placed in a separate JAR file that is contained in the project JAR file (such as ApplicationController.JAR/classlib/mobileApplicationArchive.jar). The .maa file created in OEPE can be imported only in OEPE. It cannot be imported into OEPE.
ExternalLibs	Contains the application-level libraries (including FARs) that are external to the original mobile application.
META-INF	Includes the maf.properties and logging.properties files.
resources	Includes the following directories: ■ android—Contains Android-specific image files for application icons and splash screens. ■ ios—Contains iOS-specific image files for application icons and splash screens. ■ security—Includes the cacerts file (the keystore file).

In addition to the artifacts listed in Table 27–4, the .maa file includes any folder containing FARs or JAR files that are internal to the original mobile application. See also Section 27.8.2, "What Happens When You Import a MAF Application Archive File."

Note: Importing an .maa file into an existing application overwrites the workspace and project container files (the .jws and .jpr files, respectively). As a result, all prior changes to MAF AMX pages and configuration files, such as maf-application.xml, maf-config.xml, connections.xml, and adf-config.xml, will not be retained.

27.7.1 How to Create a Mobile Application Archive File

The MAF Application Archive (.maa) file format enables you to provide third-parties with an unsigned mobile application. By deriving a mobile application from an imported .maa file, you enable various customizations, which include:

- Giving an application a unique application ID (to enable push notifications, for example).
- Signing an application with a company-specific credential or certificate.

- Replacing the resources with customized splash screens and application icons.

You can create a new mobile application from a n existing mobile application by first packaging the original mobile application as a Mobile Application Archive (`.maa`) file and then by deriving a new mobile application from this file. A `.maa` file can be used by third parties, as described in Section 27.8, "Creating Unsigned Deployment Packages."

A `.maa` file preserves the structure of the mobile application. Table 27–4 describes the contents of this file.

Table 27–5 Contents of a Mobile Application Archive File

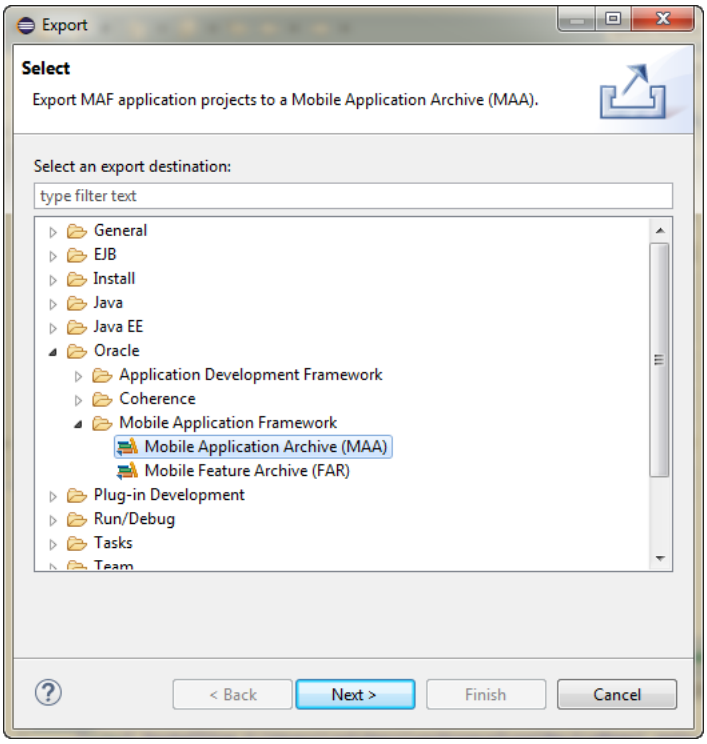
Directory	Description
`adf`	Contains the `META-INF` directory, which contains the metadata files, including: ■ The `adf-config.xml` file ■ The `maf-application.xml` file ■ The `maf-config.xml` file ■ Other applicable application-level files, such as the `connections.xml` file
`Projects`	Contains a JAR file for each project in the workspace. For example, a `ViewController.jar` file and a `ApplicationController.jar` file are located in this directory when you deploy a default mobile application to a `.maa` file. The Projects directory of the `.maa` file does not include the `.java` files from the original project. Instead, the `.java` files are compiled and the resulting `.class` files are placed in a separate JAR file that is contained in the project JAR file (such as `ApplicationController.JAR/classlib/mobileApplicationArchive.jar`). The `.maa` file created in OEPE can be imported only in OEPE. It cannot be imported into JDeveloper.
`ExternalLibs`	Contains the application-level libraries (including FARs) that are external to the original mobile application.
`META-INF`	Includes the `maf.properties` and `logging.properties` files.
`resources`	Includes the following directories: ■ `android`—Contains Android-specific image files for application icons and splash screens. ■ `ios`—Contains iOS-specific image files for application icons and splash screens. ■ `windows`—Contains iOS-specific image files for application icons and splash screens. ■ `security`—Includes the cacerts file (the keystore file).

In addition to the artifacts listed in Table 27–4, the `.maa` file includes any folder containing FARs or JAR files that are internal to the original mobile application. See also Section 27.8.2, "What Happens When You Import a MAF Application Archive File."

To package a mobile application as a MAF Application Archive file:

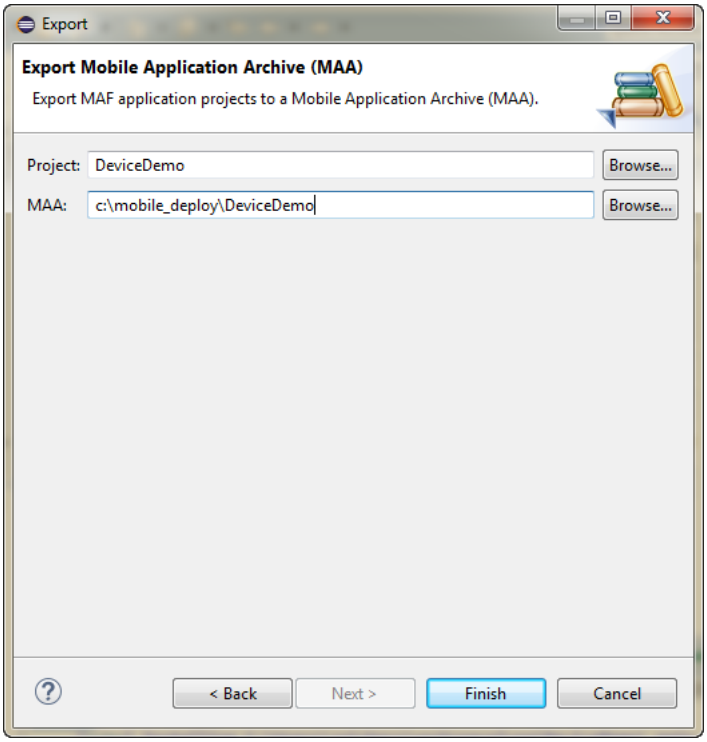
1. Click **File** then choose **Export**.
2. In the Export wizard, expand Oracle, then Mobile Application Framework, and choose Mobile Application Archive (MAA), as shown in Figure 27–38.

Figure 27–38 Creating a Mobile Application Archive File



3. Click **Next**, and choose the name of the application you want to archive and enter a name for the Mobile Application Archive, as shown in Figure 27–39.

Figure 27–39 Entering a Name and Path for the Mobile Application Archive File



4. Click **Finish**

27.8 Creating Unsigned Deployment Packages

The MAF Application Archive (`.maa`) file format enables you to provide third-parties with an unsigned mobile application. By deriving a mobile application from an imported `.maa` file, you enable various customizations, which include:

- Giving an application a unique application ID (to enable push notifications, for example).
- Signing an application with a company-specific credential or certificate.
- Replacing the resources with customized splash screens and application icons.

> **Note:** You cannot import an `.maa` file into a workspace that already contains one or more of the same projects. Instead, you must import the `.maa` file into a different workspace.

27.8.1 How to Create an Unsigned Application

You create an unsigned application by importing an `.maa` file into a new mobile application.

To create an unsigned application:

1. Choose **File** then choose **Import**.
2. In the Import wizard, expand Oracle and choose Mobile Application Archive (MAA). Click **Next**.
3. In the Import Mobile Application Archive (MAA) page, click **Browse** and navigate to the MAA file. Click **Next**.
4. In the Configure Deployment Targets page, select the targets you will want to deploy to, and click **Finish**.

27.8.2 What Happens When You Import a MAF Application Archive File

MAF performs the following after you import an `.maa` file:

1. Creates a project for each of the assembly project, the application project and the view projects.
2. Unpacks files from the `.maa` file.

27.9 Deploying with Oracle Mobile Security Suite

Oracle Mobile Security Suite (OMSS) provides enterprise-level security for mobile applications. It offers data leakage prevention and encryption of application data and database content through containerization at deployment time. For more information about containerizing your MAF application with OMSS, see Section 2.8, "Containerizing a MAF Application for Enterprise Distribution."

You can containerize MAF applications that you deploy to the iOS and Android platforms with OMSS. MAF applications that you deploy to the Universal Windows Platform cannot be containerized with OMSS.

Before you begin:

You must have the OMSS containerization tool installed. This tool is a command-line utility with the file name `c14n`. You can download OMSS and read instructions on installation on the Oracle Technology Network:

http://www.oracle.com/technetwork/middleware/id-mgmt/overview/default-2099033.html

---

Notes: For iOS, you must also have selected the name of the provisioning profile in the iOS Platform preferences page as described in Section 27.4.2, "Setting the Device Signing Options."

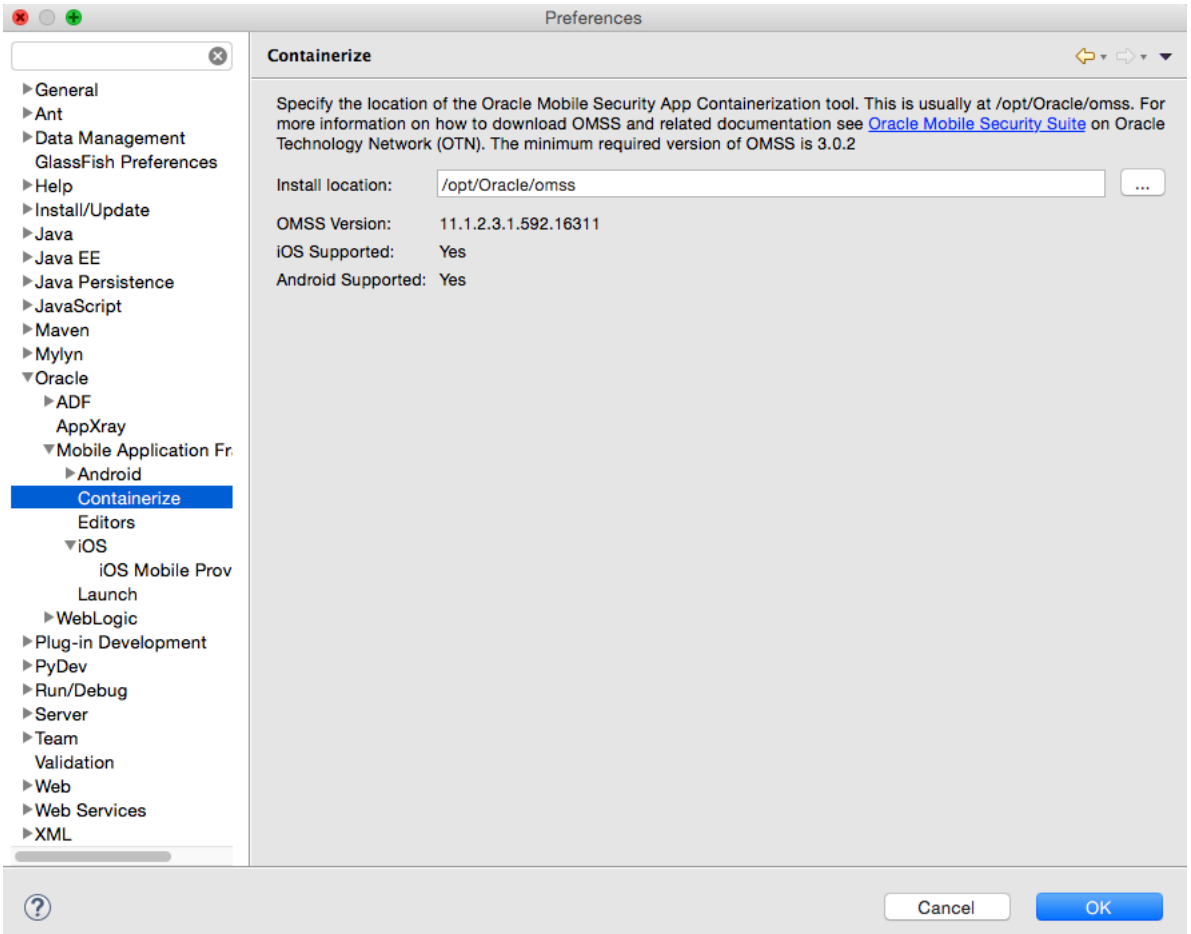
Containerization is not supported for Android or Windows.

OEPE will not allow you to deploy a mobile app containerized with OMSS to an emulator with default names.

---

1. Select **Run > Run Configurations**.
2. In the Target Configuration pane, select **MAF Application** and either create a new run configuration or select an existing run configuration.
3. At the bottom of the Main tab, in the Containerization pane, select **Enable Oracle Mobile Security Suite** and click **Containerize Tools Location**. This displays the Containerize dialog from the Preferences menu (see Figure 27–40).

Figure 27–40 Specifying the location of the Containerization tool



4. Click the **Browse** icon and browse to the location of the Oracle Mobile Security App Containerization tool on your local file system.

5. Click **OK** to select the containerization tool.
6. From the Run Configurations dialog, select **Run**.

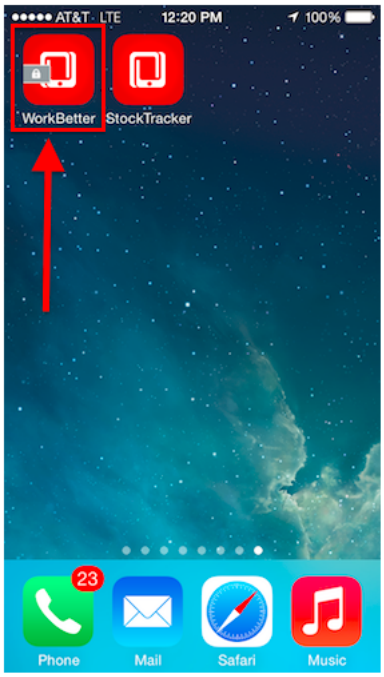
> **Note:** For iOS, containerization is supported for iTunes deployment only. Deployment to a distribution package or simulator will not create a containerized `.ipa` file.

A platform-specific file for Android) secured by Oracle Mobile Security Suite will be created:

- `.ipa` for iOS.
- `.apk` for Android. OMSS may create an intermediate `.h` header file which is compiled into the app.

After you add the containerized application to your device, it will display a lock icon, as shown in Figure 27–41.

Figure 27–41 Mobile Application Displaying the Lock Icon for Containerization



27.9.1 What Happens When You Containerize Your App with OMSS

When you deploy a MAF application containerized with Oracle Mobile Security Suite (OMSS), you can then upload it to the OMSS Mobile App Catalog, which displays a list of currently available apps. Before publishing the application to end users, the OMSS administrator applies various policies to the app to manage its functionality. By applying specific policies on the OMSS server side, the OMSS administrator can manage the security and sharing requirements for applications containerized with the Oracle Mobile Security App Containerization Tool. Data in transit and data stored locally inside containerized apps on the mobile device is encrypted. Encrypted data storage includes application data, including files, databases, application cache, and user preferences.

After applying OMSS data leakage protection and encryption, you can then make the app available to users from a download site.

> **Note:** iOS MAF apps that are containerized with OMSS can only be distributed to users through an internal download site or enterprise app store. Containerized iOS MAF apps cannot be uploaded to the Apple App store. This restriction is not applicable to Android MAF apps that are containerized with OMSS. These apps can be distributed through an internal download site, enterprise app store, or Google Play.

For details about how system administrators use the OMSS Mobile App Catalog to manage the MAF application provisioned to devices and workspaces, see the "Managing Devices and Workspaces" chapter in *Administering Oracle Mobile Security Suite*.

The following OMSS data leakage protection policies restrict how and if users can share data within an app:

- **Email allowed** can restrict the ability to send email from an app.
- **Instant Message allowed** can restrict the ability to send Instant Message from an app.
- **Video chat allowed** restricts the ability to share information via services such as FaceTime.
- **Social Share allowed** restricts the ability to share information via services such as Facebook or Twitter.
- **Print allowed** restricts the ability of the user to print.
- **Restrict file sharing** restricts the ability of the user to share files outside the secure enterprise workspace.
- **Restrict copy/paste** allows copy/paste inside the secure container, containerized apps or between containerized apps, but not to apps outside the secure enterprise workspace.
- **Redirects to container allowed** prevents any app outside the Mobile Security Container workspace from redirecting a URL into the container.
- **Save to media gallery allowed** prevents images, videos and audio files from being saved to media gallery and photo stores.
- **Save to local contacts allowed** prevents contacts inside secure enterprise workspace apps from being saved down to native device contacts app.
- **Redirects from container allowed** prevents any vApp from the Mobile Security Container workspace or containerized app from redirecting a URL outside the Mobile Security Container workspace or containerized app.

28

Understanding Secure Mobile Development Practices

Mobile Application Framework provides protection from common security risks identified by the Open Web Application Security Project (OWASP), which are described in the following sections:

- Section 28.1, "Weak Server-Side Controls"
- Section 28.2, "Insecure Data Storage on the Device"
- Section 28.3, "Insufficient Transport Layer Protection"
- Section 28.4, "Side-Channel Data Leakage"
- Section 28.5, "Poor Authorization and Authentication"
- Section 28.6, "Broken Cryptography"
- Section 28.7, "Client-Side Injection From Cross-Site Scripting"
- Section 28.8, "Security Decisions From Untrusted Inputs"
- Section 28.9, "Improper Session Handling"
- Section 28.10, "Lack of Binary Protections Resulting in Sensitive Information Disclosure"

28.1 Weak Server-Side Controls

Build security into a mobile application. Even in the earliest stages of designing a mobile application, you must assess not only the risks that are unique to mobile applications, but also those that are common to the sever-side resources that the mobile application accesses. Like their desktop counterparts, mobile applications can be made vulnerable by attacks on the backend services that store their data. Because this risk is not unique to mobile applications, the standards described by the OWASP Top Ten Project (`https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`) also apply when you create mobile applications. Because client applications running on mobile devices can be vulnerable, do not use them to enforce access control. Because this function should be performed by the server-side application, MAF does not provide anything out-of-the box for validating data sent from the client. You must ensure that the data intended for a mobile application is valid. For more information, see the following:

- "Understanding Web Service Security Concepts" in *Oracle Fusion Middleware Understanding Oracle Web Services Manager*

- "Web Service Security Standards" in *Oracle Fusion Middleware Understanding Oracle Web Services Manager*
- *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services*

28.2 Insecure Data Storage on the Device

Shortcomings in a mobile application's design can make local files accessible to users, thereby exposing sensitive data stored on a device's local file system. This data may include usernames and passwords, cookies, and authentication tokens. Although most users may not be aware that their data is vulnerable—or that it is even stored on the device itself—a malicious user could exploit this situation by having the tools to open the local database and view credentials. When assessing the security requirements for an application, you should assume the likelihood of a phone falling into the wrong hands. MAF provides the API to secure data stored on the device by encrypting the device database and local data stores.

28.2.1 Encrypting the SQLite Database

MAF's embedded SQLite database protects locally stored data. MAF applications do not share the SQLite database; the application that creates the database is the only application that can access it. Further, only users with the correct username and password can access this database. The `AdfmfJavaUtilities` class enables you to create keys to secure the password for this database and also to encrypt the data stored within it. To provide a secure key to the database, the `AdfmfJavaUtilities` class includes the `GeneratedPassword` utility class that generates a strong password and then stores it securely. The `AdfmfJavaUtilities` class also provides the `encryptDatabase` method for encrypting the database with a password. For general information about the SQLite database, see Chapter 19, "Using the Local Database in MAF AMX" For more information on the `GeneratedPassword`, the `encryptDatabase` (and its counterpart, `decryptDatabase`), see *Java API Reference for Oracle Mobile Application Framework* and Section 19.2.7, "How to Encrypt and Decrypt the Database." For a sample application, see the StockTracker sample in the `PublicSamples.zip`, as described in Appendix E, "MAF Sample Applications."

Note: Always use the `GeneratedPassword` utility. Do not hard-code the key.

28.2.2 Securing the Device's Local Data Stores

You can store files in the local file system programmatically on both the iOS and Android platforms using the `adfmfJavaUtilties` class' `getDirectoryPathRoot` method. Using this method provides agnostic access to store application data on the device. The following options are available for this method:

- Temporary directory
- Application directory
- Cache directory
- Download directory

Tip: When users synchronize their devices to their desktop computers, the data stored in the device's Application directory is transferred to the desktop system where it can be exposed. Store data in the Temporary directory. For iOS, data stored in the temporary directory is not synchronized with the desktop when the device is synchronized using iTunes.

For any files that require security, you can encrypt and decrypt them using the Java cryptographic APIs (`javax.crypto`). For more information on the `javax.crypto` package, see Java Platform, Standard Edition 1.4 API. For more information, refer to *Java API Reference for Oracle Mobile Application Framework* and Section B.3, "Accessing Files Using the getDirectoryPathRoot Method." See also the "File System Basics" section in *File System Programming Guide*, available from the iOS Developer Library (`https://developer.apple.com/library/`).

28.2.3 About Security and Application Logs

Ensure that no sensitive data can be written to log files because they can be viewed if the device is synchronized with a desktop computer. When users connect their iOS devices to a desktop system to synchronize data, the application log files are ultimately stored on the desktop in an unencrypted format. Log files synchronized from Android devices can be viewed using the Android Device Monitor tool. See also Section 28.4, "Side-Channel Data Leakage."

28.3 Insufficient Transport Layer Protection

Mobile applications may use SSL/TLS when accessing data over a provider network, or neither of these protocols if they use WiFi. Because provider networks can be hacked, never assume that they are safe. You should therefore enforce SSL when the application transports sensitive data and validate that all certificates are legitimate and signed by public authorities.

Because all of the endpoints used by a mobile application must be secured with SSL, MAF provides a set of web service policies that support SSL. For a list of policies that you may associate with REST web services, see Section 15.3, "Accessing Secure Web Services."

MAF provides a `cacerts` file seeded with entries of known and trusted Certificate Authorities. Application developers can add other certificates to this file, if needed. For more information, see Section 29.11, "Configuring a MAF Application to Enable Two-Way SSL for Authentication."

28.4 Side-Channel Data Leakage

Unintended data leakage can originate from such sources as:

- Disabling screen shots (backgrounding) -- iOS and Android take screen shots of the application before backgrounding the application for improving perceived performance of the application reactivation. However, these screen shots are a cause of security concern due to the potential leak of customer data.
- Key stroke logging -- On iOS and Android, some of the information entered via keyboard is automatically logged in the application directory for use with type-ahead capabilities. This feature could lead to potential leaks of customer data.

- Debugging messages -- Applications can write sensitive data in debugging logs. Setting the logging level to FINE results in log messages being written for all of the data transmitted between the user's device and the server.
- Disable clipboard copy and open-in functionality for sensitive documents displayed as part of the application. MAF currently does not provide the capability to disable copy and open-in functionality and is being targeted for a future release.
- Temporary directories -- They may contain sensitive information.
- Third-party libraries -- These libraries (such as ad libraries) can leak user information about the user, the device, or the user's location.

To prevent data leakage:

- Do not log credential, personally identifiable information (PII), or other sensitive data to the application log. Store all sensitive information in the native keychain or an encrypted database or file system.
- When debugging an application, review any files that are created and anything written to them.
- Remove debugging messages before publishing the application.

28.5 Poor Authorization and Authentication

Weak authentication mechanisms and client-side access control both compromise security.

Although it may be easier for end users to authenticate a device using a phone number or some type of identifier (IMEI, IMSI, or UUID) rather than a user name and password, these identifiers can easily be discovered through brute force attacks and should never be used as a sole authenticator. Mobile applications must instead use strong credentials when accessing sensitive data. The authentication should reflect the user, not the device. Further, you can enhance authentication by using contextual identifiers (such as location), voice, fingerprints, or behavioral information.

A developer can use either the default login page provided by MAF or a custom login page that they create. For more information, see Section 29.6.2, "How to Designate the Login Page."

All features in a MAF application that require secure access must enable security, as described in Section 29.6.1, "How to Enable Application Features to Require Authentication."

Additionally, access control must be enforced by the server, not the client. Locating this function on the client mobile application is less secure. Access Control Service (ACS) allows developers to use roles/privileges defined on the server to enforce access control in the mobile application. Access Control Service is a RESTful service that could be implemented by application developers to filter the user roles/privileges that are valid for the application. While an application may support thousands of user roles, the service only returns the roles that you designate for the mobile application. For more information, see Section 29.5.18, "How to Configure Access Control."

28.6 Broken Cryptography

Encryption becomes fallible because:

1. Applications use broken implementations or use known algorithms improperly.

2. Data is insecure because of easily defeated cryptography.

In addition, Base-64 encoding, obfuscation, and serialization are not encryption (and should not be mistaken for encryption).

To encrypt data successfully:

- Do not store the key with the encrypted data.
- Use the platform-specific file encryption API or another trusted source. Do not create your own cryptography.

In addition to securing the embedded SQLite database using the encryption methods mentioned in Section 28.2, "Insecure Data Storage on the Device." Also, apply SSL to create secure web service calls as described in Section 28.3, "Insufficient Transport Layer Protection." MAF uses Oracle Access Manager for Mobile and Social IDM SDK for secure handling of credentials.

28.7 Client-Side Injection From Cross-Site Scripting

Because mobile applications draw content and data from many different sources, they can be vulnerable to Cross-Site Scripting (XSS) injections, which co-opt the user session. MAF protects against XSS through encoding.

In addition to injection attacks, mobile applications are vulnerable to Cross-Site Request Forgery (CSRF), where a malicious page performs an unintended action in a targeted application on behalf of a user through the cookies cached in a web browser that store user identity. Application sandboxing addresses CSRF concerns.

Also consider disabling application features (particularly application features with Remote URL content) access to the native container. You can prevent selected application features within a MAF application from accessing the native container. For example, your MAF application includes an application feature that references remote content from a web application that you do not trust (Remote URL content application feature). In this scenario, you prevent this specific application feature from accessing the native container, as shown in the following example:

```
<adfmf:featureReference refId="remoteAppfeature1" id="fr1"
allowNativeAccess="false"/>
```

The default value of the `allowNativeAccess` property is `true`.

28.7.1 Protecting MAF Applications from Injection Attacks Using Device Access Permissions

The URIs that can access data stored on the user's device and its various device capabilities, such as its camera or address book. Such access is not granted by default; as described in Section 9.2, "Enabling a Core Plugin in Your MAF Application," you can configure a MAF application to limit the device's capabilities that a URI can access to any of the following:

- open network sockets (must be granted when user authentication is configured)
- GPS and network-based location services
- contact
- e-mail
- SMS
- phone

- push notifications
- locally stored files

Tip: You can programmatically protect users against such security risks as fake login pages injected by XSS through the `updateSecurityConfigWithURLParameters` method, which detects changes in the login configuration and then prompts users to confirm the change by re-authenticating, as described in Section 29.5.6, "How to Update Connection Attributes of a Named Connection at Runtime." Additionally, MAF informs users whenever they open a secured application feature. Authentication can be deferred when the default application does not participate in security. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

28.7.2 About Injection Attack Risks from Custom HTML Components

Using HTML to create a custom user interface component in a MAF AMX page may leave an application open to an injection attack. MAF provides two components for HTML content in MAF AMX pages: the `<amx:verbatim>` component and the `<amx:outputHTML>` component. Because the `<amx:verbatim>` component does not allow dynamic HTML, it is not susceptible to an injection attack. However, the `<amx:outputHTML>` component, which delivers dynamic HTML content through an EL binding, may be vulnerable when you configure its `security` attribute to `none`. By default, this attribute is set to `high` to enable the framework to escape various HTML tags and remove JavaScript, such as an `onClick` event. Because setting it to `none` enables iFrame components and JavaScript (which allows AJAX requests within the AMX page), you must ensure that the HTML and JavaScript are properly encoded. For more information, see Section 12.3.18, "How to Use Verbatim Component" and Section 12.3.19, "How to Use Output HTML Component." See also Section 28.8, "Security Decisions From Untrusted Inputs."

28.7.3 About SQL Injections and XML Injections

Mobile applications are vulnerable to SQL injections, which can enable an attacker to read the data stored in the embedded SQLite database.

To prevent SQL injections:

- Application developers are required to validate and encode all data stored in the local database.
- Application developers are expected to encode and validate XML and HTML content processed by the application.

28.8 Security Decisions From Untrusted Inputs

On both iOS and Android platforms, applications (such as Skype) may not always request permissions from outside parties, providing an entry point for attackers that may result in malicious applications circumventing security. As a result, applications are vulnerable to client-side injection and data leakages. Always prompt for additional authorization or provide additional steps to launch sensitive applications when additional authorization is not possible.

You must ensure that all of the data that the application receives from (or sends to) an untrusted third-party application can be subject to input validation. The client side XML input to the application must be encoded and validated. Although MAF AMX components can validate user input, data must be validated on the server, which

should never trust the data it receives from a client. In other words, the server is responsible for ensuring that the XML, JSON, and JavaScript that is sent back and forth between it and the client is properly encoded.

When you configure the URL scheme that launches a MAF application from another application, you must validate the parameters sent through the URL to ensure that no malicious data or URIs can be passed to the MAF application. For more information, see Section 4.10, "Using Custom URL Schemes in MAF Applications." See also Section 28.1, "Weak Server-Side Controls."

About JSON Parsing

Use MAF's JSON encoding API where possible. For scenarios requiring custom JSON composition, be careful when composing JSON with user-entered data. For more information about processing JSON data, see the *Java API Reference for Oracle Mobile Application Framework*.

28.9 Improper Session Handling

Usability requirements for mobile applications often require sessions to last for long periods. Mobile applications use cookies, SSO services, and OAUTH tokens for session management.

> **Note:** OAuth access tokens can be revoked remotely.

To enable proper session handling:

- Configure session timeout in the Login Server connection to a value less than server-side session timeout.

  Do not use a device ID as a session token because it never expires. An application should expire tokens, even though doing so forces users to re-authenticate.

- Ensure that proper best practices (see OWASP Top Ten Project, `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`) are followed for token generation on the server.

  Do not use session tokens that can be easily guessed or are poorly generated. A session token should be unpredictable and have high entropy.

  Oracle Identity Management (IDM) stack provides support for standards-based tokens (such as, OAuth Access Token, JWT Token) for use with mobile applications. MAF provides out of the box support for Oracle IDM OAuth server and Oracle recommends using such standards-based authentication mechanisms with MAF applications.

As described in Section 29.5.3, "How to Configure Basic Authentication," configuring an application that requires users to authenticate against a login server includes options to set the duration of the session and idle timeouts. By default, the duration of an application feature session lasts eight hours. The default time for an application feature to remain idle is five minutes. MAF expires user credentials when either of the configured time periods expire and prompts users to re-authenticate.

28.10 Lack of Binary Protections Resulting in Sensitive Information Disclosure

Using reverse engineering, attackers can discover such sensitive data as API keys, passwords, and sensitive business logic. To protect this information:

- Store API keys and sensitive business logic on the server.
- Do not store passwords in the application binary.
- Never hard-code a password. Instead, use the `GeneratedPassword` utility described in Section 28.2, "Insecure Data Storage on the Device."
- Because log files can be monitored, ensure that applications do not write sensitive information to the log files. See also Section 28.4, "Side-Channel Data Leakage."
- Keep in mind that information stored on a file system (that is, stored externally from the mobile application). Store sensitive data in an encrypted database or file system, or in the native keychain. See also Risk 1: Insecure Data Storage on the Device.

29

Securing MAF Applications

This chapter provides an overview of the security framework within MAF and also describes how to configure MAF applications to participate in security.

This chapter includes the following sections:

- Section 29.1, "Introduction to MAF Security."
- Section 29.2, "About the User Login Process"
- Section 29.3, "Overview of the Authentication Process for Mobile Applications"
- Section 29.4, "Overview of the Authentication Process for Containerized MAF Applications"
- Section 29.5, "Configuring MAF Connections"
- Section 29.6, "Configuring Security for Mobile Applications"
- Section 29.7, "Allowing Access to Device Capabilities"
- Section 29.8, "Enabling Users to Log Out from Application Features"
- Section 29.9, "Using MAF Authentication APIs"
- Section 29.10, "Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL"
- Section 29.11, "Configuring a MAF Application to Enable Two-Way SSL for Authentication"

29.1 Introduction to MAF Security

MAF presents users with a login page when a secured application feature has been activated. For example, users are prompted with login pages when an application feature is about to be displayed within the web view or when the operating system returns an application to the foreground. MAF determines whether access to the application feature requires user authentication when an application feature is secured by an authentication server, or when it includes constraints based on user roles or user privileges. Only when the user successfully enters valid credentials does MAF render the intended web view, UI component, or application page.

While the presence of these conditions in any of the application features can prevent users from accessing a mobile application without a successful login, you can enable users to access a mobile application that contains both secured and non-secured application features by including a default application feature that is neither secured nor includes user access-related constraints. In this situation, users can access the MAF application without authentication. The default application feature provides the entrance point to the mobile application for these anonymous users, who can both

view non-secured data and authenticate against the remote server when accessing a secured application feature. You can designate a non-secure default application feature by:

- Allowing anonymous users access to public information through the default application feature, but only enabling authorized users to access secured information.
- Allowing users to authenticate only when they require access to a secured application feature. Users can otherwise access the mobile application as anonymous users, or login to navigate to secured features.
- Allowing users to log out of secured application features when secured access is not wanted, thereby explicitly prohibiting access to secured application features by unauthorized users.

> **Note:** MAF enables anonymous users because the application login process is detached from the application initialization flow; a user can start a mobile application and access unsecured application features as an anonymous user without having to provide authentication credentials. In such a case, MAF limits the user's actions by disabling privileged UI components.

For more information, see Section 29.6.1, "How to Enable Application Features to Require Authentication," and Section 22.2.4, "About User Constraints and Access Control."

A MAF application uses either the default page or a customized login page that is written in HTML.

Application features defined with `user.roles` or `user.privileges` constraints can be accessed only by users who have been granted the specific role and privileges. When users log into such an application feature, a web service known as the Access Control Service (ACS) returns the user objects that grant them access to this application feature. For more information about ACS, see Section 29.5.19, "What You May Need to Know About the Access Control Service."

The developer can elect to containerize the MAF application at the time of deployment. Containerization allows the application to utilize the Secure Networking and Network Tunneling capabilities of the enterprise networking platform configured with Oracle Mobile Security Suite (OMSS). Distribution of containerized applications is managed by OMSS administrators through the Oracle Mobile Security Access Server (MSAS) component App Catalog and Secure Workspace container features. Authentication of users is performed by MSAS configured for the desired authentication type. For more information about developing the MAF application and OMSS containerization, see Section 2.8, "Containerizing a MAF Application for Enterprise Distribution."

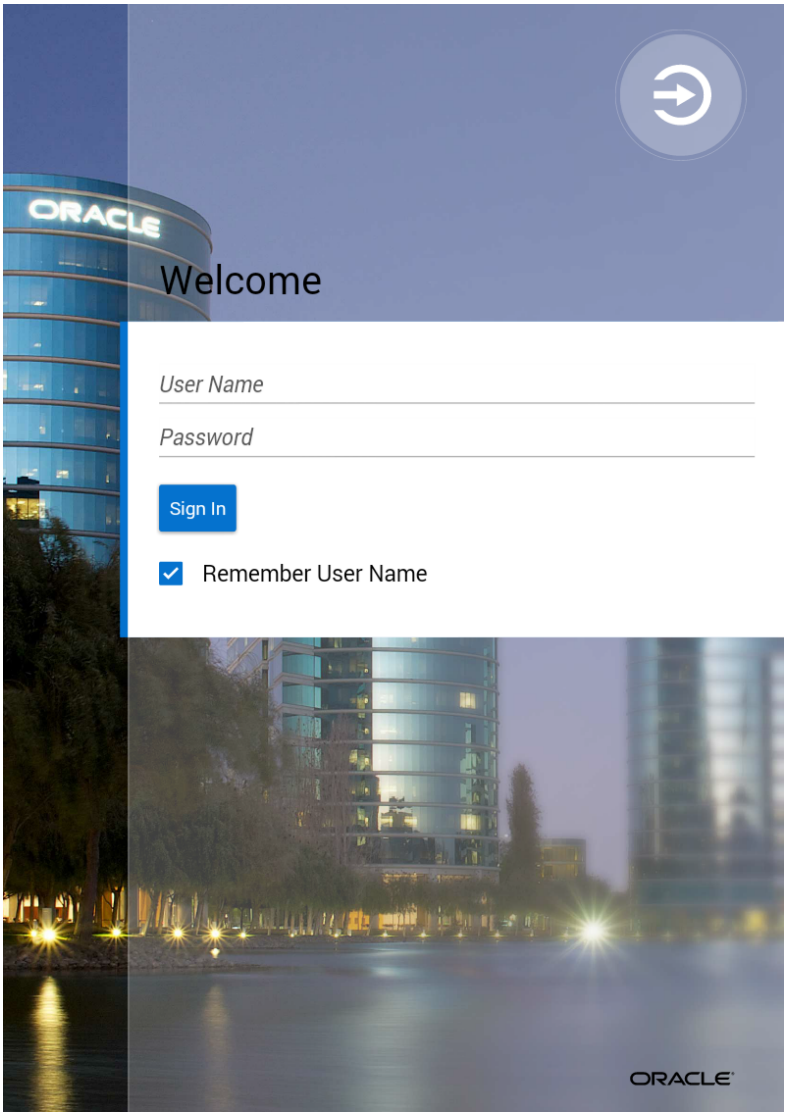
29.2 About the User Login Process

From the end-user perspective, the login process is as follows:

> **Note:** For more information about how containerizing the MAF application changes the login process, see Section 29.4, "Overview of the Authentication Process for Containerized MAF Applications."

1. MAF presents a web view of a login page, shown in Figure 29–1 whenever the user attempts to access an application feature that is secured. If the secured application feature is the default, then MAF prompts users with the login page when they launch the mobile application.

Figure 29–1 The Default Login Page



Note: As described in Section 29.6.4.2, "The Custom Login Page," MAF provides not only a default login page, but also supports the use of a custom login page.

2. The user enters a user name and password and then clicks **Sign In**.

Note: MAF allows multiple users for the same application. Users may freely log in to an application after a previous user logs out.

3. If the user name and password are verified, MAF displays the intended web view, page, or user interface component.
4. MAF presents challenges to the user name and password until the user logs in successfully. When users cannot login, they can only navigate to another application feature.

> **Note:** Authentication times out when a predefined time period has passed since the last activation of an application feature. MAF only renews the timer for the idle timeout when one of the application features that uses the connection to the authentication server has been activated.

29.3 Overview of the Authentication Process for Mobile Applications

MAF applications may require that user credentials be verified against a remote login server (such as the Oracle Access Manager Identity Server used by Oracle ADF Fusion web applications) or a local credential store that resides on the user's device. To support local and remote connectivity modes, MAF supports these authentication protocols:

- HTTP Basic
- OAuth
- Web SSO

> **Note:** At present, MAF applications that you deploy to the Universal Windows Platform support HTTP Basic only.

By default, authentication of the MAF application user is against the remote login server regardless of the authentication protocol chosen at design time. Developers may configure the application in the case of basic authentication to enable local authentication. However, initially, because the local credential store is not populated with credentials, login to access secured application features requires authentication against a remote login server. Successful remote authentication enables the subsequent use of the local credential store, which houses the user's login credentials from the authentication server on the device. Thus, after the user is authenticated against the server within the same application session (that is, within the lifecycle of the application execution), MAF stores this authentication context locally, allowing it to be used for subsequent authentication attempts. In this case, MAF does not contact the server if the local authentication context is sufficient to authenticate the user. Although a connection to the authentication server is required for the initial authentication, continual access to this server is not required for applications using local authentication.

> **Tip:** While authentication against a local credential store can be faster than authentication against a remote login server, Oracle recommends authentication using OAuth or Web SSO authentication protocols, which only support remote connectivity.

Table 29–1 summarizes the login configuration options of a MAF application. The connectivity mode depends on the chosen authentication protocol.

Table 29–1 MAF Connectivity Modes and Supported Authentication Protocols

Connectivity Mode	Support Protocols	Mode Description
`local`	▪ `HTTP Basic`	Requires the application to authenticate against a remote login server only when locally stored credentials are unavailable on the device. The initial login is always against the remote login server. After the initial successful login, MAF persists the credentials locally within a credential store in the device. These credentials will be used for subsequent access to the application feature. See also Section 29.5.17, "What You May Need to Know about Web Service Security."
`remote`	▪ `HTTP Basic` ▪ `OAuth` ▪ `Web SSO`	Requires the application to authenticate against a remote login server, such as Oracle Access Manager (OAM) Identity Server or a secured web application. Authentication against the remote server is required each time a user logs in. If the device cannot contact the server, then a user cannot access the secured MAF feature despite a previously successful authentication.
`hybrid`	▪ `HTTP Basic`	Requires the application to authenticate against a remote login server when network connectivity is available, even when local credentials are available on the device. Only when a lack of network connectivity prevents access to the login server will local credentials on the device will be used.

For information about how containerizing the MAF application changes the authentication process, see Section 29.4, "Overview of the Authentication Process for Containerized MAF Applications."

29.4 Overview of the Authentication Process for Containerized MAF Applications

When you develop the MAF application, you must define a login connection to develop and test secure features. During testing, the MAF login page will be used to authenticate before accessing the protected resources. At deployment time, you may choose to containerize the MAF application to utilize the Secure Networking and Network Tunneling capabilities of the enterprise networking platform configured with Oracle Mobile Security Suite (OMSS) and thus eliminates the need for mobile VPN.

Post deployment of the containerized MAF application, access to backend resources behind the corporate firewall will rely on the Mobile Security Access Server (MSAS), a component of OMSS, to provide a central access point for securing traffic from mobile devices to corporate resources. In this case, the MSAS instance is configured to enforce an authentication endpoint to use for the initial authentication of the user.

User authentication is handled by container Single Sign-On (SSO) integration provided by MSAS to the registered MAF application. To allow the MAF application to communicate with MSAS, the user installs and registers a Secure Workspace app for the type of authentication that has been configured for the MSAS instance. Then when the user attempts to access a protected resource, the MAF login page will be suppressed and MSAS will present its own login through the Secure Workspace app on the user's mobile device.

Note: To enable authentication by MSAS and to utilize the AppTunnel, the MSAS instance must be configured to proxy the login URL or authentication endpoint that the MAF application defines. For details about how MSAS is configured for a MAF login connection, see Section 29.5.11, "What You May Need to Know About Login Connections and Containerized MAF Applications."

Whether backend resource requests are proxied using the MSAS AppTunnel is determined by a MSAS-generated Proxy Auto-Configuration file used by the MAF application and Secure Workspace app. The AppTunnel is a mutually authenticated SSL tunnel from each Secure Workspace app that provides secure access to the containerized MAF application. The AppTunnel encrypts all data in transit and provides protection from rogue apps on a user's mobile device that device-level mobile VPNs are subject to.

For an overview of OMSS support for containerized MAF applications, see Section 2.8, "Containerizing a MAF Application for Enterprise Distribution."

29.5 Configuring MAF Connections

You must define at least one connection to the application login server for an application feature that participates in security. The absence of a defined connection to an application login server results in an invalid configuration. As a result, the application will not function properly.

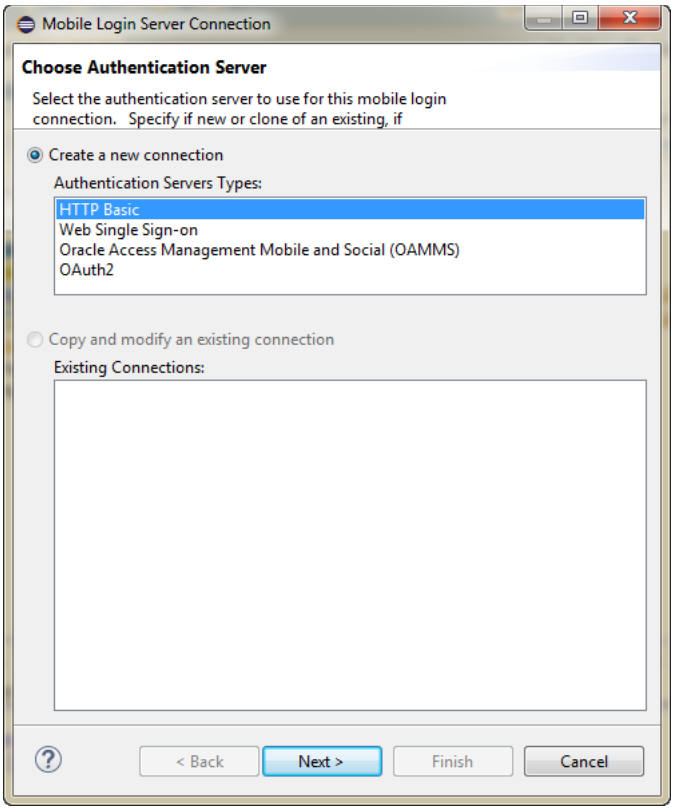
29.5.1 How to Create a MAF Login Connection

As Figure 29–2 shows, you can use the Mobile Login Server Connection wizard to select the connection type and, depending on the connection type, enable both local and remote authentication (hybrid). Depending on application requirements, you can configure a connection to servers that support the following authentication protocols:

- HTTP Basic
- OAuth
- Web SSO

Note: Oracle recommends that you configure a connection to the login server using the OAuth or Web SSO connection type. OAuth and Web SSO require authentication against a remote login server and do not allow users to authenticate on the device from a local credential store.

Figure 29–2 Configuring Authentication



To create a login server connection:

1. In the Project Explorer, expand the assembly project, then MAF and double-click MAF Application Editor.

   Note: The MAF Application Editor lets you define the application login server connection and assign it to the default application feature (if the default application feature is secured). In this case, credentials specified for the application login server are also used to retrieve user and roles and services through the Access Control Service (ACS). See also Section 29.5.19, "What You May Need to Know About the Access Control Service."

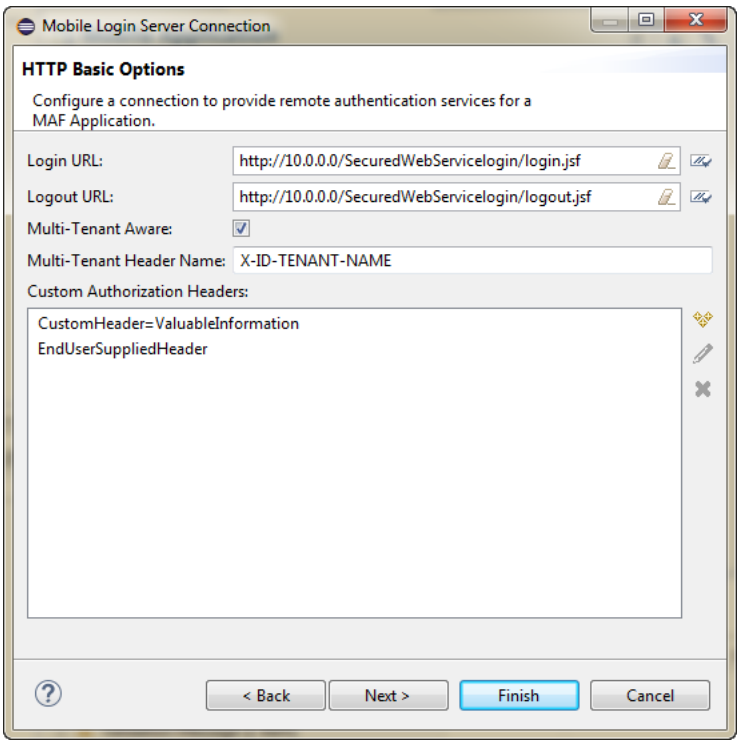
2. In the editor, select **Security** in the outline, and then click **Create** next to one of:
   - Default Login Server
   - Login Page
3. In the Mobile Login Server Connection wizard, select Create a new connection and choose the authentication server type, as shown in Figure 29–2. The connection types are described in the following sections.

29.5.2 How to Create a Multi-Tenant Aware MAF Login Connection

As Figure 29–3 shows, you can use the options page of the Mobile Login Server Connection wizard to create a MAF application connection that supports the notion of multi-tenancy, where an application includes a hosted application feature that can be

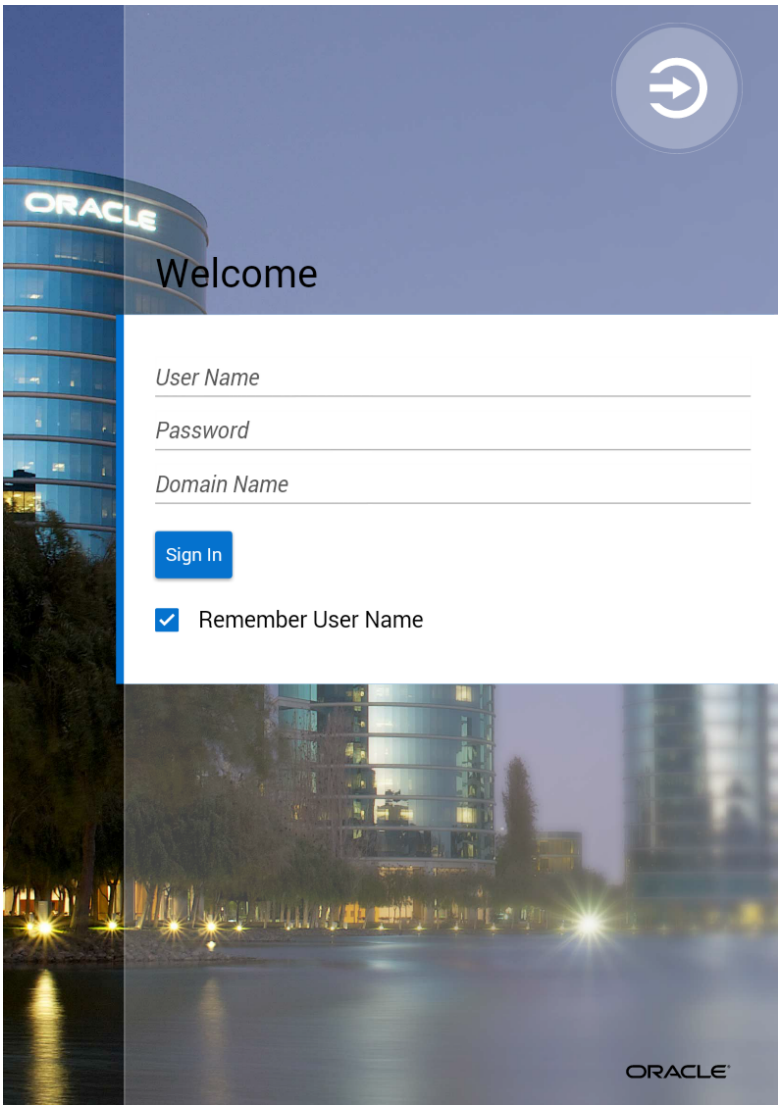
shared by different organizations (tenants), but can appear as though it is owned by a particular tenant. You can configure a multi-tenant aware connection to servers that support the HTTP Basic authentication protocol.

Figure 29–3 Configuring a Multi-Tenant Aware Connection



As Figure 29–4 shows, the default login page displayed by the MAF application with a multi-tenant aware connection defined, will prompt the user to enter the domain ID to propagate the tenant value on the HTTP Request.

Figure 29–4 Default Login Page for Multi-Tenant Aware Connection



To create a multi-tenant aware login server connection:

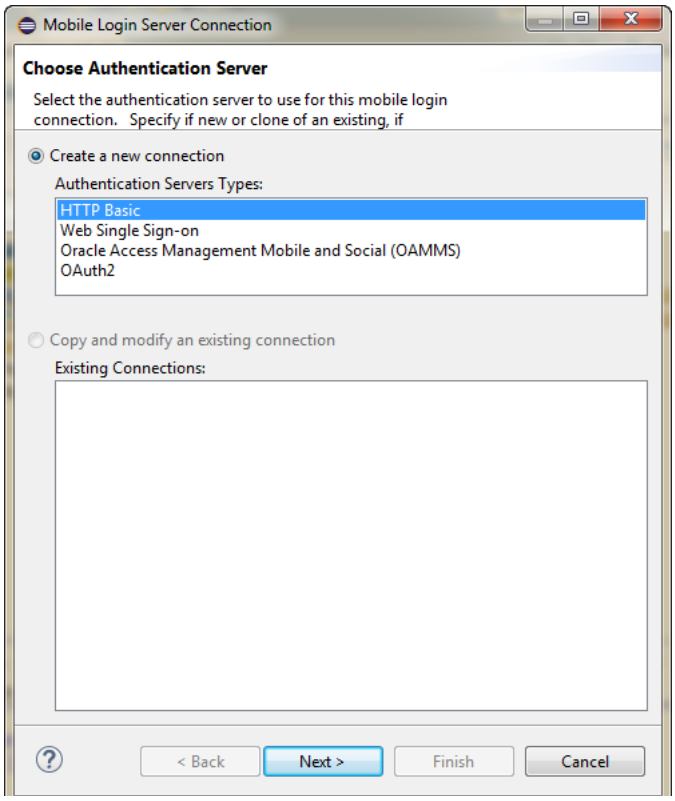
1. In the Mobile Login Server Connection wizard, choose an Authentication Server Type that supports multi-tenant login.
2. In the options page, select **Multi-Tenant Aware** and enter the tenant header name expected by the authentication server. For example, to solicit the tenant ID from the end user during login, enter the multi-tenant header name: `X-ID-TENANT-NAME`. As Figure 29–4 shows, the default login page will prompt the user to enter the domain name.
3. Optionally, enter the name of additional custom headers required to perform authentication. These may be configured in addition to the multi-tenant header. See also Section 29.5.14, "What You May Need to Know About Custom Headers."
4. If the header value of the custom header is known, enter the value. If the value for the named custom header is to be overridden during login, leave the corresponding Value field empty. When you want to provide the header value at runtime, you must set the value programmatically using the `OverrideConnectionHandler` API. For information about using the API to

configure headers, see Section 29.5.21, "How to Configure Login Credentials Programmatically Prior to Authentication."

29.5.3 How to Configure Basic Authentication

As Figure 29–5 shows, you can select the **HTTP Basic** authentication server type in the Mobile Login Server wizard to configure a connection for basic authentication.

Figure 29–5 Configuring Basic Authentication



To configure basic authentication:

1. In the Mobile Login Server Connection wizard, choose **HTTP Basic** as the authentication server type. Click **Next**.

   For information about opening the Mobile Login Server Connection wizard, see Section 29.5.1, "How to Create a MAF Login Connection."

2. In the General Options page, define the following:

   - **Name**—Enter a name for the connection.
   - **Authentication Mode**—Select the type of authentication, as described in Table 29–1.
   - **Idle Timeout**—Enter the time for an application feature to remain idle after MAF no longer detects the activation of an application feature. After this period expires, the user is timed-out of all the application features that are secured by the login connection. In this situation, MAF prompts users with the login page when they access the feature again. By default, MAF presents the login page when an application remains idle for 300 seconds (five minutes).

Note: MAF authenticates against the local credential store after an idle timeout, but does not perform this authentication after a session timeout.

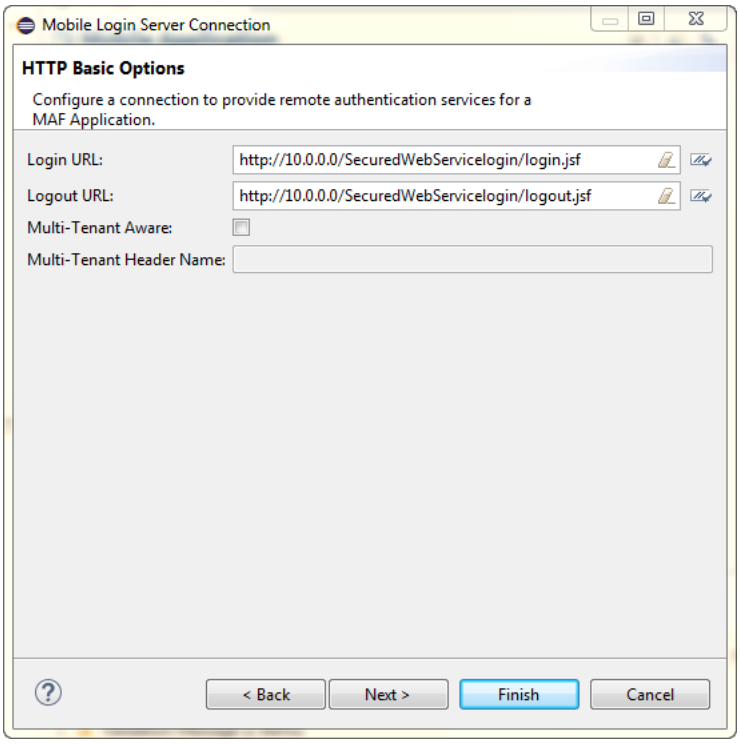
- **Session Timeout**—Enter the time, in seconds, that a user can remain logged in to an application feature. After the session expires, MAF prompts the user with a login page if the idle timeout period has not expired. By default, a user session lasts for 28,800 seconds (eight hours).
- **Include basic authentication header in HTTP requests**—Select this option to enable MAF to add a basic authentication header into the HTTP requests made from a web view. Basic authentication is the default request method used by MAF. A basic authentication header is injected only when the login connection type is HTTP Basic. See also Section 29.5.16, "What You May Need to Know About Injecting Basic Authentication Headers."
- **Realm**—If you define an authentication realm for the user name and password, OEPE populates the **connections.xml** file with a defined `<realm>` element which can be reused. For more information, see Section 29.5.16, "What You May Need to Know About Injecting Basic Authentication Headers.". Pages in the same realm share credentials. If your credentials work for a page with the realm "My Realm", the same username and password combination will work for another page with the same realm.

Click **Next**

3. In the HTTP Basic Options page, enter the following, as shown in Figure 29–6.
   - **Login URL**—Enter the login URL for the authentication server.
   - **Logout URL**—Enter the logout URL for the authentication server.
   - **Multi-Tenant Aware**—MAF supports the notion of multi-tenancy, where a mobile application includes a hosted application feature that can be shared by different organizations (tenants), but can appear as though it is owned by a particular tenant. You can define multi-tenancy awareness for the mobile application connection by selecting this option. See also Section 29.5.9, "What Happens When You Create a Multi-Tenant Aware Connection."
   - **Multi-Tenant Header Name**—When Multi-Tenant Aware is selected, provide the header name.
   - **Custom Authorization Headers**—Enter the name of any custom headers required to perform authentication. See also Section 29.5.14, "What You May Need to Know About Custom Headers."

     If the header value of the custom header is known, enter the value. If the value for the named custom header is to be overridden during login, leave the corresponding **Value** field empty. When you want to provide the header value at runtime, you must set the value programmatically using the `OverrideConnectionHandler` API. For information about using the API to configure headers, see Section 29.5.21, "How to Configure Login Credentials Programmatically Prior to Authentication."

Figure 29–6 Configuring Basic Authentication

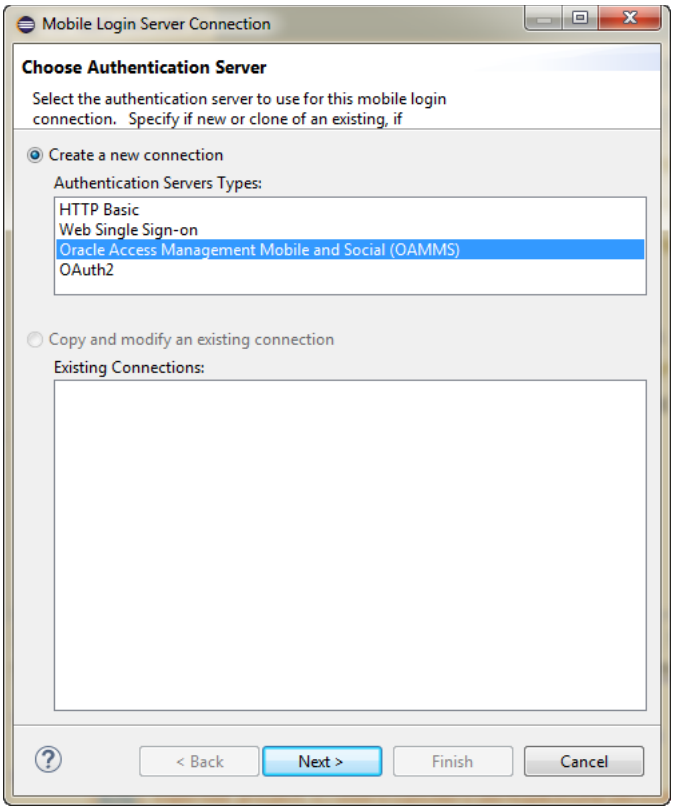


4. Click **Test** to test the connection to the authentication server. The results are displayed on the wizard.
5. Click **Next** to open the Login Options page and configure the parameters as described in Section 29.5.7, "How to Store Login Credentials."
6. Click **Next** to open the Authorization Options page and configure the parameters as described in Section 29.5.18, "How to Configure Access Control."
7. Click **Finish** to create the connection.

29.5.4 How to Configure OAuth Authentication

As Figure 29–7 shows, you can use the Create MAF Login Connection dialog to configure how third-party applications (clients) gain limited access to protected data or services stored on a remote server. The Relying Party authentication provided by Oracle Mobile and Social server enables an application to authenticate against a third-party OAuth provider. Oracle Web Services Manager (OWSM) Lite Mobile ADF Application Agent injects the cookie into the security header of the web service call.

Figure 29–7 Configuring OAuth



Before you begin:

Configure the server to use the `OM_PROP_OAUTH_OAUTH20_SERVER` property key.

To configure authentication with an OAuth server:

1. In the Mobile Login Server Connection wizard, choose **OAuth2** as the authentication server type, and click **Next**.

   For information about opening the Mobile Login Server Connection wizard, see Section 29.5.1, "How to Create a MAF Login Connection."

2. In the General Options page, define the following:

   - **Connection Name**—Enter a name for the connection.
   - **Authentication Mode**—Select the type of authentication, as described in Table 29–1.
   - **Idle Timeout**—Enter the time for an application feature to remain idle after MAF no longer detects the activation of an application feature. After this period expires, the user is timed-out of all the application features that are secured by the login connection. In this situation, MAF prompts users with the login page when they access the feature again. By default, MAF presents the login page when an application remains idle for 300 seconds (five minutes).

     Note: MAF authenticates against the local credential store after an idle timeout, but does not perform this authentication after a session timeout.

- **Session Timeout**—Enter the time, in seconds, that a user can remain logged in to an application feature. After the session expires, MAF prompts the user with a login page if the idle timeout period has not expired. By default, a user session lasts for 28,800 seconds (eight hours).
- **Include login server in REST calls**—For application features using remote authentication, select this option to enable a REST web service to retrieve the authorized user data that is stored on a login server through a login server-generated user session cookie.

  Note: To enable cookies to be injected in the REST web service call, the application feature must use remote authentication (MAF does not support injecting cookies for application features that store user credentials locally) and the domain entered for **Login URL** must be identical to the domain of the REST web service end point.

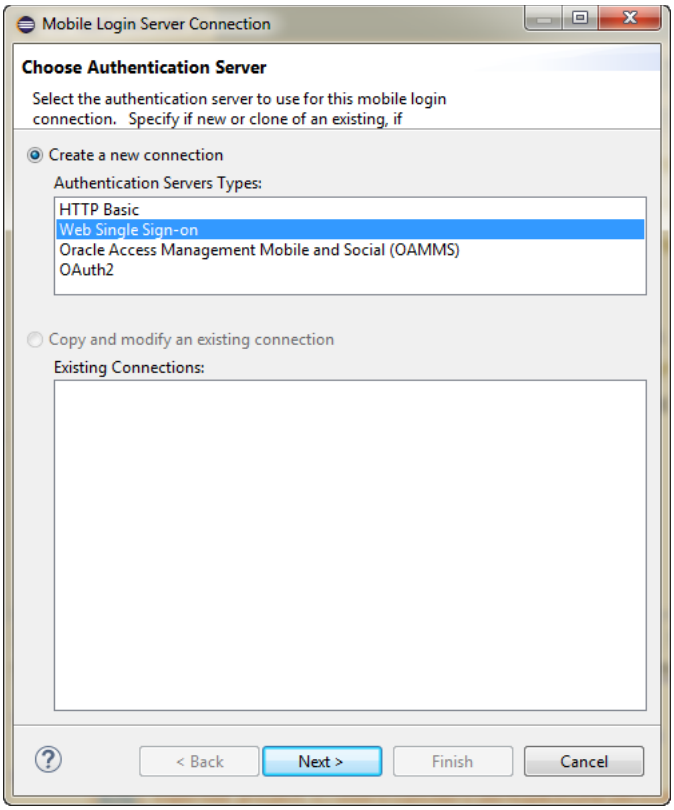
- **Include basic authentication header in HTTP requests**—Select this option to enable MAF to add a basic authentication header into the HTTP requests made from a web view. Basic authentication is the default request method used by MAF. A basic authentication header is injected only when the login connection type is HTTP Basic. See also Section 29.5.16, "What You May Need to Know About Injecting Basic Authentication Headers."

3. Click **Next** to open the Login Options page where you can and configure the parameters as described in Section 29.5.7, "How to Store Login Credentials."
4. Click **Next** to open the Authorization Options page where you can configure the parameters as described in Section 29.5.18, "How to Configure Access Control."

29.5.5 How to Configure Web SSO Authentication

As Figure 29–8 shows, you can use the Mobile Login Server Connection wizard to configure a cross-domain single sign-on.

Figure 29–8 Configuring Federated SSO Authentication



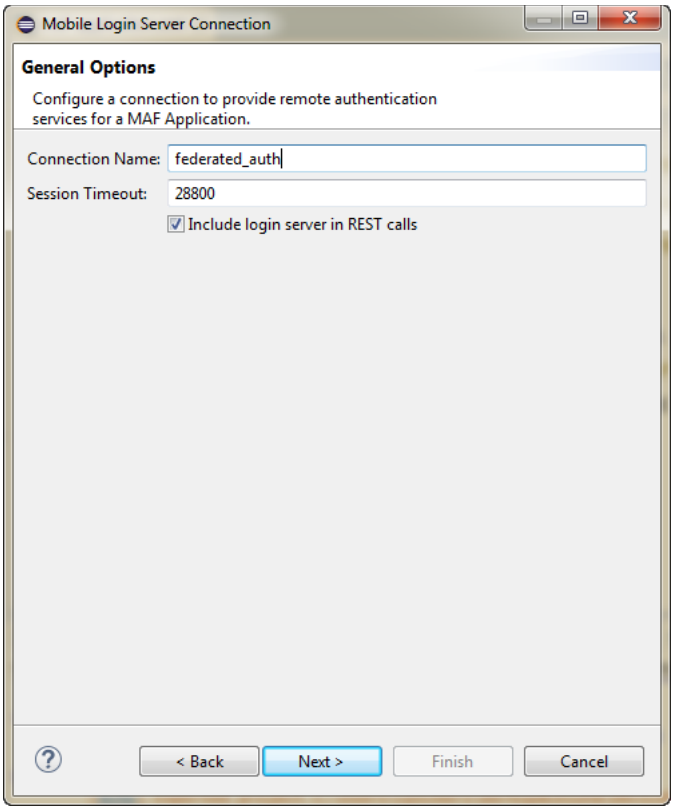
To configure authentication with a Web SSO server:

1. In the Mobile Login Server Connection wizard, choose **Web Single Sign-on** for the authentication server type and click **Next**.

   For information about opening the Mobile Login Server Connection wizard, see Section 29.5.1, "How to Create a MAF Login Connection."

2. In the General Options page, configure the following, as shown in Figure 29–9:

Figure 29–9 Configuring the Connection

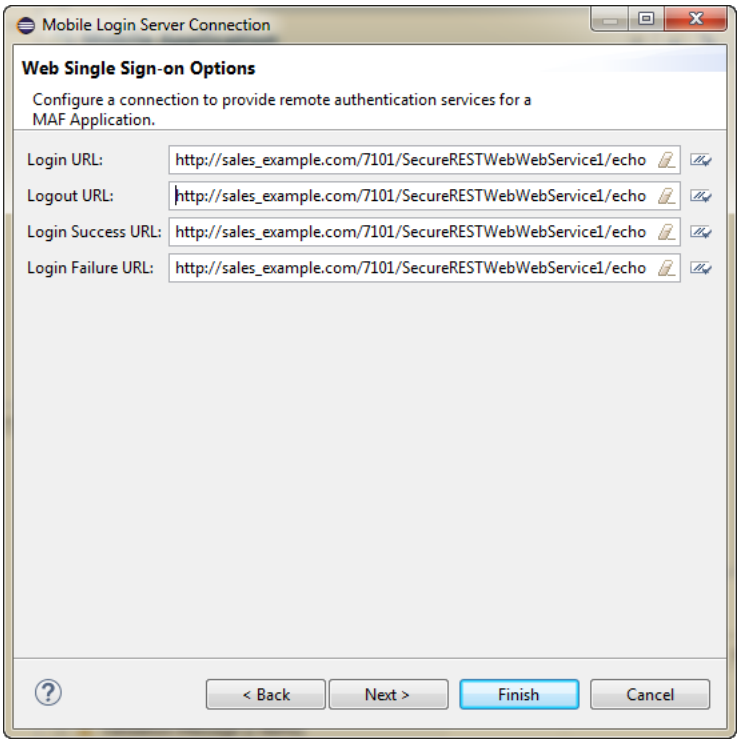


- **Connection Name**—Enter a name for the connection.
- **Session Timeout**—Enter the time, in seconds, that a user can remain logged in to an application feature. After the session expires, MAF prompts the user with a login page if the idle timeout period has not expired. By default, a user session lasts for 28,800 seconds (eight hours).
- **Include login server in REST calls**—For application features using remote authentication, select this option to enable a REST web service to retrieve the authorized user data that is stored on a login server through a login server-generated user session cookie.

Click **Next**.

3. On the Web Single Sign-on Options page, configure the URLs that enable successful and unsuccessful logins. You can test the URLs by clicking .

Figure 29–10 Configuring the Authentication URLs



4. Click **Next**, and on the Authorization Options page configure the parameters, as described in Section 29.5.18, "How to Configure Access Control."

29.5.6 How to Update Connection Attributes of a Named Connection at Runtime

The `AdfmfJavaUtilities` class provides the `overrideConnectionProperty` and `updateSecurityConfigWithURLParameters` methods that application developers can use to define or to redefine the connection attributes of a connection that already exists: either by placeholder (when you select Specify Values at Runtime in the Create MAF Login Connection dialog) or by a fully populated connection definition. Both methods must be invoked in conjunction with the `clearSecurityConfigOverrides` and `updateApplicationInformation` APIs that the `AdfmfJavaUtilities` class also provides.

The `updateSecurityConfigWithURLParameters` method updates parameters required for authentication only. Additional parameters that a connection in `connections.xml` may specify cannot be updated using the `updateSecurityConfigWithURLParameters` method. Use `overrideConnectionProperty` to update all the non-authentication parameters as well as all the parameters that can be updated with `updateSecurityConfigWithURLParameters`.

Note: The typical timing is to call the `AdfmfJavaUtilities.updateSecurityConfigWithURLParameters` API in a `start()` method implementation within an application lifecycle listener. You must not call this method from within the feature lifecycle listener.

To update connection attributes associated with the `configUrlParam` parameter, call the `updatedSecurityConfigWithURLParameters` method in conjunction with the other methods shown in the following example:

```
AdfmfJavaUtilities.clearSecurityConfigOverrides(loginConnectionName);
AdfmfJavaUtilities.updateSecurityConfigWithURLParameters(configUrlParam, key,
message, showConfirmation);
// Final step to apply the changes
AdfmfJavaUtilities.updateApplicationInformation(false);
```

The key parameter is set as a String object from the value defined for the `adfCredentialStoreKey` parameter in the `connections.xml` file. Use this key parameter in all references to the configuration, such as subsequent updates. The method may be invoked with the `showConfirmation` parameter set to true to allow MAF to display a confirmation prompt to the end user once MAF detects a connection configuration change to an existing attribute of the connection.

The string value that you pass to the `configUrlParam` parameter must be UTF-8 encoded and formatted as follows:

```
String parameterString = "http://settings?" +
  "&<Parameter Name1>::=<Parameter Value1>" +
  "&<Parameter Name2>::=<Parameter Value2>" +
  ...
  "&<Parameter NameN>::=<Parameter ValueN>";
```

For example, passing the following values to the `configUrlParam` parameter:

```
http://settings?AuthServerType::=HTTPBasicAuthentication
&ApplicationName::=Approvals
&LoginURL::=http://hostname.com:8008/OA_HTML/RF.jsp?function_id=mLogin
&LogoutURL::=http://hostname.com:8008/OA_HTML/RF.jsp?function_id=mLogout
&SessionTimeOutValue::=28800
&IdleTimeOutValue::=7200
&CryptoScheme::=PlainText
```

Requires you to create a URL as follows:

```
http://settings?AuthServerType::=HTTPBasicAuthentication&ApplicationName::=Approva
ls&LoginURL::=http%3A%2F%2Fhostname.com%3A8008%2FOA_HTML%2FRF.jsp%3Ffunction_
id%3DmLogin&LogoutURL::=http%3A%2F%2Fhostname.com%3A8008%2FOA_
HTML%2FRF.jsp%3Ffunction_
id%3DmLogout&SessionTimeOutValue::=28800&IdleTimeOutValue::=7200&CryptoS
cheme::=PlainText
```

Note the following additional points about the `updateSecurityConfigWithURLParameters` and `overrideConnectionProperty` methods:

- The `updateSecurityConfigWithURLParameters` method persists the new configuration immediately while `overrideConnectionProperty` does not persist the configuration change until the next time the MAF application uses the login connection. For example, an end user navigates to a feature that is protected by the login connection and MAF shows the login view.
- You can use `overrideConnectionProperty` to reconfigure any top-level properties in a connection reference and connection references not limited to login connections in the `connections.xml` file. The `updateSecurityConfigWithURLParameters` method can be only used to update login connections.

- Calls to `overrideConnectionProperty` calls are cumulative while calls to `updateSecurityConfigWithURLParameters` reconfigure previous calls to `updateSecurityConfigWithURLParameters` and result in a new login URL each time `updateSecurityConfigWithURLParameters` is called. The following example demonstrates how a sequence of `overrideConnectionProperty` calls overrides the values of the `login`, `logout`, and `accessControl` properties for a connection named `MyLoginConnection`.

```
AdfmfJavaUtilities.clearSecurityConfigOverrides("MyLoginConnection");
AdfmfJavaUtilities.overrideConnectionProperty("MyLoginConnection", "login",
"url", newLoginUrl);
AdfmfJavaUtilities.overrideConnectionProperty("MyLoginConnection", "logout",
"url", newLogoutUrl);
AdfmfJavaUtilities.overrideConnectionProperty("MyLoginConnection",
"accessControl", "url", newAccessControlUrl);
AdfmfJavaUtilities.updateApplicationInformation(false);
```

- The second parameter value in `overrideConnectionProperty (String node)` is the parameter named used in the `connections.xml` file. For example, to update the following connection:

```
<Reference name="remotePage"
className="oracle.adf.model.connection.url.HttpURLConnection" xmlns="">
    <Factory className="oracle.adf.model.connection.url.URLConnectionFactory"/>
    <RefAddresses>
      <XmlRefAddr addrType="remotePage">
        <Contents>
          <urlconnection name=" remotePage_urlconnectionName "
url="http://www.google.com"/>
        </Contents>
      </XmlRefAddr>
    </RefAddresses>
  </Reference>
```

  You call `overrideConnectionProperty` to change the URL as follows:

```
overrideConnectionProperty("remotePage", "urlconnection", "url",
"http://www.oracle.com");
```

  This is not the same parameter name as in `updateSecurityConfigWithURLParameters`. Follow the URL construction pattern described above when using `updateSecurityConfigWithURLParameters`. Knowledge of the contents of the connections.xml file is not required.

For more information on the `oracle.adfmf.framework.api.AdfmfJavaUtilities` class and the usage of the `configUrlParam` parameter, see *Java API Reference for Oracle Mobile Application Framework*.

For more information about how to override a connection property value using `overrideConnectionProperty`, see Section 29.5.21, "How to Configure Login Credentials Programmatically Prior to Authentication." The `ConfigServiceHandler.java` in the ConfigServiceDemo sample application demonstrates how to invoke the `overrideConnectionProperty` method to override a number of connection properties. For more information about the ConfigServiceDemo sample application, see Appendix E, "MAF Sample Applications."

29.5.7 How to Store Login Credentials

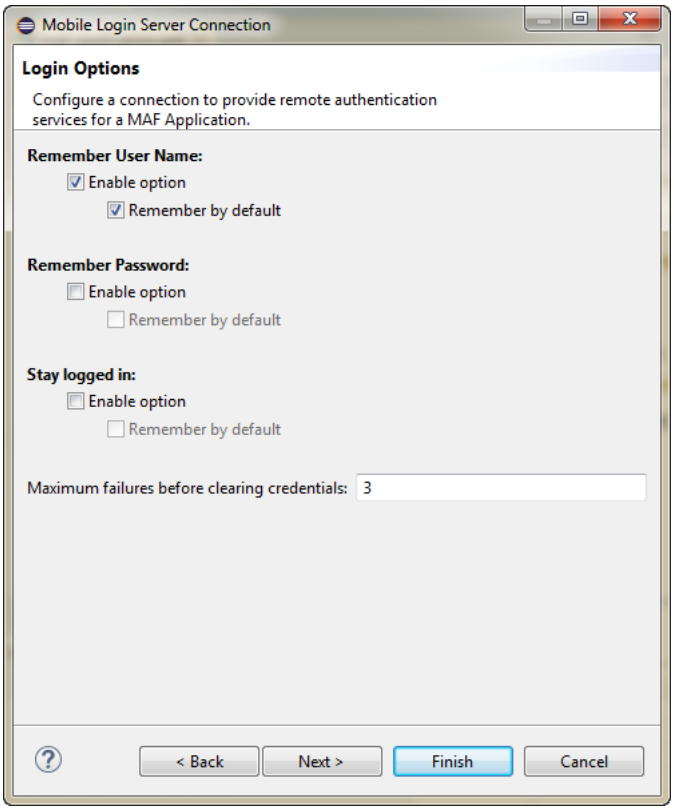
When security is not critical, MAF supports storing user credentials, which can be replayed to the login server or used to authenticate users locally (depending on the mode defined for the login connection. Storing credentials enhances the user experience by enabling users to access the MAF application without having to login. The IDM Mobile SDKs enable MAF to support the following modes:

- remember username—MAF caches the user name and populates the login page's **username** field. After the user enters the password and confirms by tapping the login button, MAF replays these credentials to the authentication server.
- remember credentials—MAF caches the user credentials and populates the login page's user name and password fields. After the user confirms these credentials by tapping the login button, MAF replays them to the authentication server.
- auto login—MAF caches the user credentials and then replays them to the authentication server during subsequent authentications. In this mode, users can start an application without MAF prompting them to enter or confirm their credentials. MAF can, however, inform users that it has started a new application session.

Note: Users can decide whether MAF stores their credentials.

As Figure 29–11 shows, you can use the Login Options page of the Mobile Login Server Connection wizard to select credential storing options. Selecting credential options populates the login page with options to remember the user name and password and should not be selected when a device is shared by multiple end users.

Figure 29–11 Caching User Credentials



29.5.8 What Happens When You Create a Connection for a MAF Application

MAF aggregates all of the connection information in the `connections.xml` file (located in the assembly project under the **adf** and **META-INF** nodes). This file, shown in the following example, can be bundled with the application or can be hosted for the Configuration Service. In the latter case, MAF checks for the updated configuration information each time an application starts.

> **Note:** As a requirement for MAF application authentication, OEPE sets the `adfCredentialStoreKey` attribute to the same name as the login connection reference (for example, `Connection_1`). When editing the `adfCredentialStoreKey` attribute or the login connection name in the `connections.xml` file be sure to set the value to be identical for each. Failure to maintain identical values will result in a MAF runtime exception.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<References xmlns="http://xmlns.oracle.com/adf/jndi">
  <Reference name="Connection_1"
             className="oracle.adf.model.connection.adfmf.LoginConnection"
             adfCredentialStoreKey="Connection_1"
             partial="false"
             manageInOracleEnterpriseManager="true"
             deployable="true"
             xmlns="">
      <Factory
className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
      <RefAddresses>
         <XmlRefAddr addrType="adfmfLogin">
            <Contents>
               <login url="http://10.0.0.0/SecuredWebServicelogin/login"/>
               <logout url="http://10.0.0.0/SecuredWebServicelogout/logout"/>
               <accessControl url="http://10.0.0.0/Identity/Authorize"/>
               <isAcsCalledAutomatically value="false"/>
               <idleTimeout value="300"/>
               <sessionTimeout value="28800"/>
               <isMultiTenantAware value="true"/>
               <multiTenantHeaderName value="Oracle_Multi_Tenant"/>
               <injectCookiesToRESTHttpHeader value="true"/>
               <userObjectFilter>
                  <role name="manager"/>
                  <privilege name="account manager"/>
                  <privilege name="supervisor"/>
                  <privilege name=""/>
               </userObjectFilter>
               <rememberCredentials>
                    <enableRememberUserName value="true"/>
                    <rememberUserNameDefault value="true"/>
                    <enableRememberPassword value="true"/>
                    <rememberPasswordDefault value="true"/>
                    <enableStayLoggedIn value="true"/>
                    <stayLoggedInDefault value="true"/>
               </rememberCredentials>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
</References>
```

29.5.9 What Happens When You Create a Multi-Tenant Aware Connection

After you complete the MObile Login Server Connection wizard with the Multi-Tenant Aware option enabled, MAF populates the `connections.xml` file with the `<isMultiTenantAware>` element set to true. In multi-tenant connections, the user name is the aggregation of tenant name and user name.

The login page uses a JavaScript utility to discern if a connection is multi-tenant aware. If the login page detects such a connection, it provides an additional field that requires the user to enter the tenant name configured in the Create MAF Login Connection, as shown in Figure 29–3. After a successful login (which includes entering the correct tenant ID), MAF stores the tenant ID in the local credential store.

29.5.10 What You May Need to Know About the Login Connection Configuration

When you define the login URL to grant access to secured MAF features, either in the `connections.xml` file or programmatically, the login URL must not point to a file resource, such as `mydocument.txt`. The login URL must point to a web resource that does not result in file transfer when requested. If a file resource is used instead, the MAF application may hang or login will fail, thus preventing the user from accessing the secured MAF feature.

29.5.11 What You May Need to Know About Login Connections and Containerized MAF Applications

In order to use the Oracle Mobile Security Suite (OMSS) AppTunnel feature to access corporate resources, the Mobile Security Access Server (MSAS) instance must be configured to proxy the URI or endpoint used by the MAF application to access the resources. For more information, see the "Managing Mobile Security Access Server Applications" chapter and the "Configuring a Mobile Security Access Server Instance" chapter in *Administering Oracle Mobile Security Access Server*.

Note that HTTP Basic authentication is supported by OMSS but the MAF login page will not be suppressed by the OMSS authentication process and therefore will require the user to authenticate two times.

For more information about authentication in containerized MAF applications, see Section 29.4, "Overview of the Authentication Process for Containerized MAF Applications."

29.5.12 What You May Need to Know About Multiple Identities for Local and Hybrid Login Connections

Like a remote connection, local and hybrid login connection modes allow a user to log in and log out using any number of identities within an application lifecycle. When you define a login connection to use these connectivity modes, you enable users to log back into a secured application feature using the local credential store if they have previously logged into a secured application feature within the current session timeout duration. In this case, users who have logged out explicitly, or have been logged out because the idle timeout has expired, can log back into a secured application feature (or any other application feature secured by the login server that protects that application feature).

Note: Local and hybrid connections are only available for basic authentication. Because OAuth and Federate SSO use remote authentication, application users cannot log back into an application unless they authenticate successfully.

29.5.13 What You May Need to Know About Migrating a MAF Application and Authentication Modes

When you migrate a MAF application, you must verify that the authentication mode defined in `maf-feature.xml` (such as `<adfmf:feature id="feature1" name="feature1" credentials="remote">`) is defined by the `authenticationMode` attribute in the `connections.xml` file.

Because the `authenticationMode` attribute in the `connections.xml` file can only be defined as either `remote` or `local`, do not migrate the value of none (`<adfmf:feature id="feature1" name="feature1" credentials="none">`), as doing so causes the deployment to fail.

29.5.14 What You May Need to Know About Custom Headers

After you complete the Create MAF Login Connection dialog with custom headers defined, MAF populates the `connections.xml` file with the `customAuthHeaders` element and individual `header` subelements.

If the value of the custom headers is to be supplied at runtime, the MAF application can use the `OverrideConnectionHandler` API in the `oracle.adfmf.framework.api.AdfmfJavaUtilities` class to configure header values. The `oracle.adfmf.framework.api.AdfmfAuthConnection` class provides convenience methods to access the connection.xml XML elements and retrieve the most recent value when they have been overridden. See Section 29.5.21, "How to Configure Login Credentials Programmatically Prior to Authentication."

After a successful login (which includes entering the correct header values), MAF stores the header details in the local credential store, and allows secure calls, such as those made to REST services, to include custom headers on the HTTP Request.

29.5.15 What Happens at Runtime: When MAF Calls a REST Web Service

After a user is authenticated locally, MAF silently authenticates the user against the login server when the mobile application calls a REST web service. After the user's credentials are authenticated, MAF executes the application's request to the REST web service. If the REST web service returns a `401` status code (Unauthorized), MAF prompts the user to authenticate again. If the REST web service returns a `302` code (Found or temporarily moved), MAF checks the login server to confirm if the user is authenticated. If so, then the code is handled as a `302` redirect.

If the user has not been authenticated against the login server, then MAF prompts the user to authenticate again. In some cases, a login server may prompt users to authenticate using its own web page when it returns a `302` status code. MAF does not support redirection in these instances and instead prompts the user to login again using a MAF login page.

29.5.16 What You May Need to Know About Injecting Basic Authentication Headers

MAF enables application features to access secure resources by injecting a basic authentication header into the HTTP requests made by the web view in an application

feature. This is useful in situations where a remote web resource is protected by basic authentication and cookies are not sufficient for authentication, or the server does not honor cookies at all. Specify a requesting realm in the **Realm** input field of the Create MAF Login Connection dialog's Authorization tab if known at the time of development. If not known at the time of development, the requesting realm can be modified using `AdfmfJavaUtilities.overrideConnectionProperty` at runtime.

The following example shows the entry that appears in the `connections.xml` file when you specify a value in the Realm input field (`realm` element).

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<References xmlns="http://xmlns.oracle.com/adf/jndi">
   <Reference name="connection1"
              className="oracle.adf.model.connection.adfmf.LoginConnection"
              adfCredentialStoreKey="connection1"
              partial="false" manageInOracleEnterpriseManager="true"
              deployable="true" xmlns="">
      <Factory className=
                    "oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
      <RefAddresses>
         <XmlRefAddr addrType="adfmfLogin">
            <Contents>
               <login url="http://www.us.example.com/userInfo"/>
               <logout url="http://www.us.example.com/userInfo"/>
               <accessControl url="http://10.0.0.0/identity/authorize"/>
               <idleTimeout value="300"/>
               <sessionTimeout value="28800"/>
               <cookieNames/>
               <realm value="Secure Area"/>
               <userObjectFilter/>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
</References>
```

29.5.17 What You May Need to Know about Web Service Security

There are no login pages for web services; user access is instead enabled by MAF injecting credentials into the header of the web service call. Web services gain access to application data using the locally stored credentials persisted by MAF after the user's first successful login to the authentication server.

The name of the local credential store is reflected by the `adfCredentialStoreKey` attribute of the login server connection (such as `adfCredentialStoreKey="Connection_1"` in the example in Section 29.5.8, "What Happens When You Create a Connection for a MAF Application."). To enable a web service to use this credential store, the name defined for the `adfCredentialStoreKey` attribute of a REST web service connection must match the name defined for the login server's `adfCredentialStoreKey` attribute. When editing the `adfCredentialStoreKey` attribute or the login connection name in the `connections.xml` file be sure to set the value to be identical for each. Failure to maintain identical values will result in a MAF runtime exception.

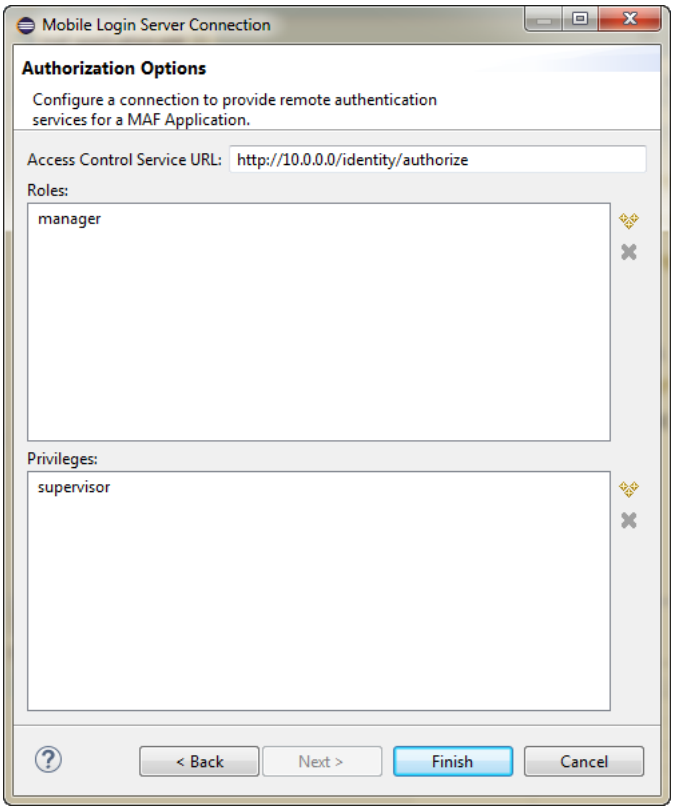
Note: Because there is no overview editor for the `connections.xml` file, you can use the Source editor to update the `<Reference>` element's `adfcredentialStoreKey` attribute with the name configured for `adfCredentialStoreKey` attribute of the login server connection.

For more information, see Section 15.3.4, "What You May Need to Know About Credential Injection."

29.5.18 How to Configure Access Control

Access Control Service (ACS) is a RESTful web service with JSON that may be optionally deployed onto an external server that is separate from your MAF application. Typically, you provide the ACS service for your MAF application to consume when your application features contain secured components and you want to allow users to download their user roles and privileges through a single HTTP POST message. If you intend to provide this service with your application, then you must implement and host the ACS service; MAF does not provide this service. Figure 29–12 shows the Authorization page of the Mobile Login Server Connection wizard that you would use to configure the MAF application to support access control.

Figure 29–12 Configuring Access Control



The access control granted by the application login server is based on the evaluation of the `user.roles` and `user.privileges` constraints configured for an application feature, as described in Section 22.2.4, "About User Constraints and Access Control." For example, to allow only a user with *manager_role* role to access an application feature, you must define the `<adfmf:constraints>` element in the `maf-feature.xml` file with the following:

```
<adfmf:constraint property="user.roles"
                  operator="contains"
                  value="manager_role"/>
</adfmf:constraints>
```

At the start of application, the RESTful web service known as the Access Control Service (ACS) is invoked for the application login server connection and the roles and privileges assigned to the user are then fetched. MAF then challenges the user to login against the application login server connection.

MAF evaluates the constraints configured for each application against the retrieved user roles and privileges and makes only the application features available to the user that satisfy all of the associated constraints.

To configure access control:

1. In the Mobile Login Server Connection wizard, navigate to the Authorization page.

   For information about opening the Mobile Login Server Connection wizard, see Section 29.5.1, "How to Create a MAF Login Connection."

2. On the Authorization page, complete the authorization requirements, as shown in Figure 29–12.

   - **Access Control Service URL**—Enter the URL that is the endpoint for the Access Control Service (ACS).
   - **Roles**—Enter the user roles checked by the application feature. Because there may be thousands of user roles and privileges defined in a security system, use the manifest provided by the application feature developer that lists the roles specific to the application feature to create this list.
   - **Privileges**—Enter the privileges checked by the application feature.

29.5.19 What You May Need to Know About the Access Control Service

The Access Control Service (ACS) is a RESTful web service with JSON that enables users to download their user roles and privileges through a single HTTP `POST` message. This is a request message, one which returns the roles or privileges (or both) for a given user. It can also return specific roles and privileges by providing lists of required roles and privileges. The request message is comprised of the following:

- Request header fields: `If-Match`, `Accept-Language`, `User-Agent`, `Authorization`, `Content-Type`, `Content Length`.
- A request message body (a request for user information).
- The requested JSON object that contains:
  - `userId`—The user ID.
  - `filterMask`—A combination of `"role"` and `"privilege"` elements are used to determine if either the filters for user roles, or for privileges, should be used.
  - `roleFilter`—A list of roles used to filter the user information.
  - `privilegeFilter`—A list of privileges used to filter the user information.

Note: If all of the roles should be returned, then do not include the `"role"` element in the `filterMask` array.

If all of the privileges should be returned, then do not include the `"privilege"` element in the `filterMask` array.

The following example illustrates an HTTP `POST` message and identifies a JSON object as the payload, one that requests all of the filters and roles assigned to a user, John Smith.

```
Protocol: POST
Authoization: Basic xxxxxxxxxxxx
Content-Type: application/json

{
        "userId": "johnsmith",
        "filterMask": ["role", "privilege"],
        "roleFilter": [ "role1", "role2" ],
        "privilegeFilter": ["priv1", "priv2", "priv3"]
}
```

The response is comprised of the following:

- A response header with the following fields: `Last-Modified`, `Content-Type`, and `Content-Length`.
- A response message body that includes the user information details.
- The returned JSON object, which includes the following:
  - `userId`—the ID of the user.
  - `roles`—A list of user roles, which can be filtered by defining the `roleFilter` array in the request. Otherwise, the response returns an entire list of roles assigned to the user.
  - `privileges`—A list of the user's privileges, which can be filtered by defining the `privilegeFilter` array in the request. Otherwise, the response returns an entire list of privileges assigned to the user.

The example below illustrates the returned JSON object that contains the user name and the roles and privileges assigned to the user, John Smith.

```
Content-Type: application/json

{
        "userId": "johnsmith",
        "roles": [ "role1" ],
        "privileges": ["priv1", "priv3"]
}
```

Note: There are no login pages for web services; user access is instead enabled by MAF, which automatically adds credentials to the header of the web service. For more information, see Section 15.3.4, "What You May Need to Know About Credential Injection."

Note: You must implement and host the ACS service; MAF does not provide this service.

29.5.20 How to Alter the Application Loading Sequence

MAF invokes the Access Control Service (ACS) after a user successfully authenticates against a login connection that defines the ACS endpoint, such as `http://10.0.0.0/Identity/Authorize` in Figure 29–12. By changing this behavior to prevent the ACS from being called immediately after a successful login, you can insert a custom process between the login and the invocation of the ACS. This additional logic may be a security context called by MAF after a successful login that is based on the specifics of an application, or related to the user's responsibilities, organization, or security group.

You can change the sequence by updating the `connections.xml` file with `<isAcsCalledAutomatically value = "false"/>` and through the following method of the `AdfmfJavaUtilities` class, which enables MAF application features to call the ACS whenever it is required:

```
invokeACS(String key, String OptionalExtraPayLoad, boolean appLogin)
```

The `invokeACS` method enables you to inject extra payload into an ACS request. The `key` parameter is returned as a `String` object from the value defined for the `adfCredentialStoreKey` parameter in the `connections.xml` file, as illustrated in the example in Section 29.5.16, "What You May Need to Know About Injecting Basic Authentication Headers.". The `appLogin` parameter may be set to `true` to cause ACS to reevaluate the feature access. The `OptionalExtraPayLoad` parameter is reserved for future use and is not used.

Invoking ACS through either the `invokeACS` method or the `isAcsCalledAutomatically` parameter retrieves the role-based constraints for an application.

Note: MAF automatically invokes the ACS after a successful login if `<isAcsCalledAutomatically value = "false"/>` is not included in the `connections.xml` file.

When a secured application feature calls the `invokeACS` method, MAF fetches the user constraints for all of the application features associated with the application login connection, including those configured for the secured application feature. When an unsecured application feature calls this method, MAF only retrieves the constraints associated with the login connection.

Note: In addition to the `invokeACS` method, the `AdfmfJavaUtilities` class includes the following lifecycle methods:

- `applicationLogout`—Logs out the application login connection.
- `featureLogout(<feature_ID>)`—Logs out the login connection associated with the application feature.

For more information, see the *Java API Reference for Oracle Mobile Application Framework*.

29.5.21 How to Configure Login Credentials Programmatically Prior to Authentication

Before the MAF application invokes the login connection to authenticate the user, it is possible to set connection values programmatically. This technique is often required, for example, when custom header names are defined in the Mobile Login Server Connection wizard but the values are to be supplied at runtime. To programmatically configure the connection details, the MAF application can invoke the `OverrideConnectionHandler` API in the `oracle.adfmf.framework.api.AdfmfJavaUtilities` class. The API overrides the current connection property value with a new value and allows the application to initiate login with the overridden values.

To override the connection values programmatically, the general process entails:

1. Obtaining the names of the XML elements that define the properties to override from the `connections.xml` file.
2. Obtaining the override value prior to authentication. For example, the MAF application may define an AMX page to solicit the values from the end user for this purpose.
3. Invoking a managed bean that implements the override methods (one for each connection property override) and defines connection property getter and setter methods. For example, in the case of an AMX page, a command button that the end user clicks may submit their entered values on the managed bean.

To specify connection property overrides, examine the `connections.xml` file to obtain the following XML element definitions:

- The connection reference name. For example, `ConnWithCustomHeader`.
- The XML element name of the property that defines the attribute to override. For example `multiTenantScheme` for the scheme used to propagated a tenant domain name.
- The XML subelement name of the property's attribute to override. In the case of unique connection properties, this is always the `value` element.

> **Note:** In the case of custom headers, do not use the XML elements `header` and `value` since all custom header definitions use the same element names to specify values. Instead, for custom headers use `Contents` and `customAuthHeaders` for the property and attribute to pass to the override method, respectively.

Obtain XML Element Names

For example, to override the value of custom headers, in the following `connections.xml` file you would pass the connection name `ConnWithCustomHeader`, the `Contents` element, and the `customAuthHeaders` subelement, which defines the header name / value pairs.

```
package mobile;

<References xmlns="http://xmlns.oracle.com/adf/jndi">
  <Reference name="ConnWithCustomHeader"
             className="oracle.adf.model.connection.adfmf.LoginConnection"
             adfCredentialStoreKey="ConnWithCustomHeader" partial="false"
             manageInOracleEnterpriseManager="true"
             deployable="true" xmlns="">
    <Factory
className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
```

```
      <RefAddresses>
        <XmlRefAddr addrType="adfmfLogin">
          <Contents>
            ...
            <isMultiTenantAware value="true"/>
            <multiTenantScheme value="custom_header"/>
            <multiTenantHeaderName value="X-ID-TENANT-NAME"/>
            <customAuthHeaders>
              <header name="State" value="Georgia"/>
              <header name="City" value="Atlanta"/>
            </customAuthHeaders>
            ...
          </Contents>
        </XmlRefAddr>
      </RefAddresses>
    </Reference>
```

Obtain Override Values

To perform the connection value override at runtime, the MAF application may solicit the values with a default unsecured feature that invokes an AMX page with prompts for the values and a command button to submit the values. The following sample shows the command button defines an action listener that triggers the override method in a managed bean:

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
  <amx:panelPage id="pp1">
    <amx:facet name="header">
      <amx:outputText value="Home" id="ot1"/>
    </amx:facet>
    <amx:facet name="primary">
      <amx:commandButton id="cb1"/>
    </amx:facet>
    <amx:facet name="secondary">
      <amx:commandButton id="cb2"/>
    </amx:facet>
    <amx:inputText label="Name1" id="it1" value="#{applicationScope.OverrideBean.headerName1}"/>
    <amx:inputText label="Value1" id="it2" value="#{applicationScope.OverrideBean.headerValue1}"/>
    <amx:inputText label="Name2" id="it3" value="#{applicationScope.OverrideBean.headerName2}"/>
    <amx:inputText label="Value1" id="it4" value="#{applicationScope.OverrideBean.headerValue2}"/>
    <amx:inputText label="TenantHeader" id="it5"
                                        value="#{applicationScope.TestBean.tenantHeaderName}"/>
    <amx:inputText label="Scheme" id="it6" value="#{applicationScope.OverrideBean.scheme}"/>
    <amx:commandButton text="Override headers" id="cb3"
                       actionListener="#{OverrideBean.overrideAndGotoOverrideFeature}"/>
  </amx:panelPage>
</amx:view>
```

Override the Connection Properties

To override the connection property values programmatically, the managed bean implements the override method for each connection property override. Note that in the following sample Ova `headers` HashMap is created to pass in the custom header values. The map is only necessary for custom headers that you want to override since the values of other properties (like `MultiTenantHeaderName`) are uniquely defined by the XML elements of the `connections.xml` definition.

```
package mobile;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;

import java.util.HashMap;

public class OverrideBean {

    private String headerName1 = "", headerName2 = "", headerValue1 = "", headerValue2 = "";
    private String tenantHeaderName = "";
    private String scheme = "";

    // Bean setter and getter methods omitted for brevity
    ...

    // Command button action listener invokes override method implementation
    public void overrideAndGotoOverrideFeature(ActionEvent e) {
        overrideAndGotoOverrideFeature();
    }

    // Override method implementation configures custom headers and other connection properties
    public void overrideAndGotoOverrideFeature()
    {
      AdfmfJavaUtilities.clearSecurityConfigOverrides("ConnWithCustomHeader");
      HashMap<String, String> headers = new HashMap<String, String>();
      headers.put(headerName1, headerValue1);
      headers.put(headerName2, headerValue2);
      AdfmfJavaUtilities.overrideConnectionProperty("ConnWithCustomHeader", "Contents",
                                                    "customAuthHeaders", headers);
      AdfmfJavaUtilities.overrideConnectionProperty("ConnWithCustomHeader",
                                                    "multiTenantHeaderName", "value",
                                                    tenantHeaderName);
      AdfmfJavaUtilities.overrideConnectionProperty("ConnWithCustomHeader", "multiTenantScheme",
                                                    "value", scheme);
      AdfmfJavaUtilities.updateApplicationInformation(false);

    }

    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }
}
```

29.6 Configuring Security for Mobile Applications

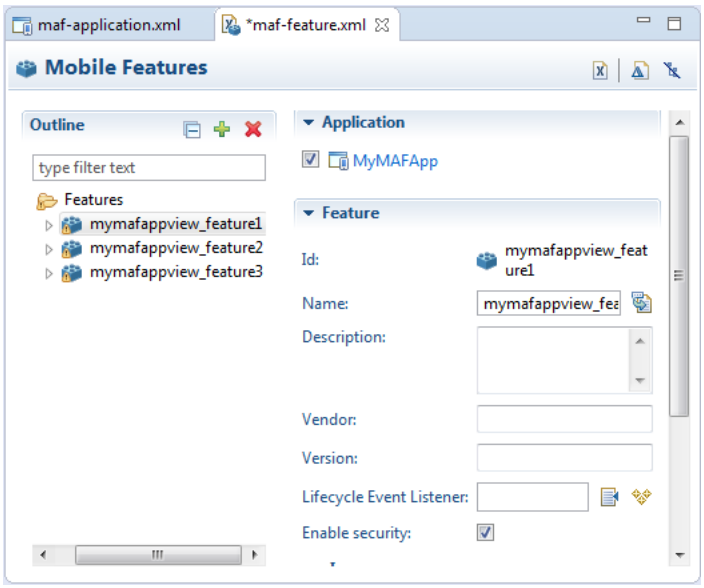
You configure security using the MAF Feature Editor and MAF Application Editor, as well as the Mobile Login Server Connection wizard. The editors enable you to designate the type of login page (default or custom) that MAF presents to users when they select application features that require authentication or to include user role- or user privilege-based constraints. They also enable you to select which embedded application features require security.

29.6.1 How to Enable Application Features to Require Authentication

You can define each application feature to participate in security. You perform the remainder of the security configuration using the Security section of the MAF Application Editor.

You can designate which application features participate in security and the type of authentication they require in the Feature section of the MAF Feature Editor, shown in Figure 29–13.

Figure 29–13 Designating User Credentials Options for an Application Feature



Before you begin:

When you enable security for a feature, the application requires access the network to authenticate the user. For more information about granting the application access to network sockets, see Section 9.2, "Enabling a Core Plugin in Your MAF Application."

To designate user access for an application feature:

1. In the Project Explorer, in the user interface project, expand **MAF**, and then double-click **MAF Feature Editor**.
2. In the MAF Feature Editor, select an application feature listed under **Features** or right-click **Features** and choose **New** to add an application feature.
3. Select **Enable security** for any application feature that requires login.

> **Tip:** If you do not apply this option to the default application, you enable users to login anonymously (that is, without presenting login credentials). Users can then access unsecured data and features and, when required, login (authenticated users can access both secured and unsecured data). Providing unsecured application features within a MAF application enables users to logout of secured application features, but remain within the application itself and continue to access both unsecured application features and data.

29.6.2 How to Designate the Login Page

After you designate security for the application features, you use the Security page of the MAF Application Editor, shown in Figure 29–14, to configure the login page as well as create a connection to the login server and assign it to each of the application features that participate in security. All of the application features listed in this page have been designated in the MAF Feature Editor as requiring security.

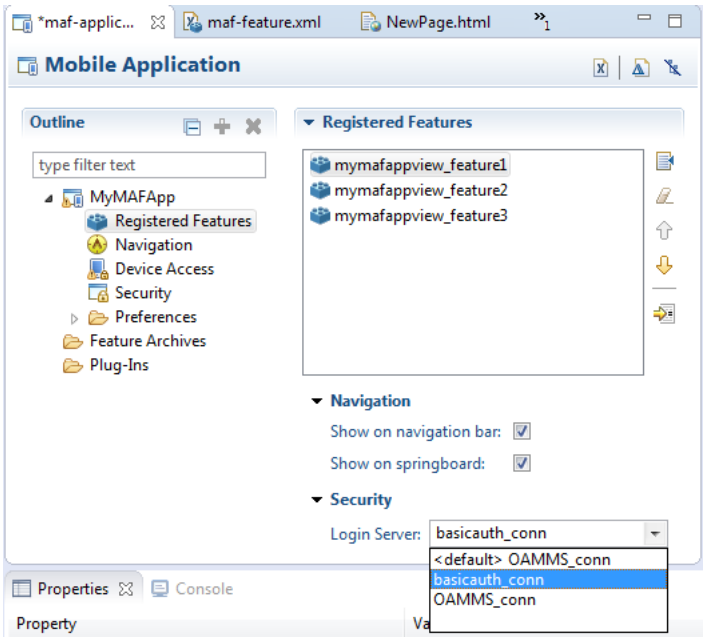
Typically, a group of application features are secured with the same login server connection, enabling users to open any of these applications without MAF prompting them to login again. In some cases, however, the credentials required for the application features can vary, with one set of application features secured by one login server and another set secured by a second login server. To accommodate such situations, you can define any number of connections to a login server for a mobile application. In terms of the `maf-application.xml` file, the authentication server connections associated with the feature references are designated using the `loginConnRefId` attribute as follows:

```
<adfmf:featureReference id="feature1" loginConnRefId="Connection_1"/>
<adfmf:featureReference id="feature2" loginConnRefId="Connection2"/>
```

MAF applications can be authenticated against any standard login server that supports basic authentication over HTTP or HTTPS. MAF also supports authentication against Oracle Identity Management. You can also opt for a custom login page for a specific application feature. For more information, see Section 29.6.4, "What You May Need to Know About Login Pages."

Note: By default, all secured application features share the same connection, which, as shown in Figure 29–14, is denoted as `<application login server>`. Login servers (connections) are defined on the Security tab. The Registered Features tab allows you to assign them to specific features, or use `<default>`. You can select other connections that are defined for the MAF application using the Mobile Login Server Connection wizard.

Figure 29–14 Registered Features in the MAF Application Editor



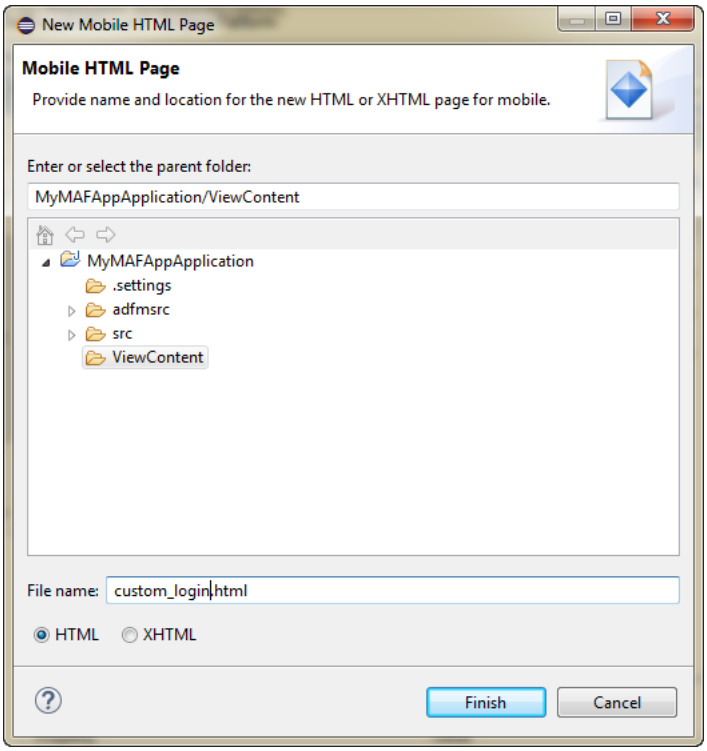
Before you begin:

If the MAF application uses a custom login page, add the file to the `ViewContent` directory of the View Controller project to make it available from the MAF Application Editor, as shown in Figure 29–14. See also Section 29.6.3, "How to Create a Custom Login HTML Page" and Section 5.4, "What You May Need to Know About Selecting External Resources."

Add constraints for user privileges and roles, as described in Section 22.2.4, "About User Constraints and Access Control."

Provision an Access Control Service (ACS) server. For more information, Section 29.5.19, "What You May Need to Know About the Access Control Service."

Figure 29–15 Adding a Custom Login Page



To designate the login page:

1. In the Project Explorer, expand the assembly project, then expand **MAF**, and double-click **MAF Application Editor**.
2. In the outline, click **Security**.
3. On the Security page, choose **Login Page** for a view that accepts a user name and password.
4. Select the content (or user interface) for the selected login page:
   - **Default**—The default login page used for all of the selected embedded application features. For more information, see Section 29.6.4.1, "The Default Login Page." The default login page is provided by MAF.
   - **Custom**—Click **Browse** to retrieve the path location of the file within the application controller project. Alternatively, click **New** to create a custom HTML page within the application controller project for the login page. For more information, see Section 29.6.4.2, "The Custom Login Page."

     Tip: Rather than retrieve the location of the login page using the **Browse** function, you can drag the login page from the Project Explorer into the field.

29.6.3 How to Create a Custom Login HTML Page

You can create the custom login page by modifying the default login page, `adf.login.html`, artifacts generated by the MAF deployment in the `www` directory.

Before you begin:

To access the login page within the `www` directory, deploy a MAF application and then traverse to the deploy directory. For iOS deployments, the pages are located at the following:

```
application workspace directory/deploy/deployment profile name/temporary_xcode_
project/www/adf.login.html
```

For Android deployments, the pages are located within the Android application package (`.apk`) file at the following:

```
application workspace directory/application name/deploy/deployment profile
name/deployment profile name.apk/assets/www/adf.login.html
```

To create a custom login page:

1. Copy the default login page to a location in the `ViewContent` directory of the View Controller project.
2. Rename the login page.
3. Update the page.
4. In the Security section of the Registered Features tab in the MAF Application Editor select the custom login page.

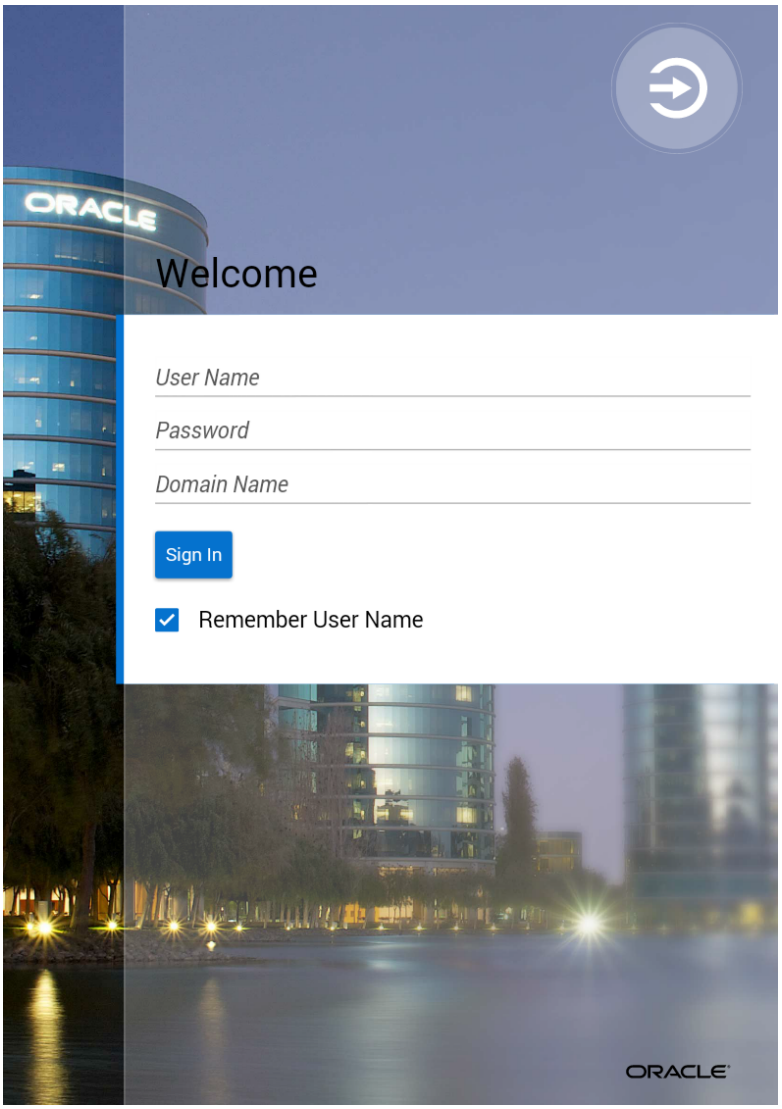
29.6.4 What You May Need to Know About Login Pages

The entry point for the authentication process to an application feature is the `activate` lifecycle event, described at Section 10.1, "Introduction to Lifecycle Listeners in MAF Applications." Every time an application feature is activated (that is, the `activate` event handler for the application feature is called), the application feature login process is executed. This process navigates to the login page (which is either the default or a custom login page) where it determines if user authentication is needed. Before the process navigates to the login page, however, the originally intended application feature must be registered with MAF. When authentication succeeds, the login page retrieves the originally intended destination from MAF and navigates to it.

29.6.4.1 The Default Login Page

The default login page provided by MAF is comprised of a login button, input text fields for the user name, password, and, optionally, multi-tenant name, and an error message section. This is a cross-platform page, one written in HTML. Figure 29–1 shows a default login page with the multi-tenant aware option enabled at design time to solicit the domain name of the tenant, in addition to the user name and password.

Figure 29–16 Default Login Page for Multi-Tenant Aware Connection



29.6.4.2 The Custom Login Page

When you add a custom login page for a selected application feature using MAF Feature Editor, OEPE adds the `<adfmf:login>` element and populates its child `<adfmf:LocalHTML>` element, as shown in the next example. As with all `<adfmf:LocalHTML>` elements, its `url` attribute references a location within the `public_html` directory. The user authentication mechanism and navigation control are identical to the default login page.

```
<adfmf:login defaultConnRefId="Connection_1">
    <adfmf:localHTML url="newlogin.html"/>
</adfmf:login>
```

Custom login pages are written in HTML. The fields for the default login page must include specifically defined `<input>` and `<label>` elements.

> **Tip:** Use the default login pages that are generated when you deploy a MAF application as a guide for creating custom login pages. To access the login pages within the `www` directory, deploy a MAF application and then traverse to the deploy directory, as described in Section 29.6.3, "How to Create a Custom Login HTML Page."

The example below illustrates the required `<input>` and `<label>` elements for a default login page.

```
<input type="text"
       autocorrect="off"
       autocapitalize="none"
       name="oracle_access_user_id"
       id="oracle_access_user_id" value="">
   </input>

<input type="text"
       autocorrect="off"'
       autocapitalize="none"
       name="oracle_access_iddomain_id"
       id="oracle_access_iddomain_id" value="">
   </input>

<input type="password"
       name="oracle_access_pwd_id"
       id="oracle_access_pwd_id" value="">
   </input>

<input type="checkbox"
       class="message-text"
       name="oracle_access_auto_login_id"
       id="oracle_access_auto_login_id">
   </input>Keep me logged in

<input type="checkbox"
       class="message-text"
       name="oracle_access_remember_username_id"
       id="oracle_access_remember_username_id">
   </input>Remember my username

<input type="checkbox"
       class="message-text"
       name="oracle_access_remember_credentials_id"
       id="oracle_access_remember_credentials_id">
   </input>Remember my password

<label id="oracle_access_error_id"
       class="error-text">
   </label>

<input class="commandButton"
       type="button"
       onclick="oracle_access_sendParams(this.id)"
       value="Login" id="oracle_access_submit_id"/>
```

29.6.5 What You May Need to Know About Login Page Elements

Every HTML login page should include the user interface elements listed in Table 29–2.

Table 29–2 Login Page Fields and Their Associated IDs

Page Element	ID
username field	`oracle_access_user_id`
password field	`oracle_access_pwd_id`
login button	`oracle_access_submit_id`
cancel button	`oracle_access_cancel_id`
identity domain/tenant name field	`oracle_access_iddomain_id`
error field	`oracle_access_error_id`
auto login check box	`oracle_access_auto_login_id`
remember credentials check box	`oracle_access_remember_credentials_id`
remember username check box	`oracle_access_remember_username_id`

Table 29–3 lists the recommended JavaScript code used by the `OnClick` event.

Table 29–3 JavaScript Used by the OnClick Event

Button	JavaScript
login button	`oracle_access_sendParams(this.id)`
cancel button	`oracle_access_sendParams(this.id)`

29.6.6 What Happens in OEPE When You Configure Security for Application Features

After an application feature has been designated to participate in security, OEPE updates the Registered Features with a corresponding feature reference, as shown in Figure 29–14. If each of the referenced application features authenticate against the same login server connection defined in the `connections.xml` file, OEPE updates the `maf-application.xml` file with a single `<adfmf:login>` element defined with a `defaultConnRefId` attribute (such as `<adfmf:login defaultConnRefId="Connection_1">`).

For application features configured to use different login server connections defined in the `connections.xml` file OEPE updates each referenced application feature with a `loginConnReference` attribute (`<adfmf:featureReference id="feature2" loginConnRefId="Connection2"/>`). For more information, see Section 29.6.1, "How to Enable Application Features to Require Authentication." See also the *Tag Reference for Oracle Mobile Application Framework*.

29.7 Allowing Access to Device Capabilities

Access to device capabilities is defined by the Cordova plugins that are included in the MAF application. A set of core plugins are provided by MAF. Enabling one of these plugins will enable any device access permissions that it requires. Any additional Cordova plugins that you include in your MAF application will also enable the device access permissions required.

Because the vast majority of MAF applications require network access, permission to access the network is enabled by default (the only device capability that is enabled by default):

- **Network Information**—Allows the application to open network sockets. You must leave the network access capability enabled when security is enabled for at least one device feature.

Because you can enable or restrict device capabilities, the various platform-specific configuration files and manifest files that are updated by the deployment framework list only the device capabilities in use (or rather, the plugins that the MAF application is registered to use). These files enable MAF to share information about the use of these capabilities with other applications. For example, a MAF application can report to the AppStore or to Google Play that it does not use location-based capabilities (even though MAF applications have this capability).

For more information about Cordova plugins in MAF applications, see Chapter 9, "Using Plugins in MAF Applications."

You can prevent selected application features within a MAF application from accessing the native container, and by extension, the device capabilities that the MAF application can access. For example, your MAF application includes an application feature that references remote content from a web application that you do not trust (Remote URL content application feature). In this scenario, you prevent this specific application feature from accessing the native container, as shown in the following example:

```
<adfmf:featureReference refId="remoteAppfeature1" id="fr1"
allowNativeAccess="false"/>
```

The default value of the `allowNativeAccess` property is `true`.

29.8 Enabling Users to Log Out from Application Features

MAF does not terminate application features when a user logs out of one that contains secured content or is restricted through constraints; a user can remain within the application and access its unsecured content and features as an anonymous user. Because MAF enables constraints to be re-initialized, it allows a user to login to an application repeatedly using the same identity. It also enables multiple identities to share the access to the application by allowing the user to login using different identities.

The `logoutFeature` and `logout` methods of the `AdfmfJavaUtilities` class, described in the *Java API Reference for Oracle Mobile Application Framework*, enable users to explicitly login and logout from the authentication server after launching an application. In addition, they enable a user to login to the authentication server after the user invokes a secured application feature. Although a user can log out from individual application features, a user will be simultaneously logged out of application features secured by the same connection.

These methods enable users to perform the following the following:

- Log out of an application feature but continuing to access its unsecured content (that is, MAF does not terminate the application).
- Authenticate with the login server while in an application to enable its secured content and UI components.
- Log out of a MAF application or application feature and then logging in again using a different identity.

- Log out of a MAF application or application feature and then logging in again using the same identity but with updated roles and privileges.

To enable logging out of the current authentication server, call the `logout` method of the `AdfmfJavaUtilities` class as follows. For example:

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
...
   AdfmfJavaUtilities.logout();
```

To enable logging from the authentication server associated with the `key` parameter, call the `logoutFeature` method as follows:

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
...
   AdfmfJavaUtilities.logoutFeature(adfCrendentialStorykey);
```

The `adfCredentialStorykey` parameter is returned as a `String` object from the value defined for the `adfCredentialStoreKey` parameter in the `connections.xml` file. For more information on the `AdfmfJavaUtilties` class and the usage of the `key` parameter, see the *Java API Reference for Oracle Mobile Application Framework*.

29.9 Using MAF Authentication APIs

MAF provides a number of authentication classes that include APIs to assist you with authentication-related tasks in your MAF application.

These APIs reside in the authentication classes of the oracle.maf.api.authentication package. Examples include the `AuthenticationHandler`, `AuthenticationPlatform`, and `AuthenticationUtility` classes.

Tasks that you can perform by implementing these APIs include processing for a logged-in user before the user accesses a secured application. You can, for example, retrieve authorization header information and use the information to determine what content an end user navigates to upon a successful login. `AuthenticationPlatform` also provides an `addLogoutCallback` API that you can implement to perform some processing once an end user logs out from a secured application feature. Use of this callback API may be useful in scenarios where your end users share a device. For more information about these authentication classes, see the *Java API Reference for Oracle Mobile Application Framework*.

The following example shows snippets of code from an implementation class where an instance of `AuthenticationPlatform` gets the authorization header information using its `getToken()` API.

Example 29–1 Getting the Authorization Header from the Authentication Platform

```
import oracle.maf.api.authentication.AuthenticationPlatform;
import oracle.maf.api.authentication.AuthenticationUtility;
…
AuthenticationPlatform ap =
AuthenticationUtility.getInstance().lookupByCredentialStoreKey("credentialStoreKey
");
String authorization = ap.getToken("Authorization");
…
```

29.10 Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL

MAF provides a `cacerts` certificate file, the Java mechanism for HTTPS handshakes between the client application and the server. OEPE creates this file within the application resources `Security` folder (located at [assembly `project\lib\Security\cacerts`). The MAF `cacerts` file identifies a set of certificates from well-known and trusted sources to the JVM and enables deployment.

You need to add private certificates to the MAF `cacerts` file when your application has to access server resources where the server uses a self-signed certificate. You also need to add a private certificate if your application requires custom certificates, such as cases where RSA cryptography is not used. Add a private certificate before you deploy the application.

Before you begin:

It may be helpful to have an understanding of the contents of the cacerts file. For more information, see the "Migrating to New cacerts Files for SSL in MAF" section in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

You may also find it helpful to understand how OEPE creates the `cacerts` file. For more information, see Section C.2, "About the Assembly-Level Resources.".

Refer to Java SE Technical Documentation (`https://download.oracle.com/javase/index.html`) for information on the `cacerts` file and how to use the keytool utility.

To add private certificates:

1. Create a private certificate. For example, create a certificate file called `new_cert`.
2. Add the private certificate to the application as follows:
   a. Create a copy of the seeded `cacerts` file (`cp cacerts cacerts.org`).
   b. Use the Java SE keytool utility to add certificates to a `cacerts` file. This example illustrates adding certificates to a `cacerts` file called `new_cert`.

```
keytool -importcert
        -keystore cacerts
        -file new_cert
        -storepass changeit
        -noprompt
```

The example illustrates how to add a single certificate. Repeat this procedure for each certificate. Table 29–4 lists the keytool options

Table 29–4 Options For Adding Certificates

Option	Description
`-importcert`	Imports a certificate.
`-keystore` *`cacerts file`*	Identifies the file location of the imported certificate.
`-file` *`certificate file`*	Identifies the file containing the new certificate.
`-storepass changeit`	Provides a password for the `cacerts` file. By default, the password is *changeit*.
`-noprompt`	Instructs the keytool not to ask the user (through `stdin`) whether to trust the certificate or not.

c. Visually inspect the contents of the new `cacerts` file to ensure that all of the fields are correct. Use the following command:

```
keytool -list -v -keystore cacerts | more
```

d. Verify that the certificate is for the given hostname.

> **Note:** The certificate's common name (CN) must match the hostname exactly.

e. Ensure that the customized certificate file is located within the `Security` directory (`users\workspaces\`*`assembly project`*`\lib\Security`) so that it can be read by JVM 1.4.

3. Deploy the application.

> **Note:** During deployment, if a certificate file exists within the `Security` directory, MAF copies it into the Android or Xcode template project, replacing any default copies of the `cacerts` file.

4. Validate that you can access the protected resources over SSL.

29.11 Configuring a MAF Application to Enable Two-Way SSL for Authentication

MAF application end users can install digital certificates (client certificate) into their application's keystore to enable authentication using a process known variously as two-way SSL, mutual authentication or two-way authentication.

Once the end user installs a client certificate into the application's keystore, the MAF application can present it to a server so that a two-way SSL communication session is used to perform authentication between the client and the server.

To implement this functionality, you need to register the client certificate file extension in your application. Doing this enables the MAF application to open and install the certificate when the end user opens the client certificate file after downloading it to their device. If the MAF application is the only application on the device to register the client certificate file extension, it will be the application that the device's operating system proposes to the end user to open the certificate. If other applications register the file extension, the device's operating system presents a list of applications from which your end user must choose the MAF application.

Administrators can distribute client certificates to MAF application end users by email attachment or by hosting the certificate in a location from which the end user can download it using a URL.

The iOS platform does not permit third party applications to open files with the default extension for client certificates (`.p12`). To work around this restriction, you (the application developer) must register an alternative certificate extension (for example, `.cert`). Administrators who distribute the certificates can rename the client certificate files to use the alternative extension so that the MAF application end user can open the client certificate directly from the email attachment or URL used to distribute the client certificate.

How to Configure a MAF Application to Enable Two-Way SSL

You register the client certificate file extension in your application by entering the file extension in the Client SSL Certificate Extension field on the Application page of the MAF Application Editor.

To configure a MAF application to enable two-way SSL:

1. In the Project Explorer, expand the assembly project, then expand **MAF**, and double-click **MAF Application Editor**.
2. In the outline, click the assembly project name.
3. In the Application page, enter the file extension in the **Client SSL Certificate Extension** field.

What Happens When You Configure a MAF Application to Enable Two-Way SSL

OEPE writes the value that you enter in the Client SSL Certificate Extension field to the `maf-application.xml` file, as shown in Example 29–2.

At runtime, the end user downloads the client certificate to the device from the location where the administrator put the certificate or opens it automatically from an email attachment. The download behavior from a server location depends on the operating system of the end user's device. For example, a MAF application end user with an Android device downloads the certificate to Android's Download directory. Once downloaded, the end user extracts (opens) the certificate in order to install it to the MAF application's keystore. To complete this step, the end user must enter a password provided by the administrator who distributed the client certificate. After the end user has installed the client certificate in their application's keystore, the MAF application can present it to a server to establish a two-way SSL session.

Example 29–2 Client Certificate File Extension in maf-application.xml File

```
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                  version="1.0" name="twowayssltest"
id="com.company.twowayssltest"
                  appControllerFolder="ApplicationController"
listener-class="application.LifeCycleListenerImpl"
                  client-ssl-certificate-extension="cert">
                 ….
```

30

Testing and Debugging MAF Applications

This chapter provides information on testing and debugging MAF applications.

This chapter includes the following sections:

- Section 30.1, "Introduction to Testing and Debugging MAF Applications"
- Section 30.2, "Testing MAF Applications"
- Section 30.3, "Configuring OEPE and MAF Applications to Debug Code"
- Section 30.4, "Debugging MAF Applications Deployed on the Android Platform"
- Section 30.5, "Debugging MAF Applications Deployed on the iOS Platform"
- Section 30.6, "Using and Configuring Logging in MAF Applications"
- Section 30.7, "Measuring MAF Application Performance"
- Section 30.8, "Sending Diagnostic Information to Oracle Mobile Cloud Service"
- Section 30.9, "Sending Analytics Information to Oracle Mobile Cloud Service"

30.1 Introduction to Testing and Debugging MAF Applications

To test or debug your MAF application, deploy it in debug mode to a device on one of the platforms (Android or iOS) that MAF supports. MAF and respective platform provides tools that let you connect the OEPE development environment with the MAF application executing on a device or a virtual device. For example, if you want to test your MAF application on an Android device, you deploy your MAF application in debug mode from OEPE to an Android device or to an Android Virtual Device (AVD). iOS provides similar tools.

> **Note:** Debugging is not available for MAF applications developed for Universal Windows Platform.

The high-level steps to debug a MAF application include the following tasks:

1. Use a debug configuration to deploy the MAF application to the test environment so that it deploys the MAF application in debug mode.
2. Deploy the MAF application to the test environment.
3. Use the appropriate tools for the debugging task that you want to complete. If, for example, you want to debug Java code in your MAF application, use the tools that OEPE provides. If you want to debug the code that renders the user interface

(HTML, CSS, or JavaScript) of your MAF application, use the tools that each platform provides for this task.

MAF provides other features to assist testing your application. These include the ability to monitor your application's performance, plus send analytics and diagnostic information to Oracle Mobile Cloud service (if your application accesses resources from that service).

30.2 Testing MAF Applications

There are two approaches to testing a MAF application:

1. Testing on a mobile device: this method always provides the most accurate behavior, and is also necessary to gauge the performance of your application. However, you may not have access to all the devices on which you want to test, making device testing inconclusive.
2. Testing on a mobile device emulator or simulator: this method usually offers better performance and faster deployment, as well as convenience. However, even though a device emulator or simulator closely approximates the corresponding physical device, there might be differences in behavior and limitations on the capabilities that can be emulated.

Typically, a combination of both approaches yields the best results.

30.2.1 How to Perform Accessibility Testing on iOS-Powered Devices

You should use a combination of the following methods to test the accessibility of your MAF application developed for iOS-powered devices:

- Testing with the Accessibility Inspector on an iOS-powered device simulator.

  For detailed information, see the "Testing the Accessibility of Your iPhone Application" section in the *Accessibility Programming Guide for iOS* available through the iOS Developer Library.

- Testing with the VoiceOver on an iOS-powered device.

  For more information, see the "Using VoiceOver to Test Your Application" section in the *Accessibility Programming Guide for iOS* available through the iOS Developer Library.

30.3 Configuring OEPE and MAF Applications to Debug Code

OEPE is equipped with debugging mechanisms that allow you to execute a Java program in debug mode and use standard breakpoints to monitor and control execution of an application.

Since a MAF application cannot be run inside OEPE, the debugging approach is different: you can use the OEPE debugger to connect to a Java Virtual Machine instance on a mobile device or simulator and control the Java portions of your deployed MAF application.

MAF automatically configures the project properties for debugging (see Section 30.3.1, "What You May Need to Know About the Debugging Configuration"). The following are the steps you need to take to use OEPE to debug the Java code in your MAF application:

To test or debug an application:

1. From OEPE's main menu, click **Run > Debug Configuration** to select a debug configuration.
2. Click **Debug** to run the application with debugging enabled.

For additional information, see the following:

- Section 30.3.1, "What You May Need to Know About the Debugging Configuration"

30.3.1 What You May Need to Know About the Debugging Configuration

When you create a new MAF debug configuration, OEPE automatically configures the application for debugging. These debug configurations allow you to debug a MAF application by clicking the OEPE Debug button in the configuration. Once the application has been deployed, it automatically starts with the debugger.

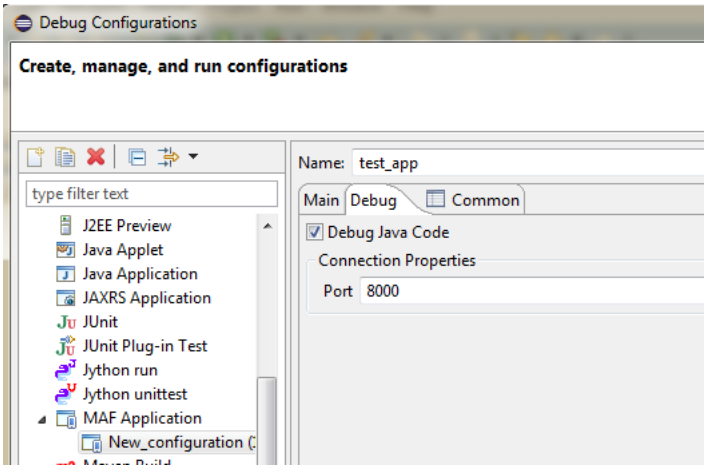
For information on how to create and edit run configurations, see Section 30.3.1.1, "Creating and Configuring a Debug Configuration."

30.3.1.1 Creating and Configuring a Debug Configuration

To create a new configuration or to modify an existing one, complete the Debug Configuration dialog as follows:

1. From OEPE's main menu, click **Run > Debug Configurations** to open the Debug Configuration dialog.
2. Create a new debug configuration by right-clicking **MAF Application** and choosing **New**.
3. Complete the Main tab of the dialog as follows:
   - For Android, follow the steps in Section 27.3.1, "How to Create an Android Deployment Configuration"
   - For iOS, follow the steps in Section 27.4.1, "How to Create an iOS Deployment Configuration"
4. In the Debug tab, select **Debug Java Code** and set the port to the appropriate port number, as shown in Figure 30–1.

Figure 30–1 Creating a Debug Configuration



Note: To avoid timeout, start the debugger soon after launching the application on the mobile device or simulator.

If the deployment is successful, and the application on the device is started in debug mode, it will block until remote debugging is connected. The timeout is usually a few minutes.

On Android

1. Ensure that port forwarding is established

On iOS

1. Ensure that your development computer and mobile device are visible to each other through TCP (they can ping each other).
2. Modify the Host field of the Remote Java Application configuration by replacing the localhost with the IP address of the mobile device.

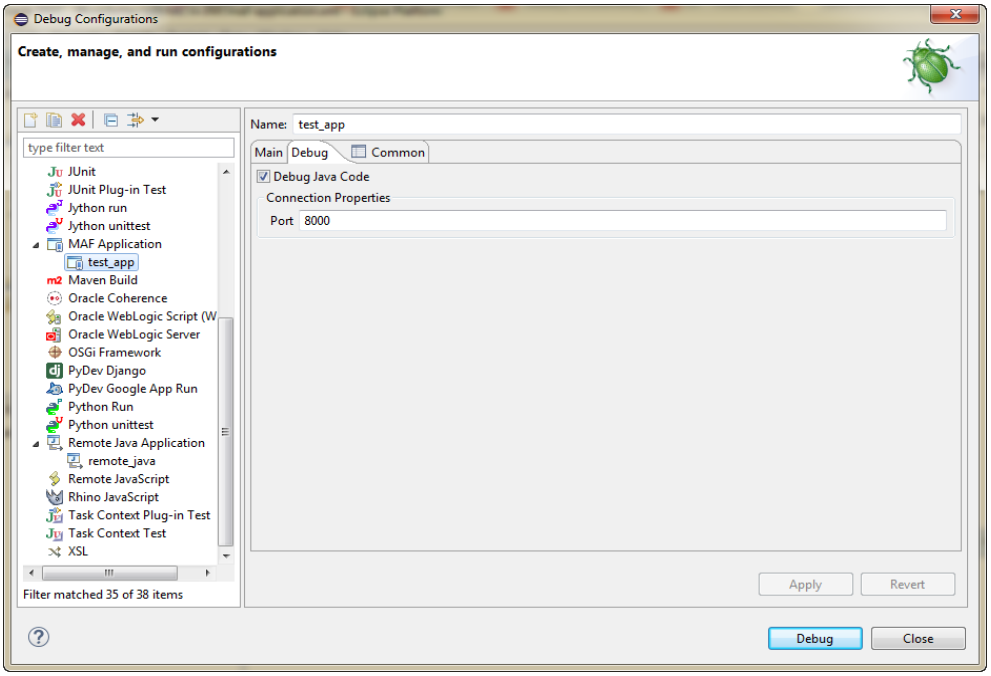
For additional information, see Section 30.3.1, "What You May Need to Know About the Debugging Configuration"

30.3.2 How to Enable Debugging of Java Code and JavaScript

You use the application's deployment configuration to specify either the release or debug execution mode for your MAF application. Only the debug mode enables you to interactively debug Java and JavaScript code. The debug mode allows for inclusion of special debugging libraries and symbols at compile time.

Figure 30–2 shows how to set the debug mode option.

Figure 30–2 Setting Debug Mode



For more information, see the following:

- Section 27.3.1, "How to Create an Android Deployment Configuration"

- Section 27.4.1, "How to Create an iOS Deployment Configuration"
- Section 27.4.9.1, "Creating iOS Development Certificates"

A `maf.properties` file allows you to specify startup parameters for the JVM and web views of MAF to enable debugging of the Java code and JavaScript. The `maf.properties` file is automatically created and placed in the `{assembly_project}/META-INF` directory (see Section 30.6, "Using and Configuring Logging in MAF Applications"), which corresponds to the *<application_name>*`/META-INF` location in your application file system.

You can use the following debugging properties in the `maf.properties` file:

- `java.debug.enabled`: Enables or disables Java debugging for MAF. Valid values are `true` and `false`.

  Caution: When `java.debug.enabled` is set to `true`, the JVM waits for a debugger to establish a connection to it. Failure of the debugger to connect will result in the failure of the MAF AMX application feature to load.

- `java.debug.port`: Specifies the port to be used during debugging. The valid value is an integer.
- `javascript.debug.enabled`: Enables or disables JavaScript debugging when the application is running in the device simulator. Valid values are `true` and `false`.
- `javascript.debug.feature`: Specifies the application feature that is to trigger the activation of JavaScript debugging in MAF. The format of the value is `featureId:port`. The port must be specified (it is initially set to a placeholder value).

  Note: The `javascript.debug.enabled` and `javascript.debug.feature` settings are only valid on iOS and Safari versions earlier than 6.0.

  If both iOS and Safari versions are later than 6.0, then neither of these two properties should be specified. Instead, follow the instructions from Section 30.3.2.1, "What You May Need to Know About Debugging of JavaScript Using an iOS-Powered Device Simulator on iOS 6 Platform."

The contents of the `maf.properties` file may be similar to the following:

```
java.debug.enabled=true
java.debug.port=8000

javascript.debug.enabled=true
javascript.debug.feature=products:8888
```

After the `maf.properties` file has been configured to debug JavaScript, you can navigate to the following URL to see a listing of all the loaded pages that can be debugged in MAF:

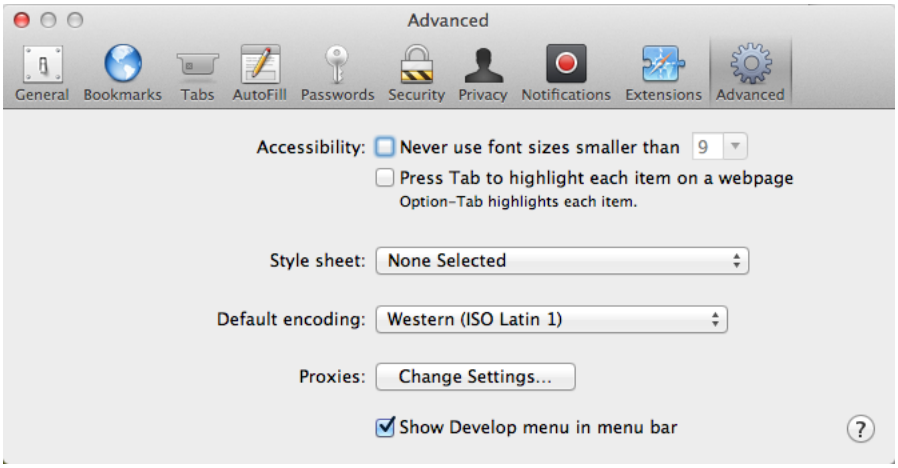
```
http://localhost:9999
```

For information on how to use OEPE to debug the Java code, see Section 30.3, "Configuring OEPE and MAF Applications to Debug Code."

30.3.2.1 What You May Need to Know About Debugging of JavaScript Using an iOS-Powered Device Simulator on iOS 6 Platform

If you are working with the iOS 6 platform, you can use the Safari 6 browser to debug JavaScript. To do so, open the Safari preferences, select **Advanced**, and then enable the Develop menu in the browser by selecting **Show Develop menu in menu bar**, as shown in Figure 30–3.

Figure 30–3 Enabling Safari Browser Options



When the Develop menu is enabled, select either **iPhone Simulator** or **iPad Simulator**, as Figure 30–4 and Figure 30–5 show, and then select a UIWebView that you are planning to debug. Whether the Develop menu displays an iPhone Simulator or iPad Simulator option depends on which device simulator is launched.

Figure 30–4 Using Develop Menu on Safari Browser for Debugging on iPhone Simulator

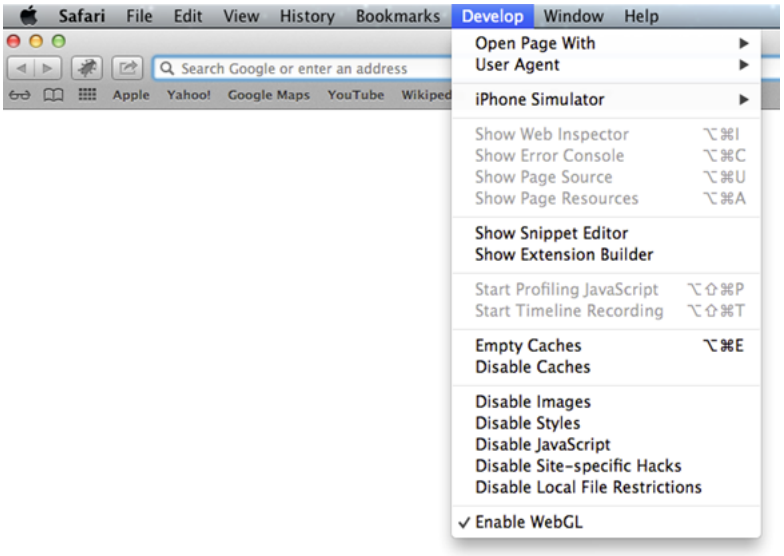
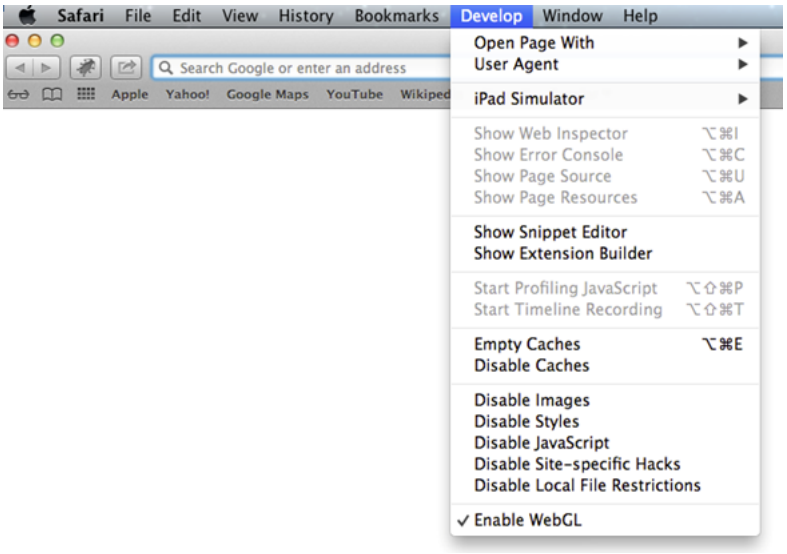


Figure 30–5 Using Develop Menu on Safari Browser for Debugging on iPad Simulator



30.3.3 How to Debug the MAF AMX Content

If your MAF application includes the MAF AMX content, after you configure the device or emulator, you can set breakpoints, view the contents of variables, and inspect the method call stack just as you would when debugging other types of applications in OEPE.

> **Note:** You can only debug your Java code and JavaScript (see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript"). Debugging of EL expressions or other declarative elements is not supported.

30.4 Debugging MAF Applications Deployed on the Android Platform

To debug a MAF application on the Android platform using OEPE, follow the generic debugging procedure described in Section 30.3, "Configuring OEPE and MAF Applications to Debug Code."

To debug Java code, you must create a debug configuration and configure the debug mode in it. For more information about these tasks, see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript" and Section 30.3, "Configuring OEPE and MAF Applications to Debug Code." Once you complete these tasks, you can deploy your MAF application in debug mode to debug your Java code, as described in Section 30.4.1, "How to Debug Java Code on the Android Platform"

To debug UI code (JavaScript, HTML, and CSS), you configure the debug mode in your debug configuration. Once you complete this task, you can debug your UI code, as described in Section 30.4.2, "How to Debug UI Code on the Android Platform"

30.4.1 How to Debug Java Code on the Android Platform

To debug a MAF application's Java code on the Android platform using OEPE, follow the debugging procedure described in Section 30.3, "Configuring OEPE and MAF Applications to Debug Code."

For information on how to configure an Android-powered device or emulator and how to deploy a MAF application for debugging, see Section 27.3.2, "How to Deploy an Android Application to an Android Emulator."

To allow debugging of a MAF application running on an Android-powered device or its emulator, verify that the Network Information plugin is enabled, as described in Introduction to Chapter 9, "Using Plugins in MAF Applications".

30.4.1.1 Troubleshooting adb

When you debug Java code, either on an Android-powered device connected through USB or on an Android-powered device emulator, the final step of deployment automatically executes port forwarding. This is the same as if you executed the following command on a terminal:

- For the device debugging:

  ```
  adb -d forward tcp:8000 tcp:8000
  ```

- For the emulator debugging:

  ```
  adb -e forward tcp:8000 tcp:8000
  ```

Sometimes the adb process freezes and you need to kill the process and restart it.

To kill the process, use:

- Mac terminal: use the `kill -9` *`procID`* command

Restart the adb daemon by executing the following command on a terminal:

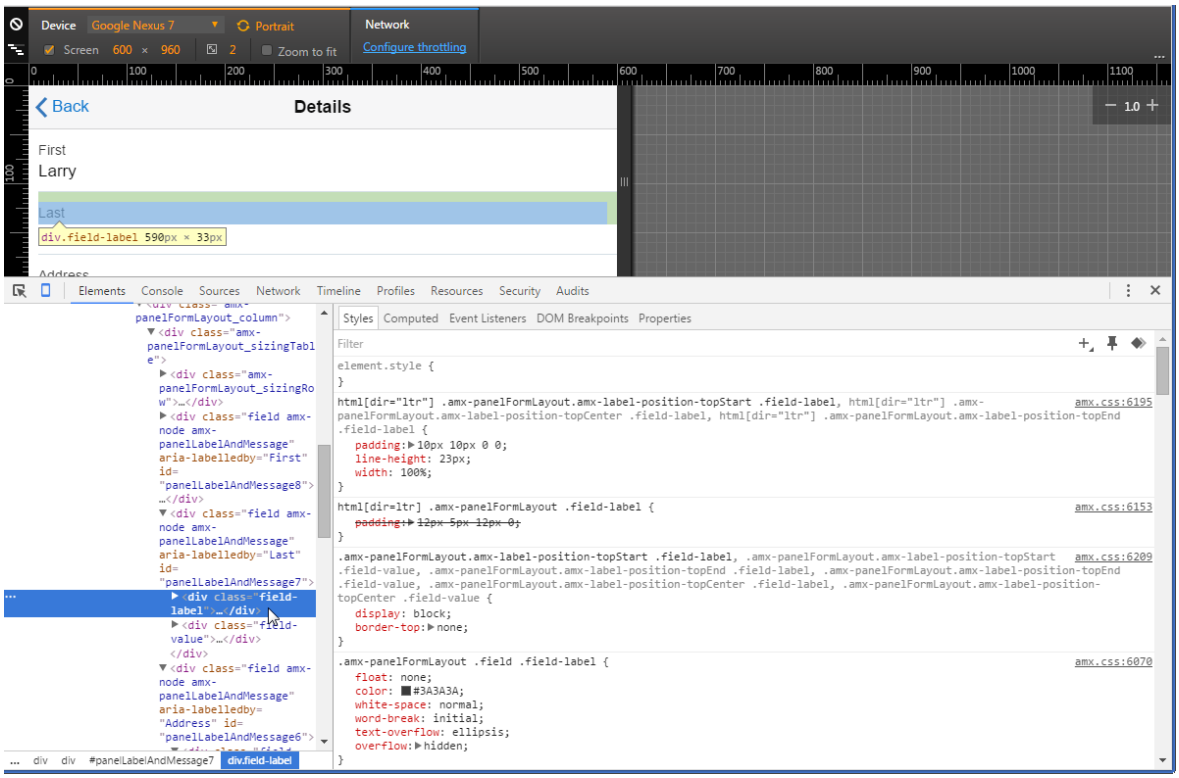
```
adb devices
```

A deployment that succeeded before adb froze will still be deployed. To debug an application, redeploy it.

30.4.2 How to Debug UI Code on the Android Platform

When developing a MAF application, you may need to debug code that renders the user interface (UI) of your application on an Android device. The code that renders the UI can include JavaScript, HTML, and CSS. You can debug this code using Google's Chrome DevTools when you deploy the MAF application from your development machine to the Android device. Figure 30–6 shows the Chrome DevTools inspecting an AMX page from a MAF application.

Figure 30–6 Chrome DevTools Inspecting an AMX Page from a MAF application



For more information on the Chrome DevTools, including the requirements to use it, see Remote Debugging on Android with Chrome DevTools on the Google Developers' site.

See also the "Debugging HTML in Oracle MAF Applications on Android" video on the Oracle Mobile Platform YouTube channel for an overview of how to debug UI code on Android. Note that the latter video makes reference to a `cvm.properties` file. This file has been renamed to `maf.properties`.

To deploy a MAF application to an Android device to debug its UI code, you need to:

- Configure the `maf.properties` file to include the following entry:
  `javascript.debug.enabled=true`

  For more information on the maf.properties file, see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript."

- Deploy in debug mode. For more information, see Chapter 27, "Deploying MAF Applications."

30.5 Debugging MAF Applications Deployed on the iOS Platform

Before you can debug Java code, you must create a debug configuration and configure the debug mode in it. For more information about these tasks, see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript" and Section 30.3, "Configuring OEPE and MAF Applications to Debug Code.". Once you complete these tasks, you can deploy your application in debug mode to the iOS device and debug your Java code. For more information about how to deploy in debug mode, see Section 30.5.1, "How to Debug Java Code on the iOS Platform."

To debug UI code (JavaScript, HTML, and CSS), you configure the debug mode in the debug configuration. Once you complete this task, you can debug your UI code, as described in Section 30.5.2, "How to Debug UI Code on the iOS Platform."

30.5.1 How to Debug Java Code on the iOS Platform

To debug a MAF application's Java code on the iOS platform using OEPE, follow the debugging procedure described in Section 30.3, "Configuring OEPE and MAF Applications to Debug Code.".

30.5.2 How to Debug UI Code on the iOS Platform

If you are working with the iOS platform, you can use the Safari browser to debug JavaScript. To do so, open the Safari preferences, select Advanced, and then enable the Develop menu in the browser by selecting Show Develop menu in menu bar.

When the Develop menu is enabled, select either **iPhone Simulator** or **iPad Simulator**, as Figure 30–3 and Figure 30–4 show, and then select a UIWebView that you are planning to debug, as Figure 30–5 shows.

Note: Whether the Develop menu displays an iPhone Simulator or iPad Simulator option depends on which device simulator is launched.

Use the `featureContentDelay` additional build argument to record log messages and errors from your custom JavaScript before the first page from your application loads. This argument specifies a delay before the WebView is populated with content. Set the additional build argument `-featureContentDelay` to 20.

Figure 30–7 Using Develop Menu on Safari Browser for Debugging on iPhone Simulator

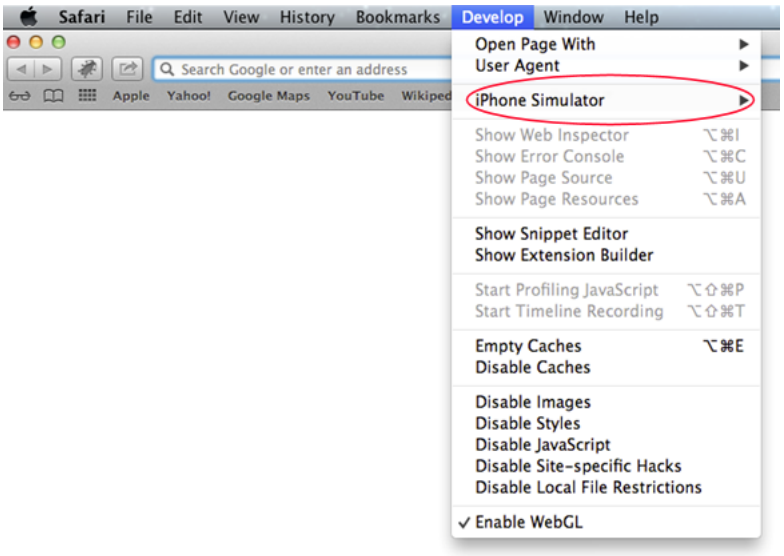


Figure 30–8 Using Develop Menu on Safari Browser for Debugging on iPad Simulator

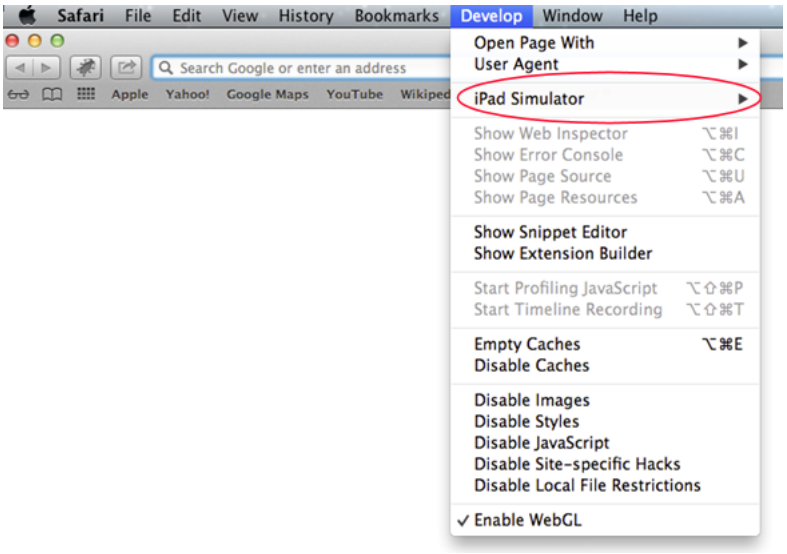


Figure 30–9 Using Develop Menu on Safari Browser to Select UIWebView

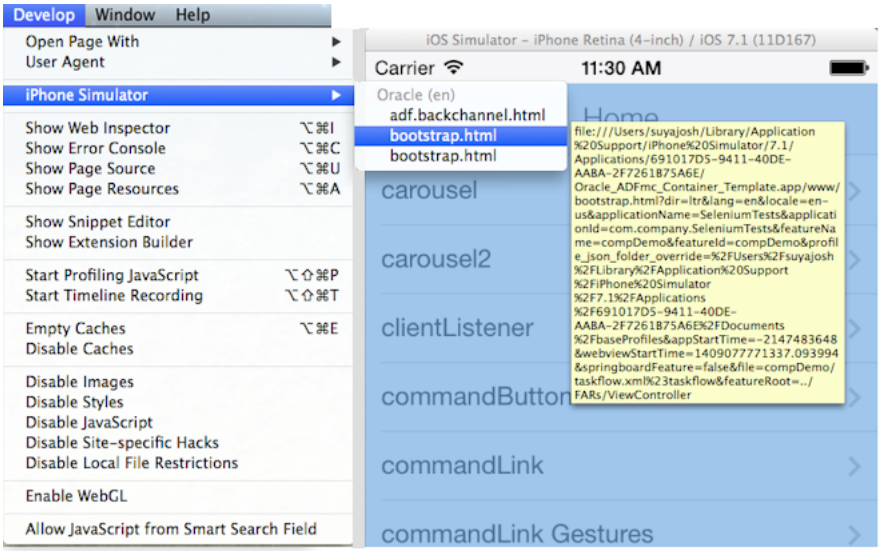


Figure 30–10 Remote Web Inspector

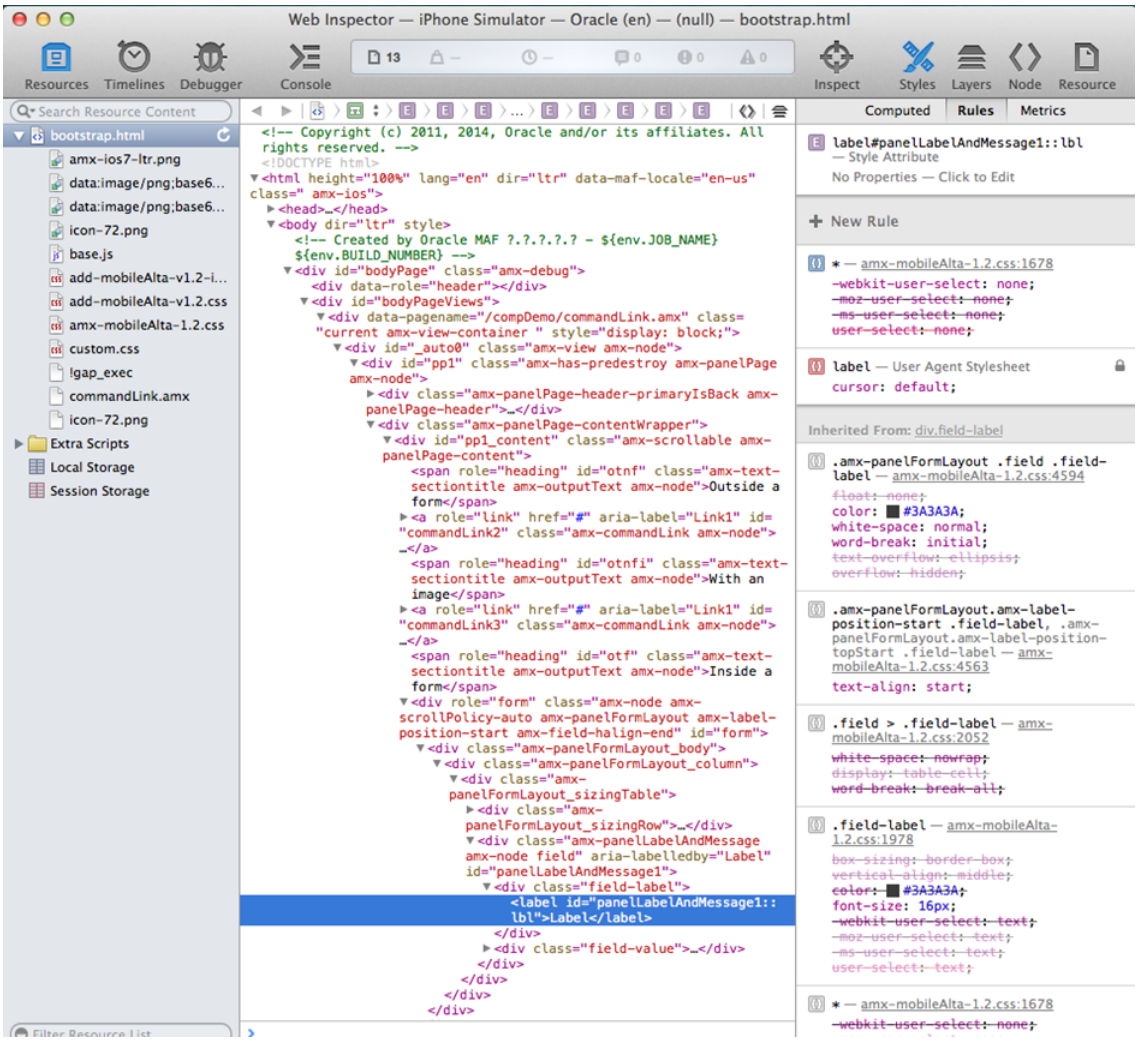


Figure 30–11 AMX Page Analyzed by Remote Web Inspector at Runtime

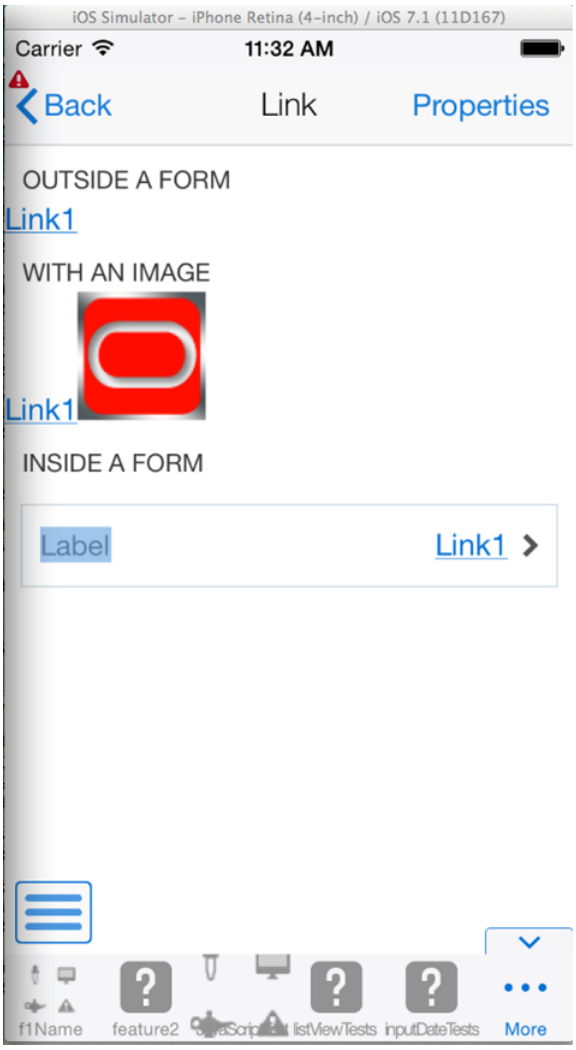


Figure 30–11 and Figure 30–12 show JavaScript debugging using breakpoints inside the Safari browser.

Figure 30–12 JavaScript Debugging in Safari Browser

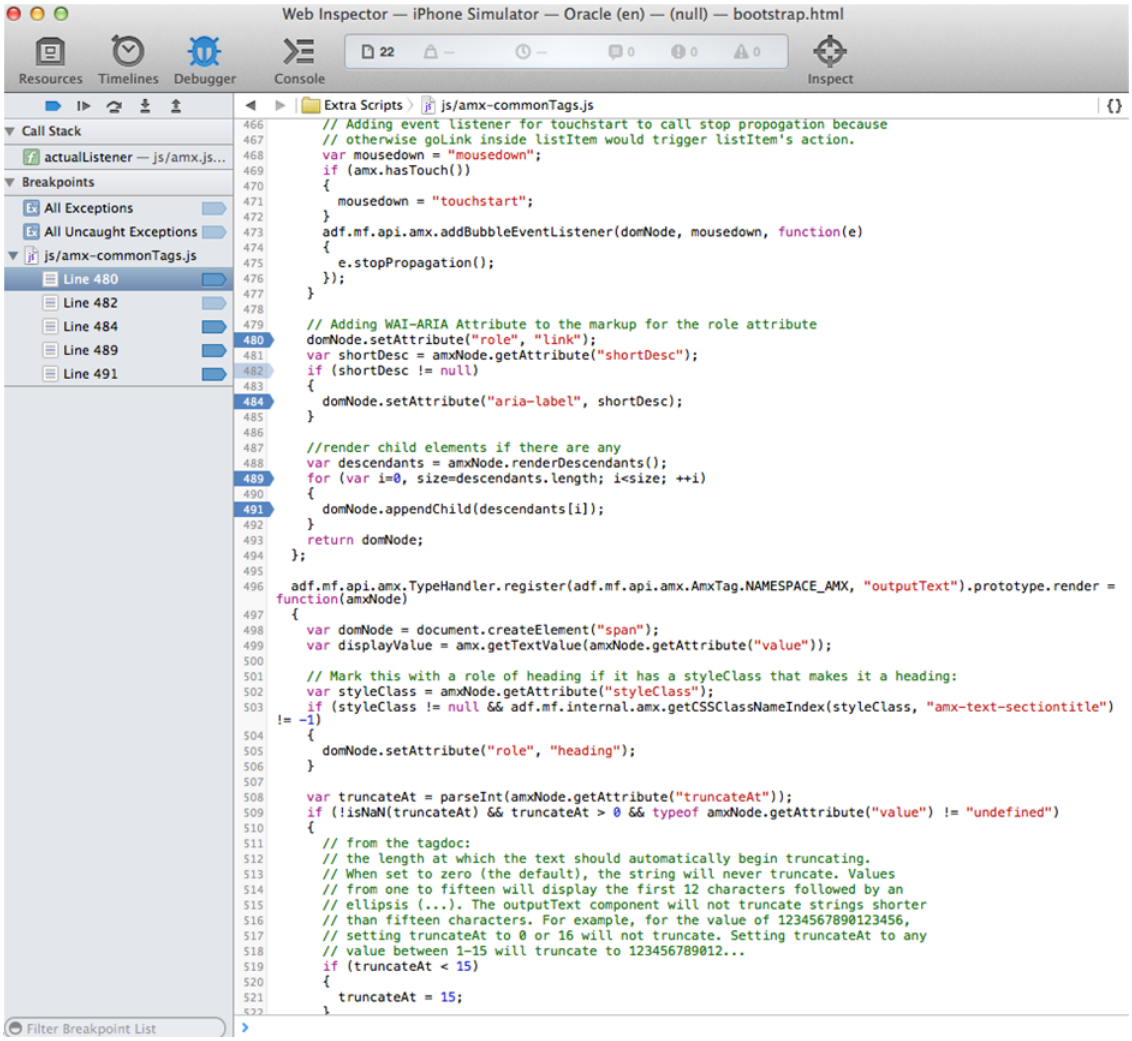
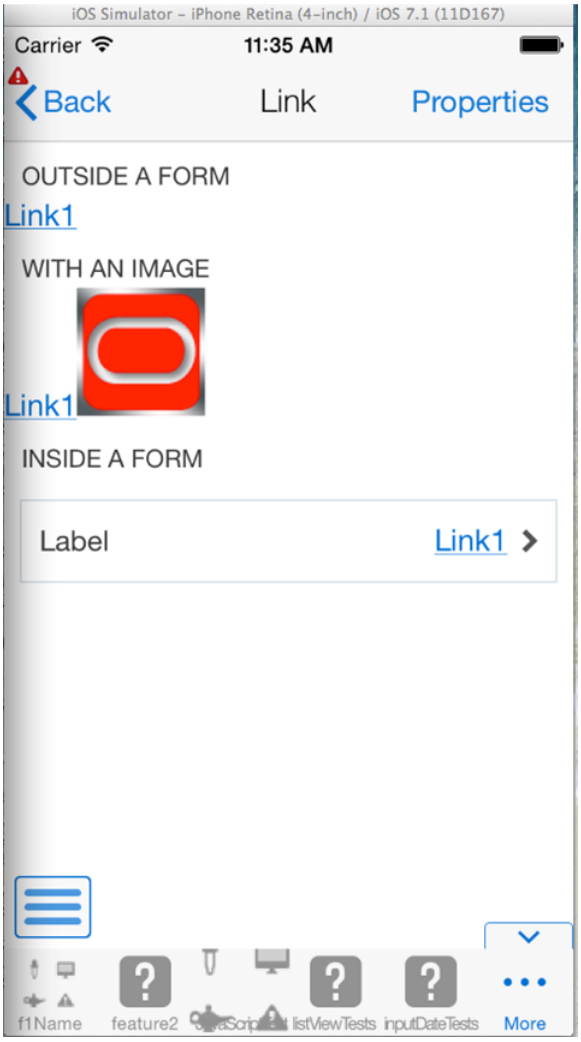


Figure 30–13 AMX Page Debugged at Runtime



30.6 Using and Configuring Logging in MAF Applications

For your MAF application, you can enable logging on all supported platforms through JavaScript (see Section 30.6.2, "How to Use JavaScript Logging") and embedded code (see Section 30.6.3, "How to Use Embedded Logging") using a single configuration with the log output directed to a single file. This log output includes the output produced by `System.out.println` and `System.err.println` statements.

The default MAF's logging process is as follows:

- The logging begins at application startup.
- The existing log file from the previous application run is deleted, so only the contents of the current run are available.
- When you are running your application on an iOS-powered device simulator, all logging output is typically sent to the console which you can access through the `Application/Utilities` directory on your development computer. However, if your development computer is running on Mac OS 10.8.*n*, you can only access the Java logging output through a file of whose name and location you are notified as soon as the output redirection occurs and the file is generated. One of the possible

locations for this file is `/Users/<userid>/Library/Application Support/iPhone Simulator/6.0/Applications/<AppID>/Documents/logs/application.log`

When you are running your application on an iOS-powered device, the console output is redirected to an `application.log` file that is placed in the `Documents/logs` directory of your application.

On Android, the output is forwarded to a text file with the same name as the application. The output file location is `/sdcard`. If this location is not present or is configured as read-only, the log output is rerouted to the application's writable data directory.

- The `logging.properties` file is automatically created and placed in the `<assembly project>/META-INF` location in your application file system (see Section 30.6, "Using and Configuring Logging in MAF Applications"). In this file, it is defined that all loggers use the `com.sun.util.logging.ConsoleHandler` and `SimpleFormatter`, and the log level is set to `SEVERE`. You can edit this file to specify different logging behavior (see Section 30.6.1, "How to Configure Logging Using the Properties File").

  Note: In your MAF application, you cannot use loggers from the `java.util.logging` package.

  MAF loggers are declared in the `oracle.adfmf.util.Utility` class as follows:

```
public static final String APP_LOGNAME = "oracle.adfmf.application";
public static final Logger ApplicationLogger = Logger.getLogger(APP_LOGNAME);

public static final String FRAMEWORK_LOGNAME = "oracle.adfmf.framework";
public static final Logger FrameworkLogger = Logger.getLogger(FRAMEWORK_
LOGNAME);
```

  The logger that you are to use in your MAF application is the `ApplicationLogger`.

  You can also use methods of the `oracle.adfmf.util.logging.Trace` class.

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

  When you are running your application on an iOS-powered device, the console output is redirected to an application.log file that is placed in the `Documents/logs` directory of your application. You can access this directory as follows on your iOS-powered device:

  1. Navigate to Xcode > Devices.
  2. Select the application from the list in the Installed Apps section.
  3. Click the gear icon.
  4. Select Download Container.
  5. Right-click the downloaded `*.xcappdata` file and select Show Package Contents.
  6. Open AppData > Documents > Logs.
  7. Right-click the downloaded `*.xcappdata` file and select Show Package Contents.

  On Android, the output is forwarded to a text file with the same name as the application. The output file location is `/sdcard`. If this location is not present or is

configured as read-only, the log output is rerouted to the application's writable data directory. The contents of the log file is replicated in the Android Logcat utility (see http://developer.android.com/tools/debugging/debugging-log.html). OEPE displays the logging output from the Logcat when you use OEPE to deploy your MAF application to an Android-powered device or emulator.

For both iOS and Android, the logging output appears in OEPE's log window immediately after the deployment. If you use a Run/Debug configuration to build, deploy, and launch the MAF application, the Terminate button in the toolbar terminates the MAF application along with the process that performs the log redirection.

- The `logging.properties` file is automatically created and placed in the `META-INF` directory under the assembly project which corresponds to the `<application_name>/META-INF` location in your application file system. In this file, it is defined that all loggers use the `java.util.logging.ConsoleHandler` and `SimpleFormatter`, and the log level is set to `SEVERE`. You can edit this file to specify different logging behavior (see Section 30.6.1, "How to Configure Logging Using the Properties File").

> **Note:** In your MAF application, you cannot use loggers from the java.util.logging package.

MAF loggers are declared in the oracle.adfmf.util.Utility class as follows:

```
public static final String APP_LOGNAME = "oracle.adfmf.application";
public static final Logger ApplicationLogger = Logger.getLogger(APP_LOGNAME);

public static final String FRAMEWORK_LOGNAME = "oracle.adfmf.framework";
public static final Logger FrameworkLogger = Logger.getLogger(FRAMEWORK_LOGNAME);
```

You can also use methods of the `oracle.adfmf.util.logging.Trace` class.

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

30.6.1 How to Configure Logging Using the Properties File

This example shows the `logging.properties` file that you use to configure logging.

```
# default - all loggers to use the ConsoleHandler
.handlers=com.sun.util.logging.ConsoleHandler
# default - all loggers to use the SimpleFormatter
.formatter=com.sun.util.logging.SimpleFormatter

oracle.adfmf.util.logging.ConsoleHandler.formatter=
             oracle.adfmf.util.logging.PatternFormatter
oracle.adfmf.util.logging.PatternFormatter.pattern=
             [%LEVEL%-%LOGGER%-%CLASS%-%METHOD%]%MESSAGE%

#configure the framework logger to only use the adfmf ConsoleHandler
oracle.adfmf.framework.useParentHandlers=false
oracle.adfmf.framework.handlers=oracle.adfmf.util.logging.ConsoleHandler
oracle.adfmf.framework.level=SEVERE

#configure the application logger to only use the adfmf ConsoleHandler
oracle.adfmf.application.useParentHandlers=false
oracle.adfmf.application.handlers=oracle.adfmf.util.logging.ConsoleHandler
oracle.adfmf.application.level=SEVERE
```

The `oracle.adfmf.util.logging.ConsoleHandler` plays the role of the receiver of the custom formatter.

The `oracle.adfmf.util.logging.PatternFormatter` allows the following advanced formatting tokens that enable log messages to be printed:

- `%LEVEL%`—the logging level.
- `%LOGGER%`—the name of the logger to which the output is being written.
- `%CLASS%`—the class that is being logged.
- `%METHOD%`—the method that is being logged.
- `%TIME%`—the time the logging message was sent.
- `%MESSAGE%`—the actual message.

The following logging levels are available:

- `SEVERE`: this is a message level indicating a serious failure.
- `WARNING`: this is a message level indicating a potential problem.
- `INFO`: this is a message level for informational messages.
- `FINE`: this is a message level providing tracing information.
- `FINER`: this level indicates a fairly detailed tracing message.
- `FINEST`: this level indicates a highly detailed tracing message.

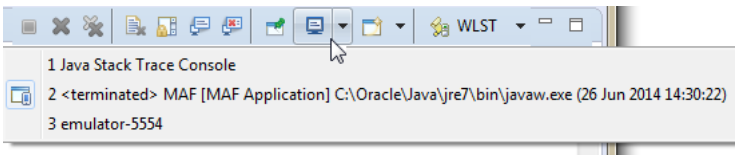
> **Caution:** When selecting the amount of verbosity for a logging level, keep in mind that by increasing the verbosity of the output at the `SEVERE`, `WARNING`, and `INFO` level negatively affects performance of your application.

There are two consoles in OEPE available when running applications on devices: iOS Console and for each device (including emulator) additional console for Android. Both types of console can stream the logging and standard output, using two additional `logging.properties` file properties which allow you to filter the output to the OEPE console:

```
oepe.console.filter.android=<some string>
oepe.sonsole.filter.ios=<some string>
```

You switch from one console to another using the Display Selected Console button in the Console menubar. To cycle through the consoles available, click . Alternatively, click the down arrow next to the button and choose from the list, as shown in Section 30–14, "Choosing the Console.".

Figure 30–14 Choosing the Console



The logger defined in the `logging.properties` file matches the logger obtained from the `oracle.adfmf.util.Utility` class (see Section 30.6, "Using and Configuring

Logging in MAF Applications"). The logging levels also match. If you decide to use the logging level that is more fine-grained than `INFO`, you have to change the `ConsoleHandler`'s logging level to the same level, as the example below shows.

```
oracle.adfmf.util.logging.ConsoleHandler.formatter=
             oracle.adfmf.util.logging.PatternFormatter
oracle.adfmf.util.logging.ConsoleHandler.level=FINEST
oracle.adfmf.util.logging.PatternFormatter.pattern=
             [%LEVEL%-%LOGGER%-%CLASS%-%METHOD%]%MESSAGE%
```

30.6.2 How to Use JavaScript Logging

JavaScript writes the output to the `console.log` or `.error/.warn/.info`. This output is redirected into the file through the `System.out` utility.

You customize the log output by supplying a message. The following JavaScript code produces `"Message from JavaScript"` output:

```
<script type="text/javascript" charset="utf-8">
   function test_function() { console.log("Message from JavaScript"); }
</script>
```

To make use of the properties defined in the logging file, you need to use the `adf.mf.log` package and the `Application` logger that it provides.

The following logging levels are available:

- `adf.mf.log.level.SEVERE`
- `adf.mf.log.level.WARNING`
- `adf.mf.log.level.INFO`
- `adf.mf.log.level.CONFIG`
- `adf.mf.log.level.FINE`
- `adf.mf.log.level.FINER`
- `adf.mf.log.level.FINEST`

To trigger logging, use the `adf.mf.log.Application` logger's `logp` method and specify the following through the method's parameters:

- the logging level
- the current class name as a String
- the current method as a String
- the message string as a String

The example below shows how to use the `logp` method in a MAF application.

```
adf.mf.log.Application.logp(adf.mf.log.level.WARNING,
                            "myClass",
                            "myMethod",
                            "My Message");
```

Upon execution of the `logp` method, the following output is produced:

```
[WARNING - oracle.adfmf.application - myClass - myMethod] My Message
```

For more information, see *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services*.

30.6.3 How to Use Embedded Logging

Embedded logging uses the `com.sun.util.logging.Logger`, as illustrated in the example below. Note that the `EmbeddedClass` represents a Java class defined in the project.

```
import com.sun.util.logging.Level;
import com.sun.util.logging.Logger;
import oracle.adfmf.util.logging.*;
...
   Utility.ApplicationLogger.logp(Level.WARNING,
                                  EmbeddedClass.class.getName(),
                                  "onTestMessage",
                                  "embedded warning message 1");
   Logger.getLogger(Utility.APP_LOGNAME).logp(Level.WARNING,
                    this.getClass().getName(),
                    "onTestMessage",
                    "embedded warning message 2");
   Logger.getLogger("oracle.adfmf.application").logp(Level.WARNING,
                                                     this.getClass().getName(),
                                                     "onTestMessage",
                                                     "embedded warning message 3");
```

The preceding code produces the following output:

```
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 1
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 2
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 3
```

30.6.4 How to Use Xcode for Debugging and Logging on iOS Platform

Even though it is not recommended to manipulate your MAF projects with Xcode because you can lose some or all of your changes during the next deployment with OEPE, you may choose to do so in exceptional circumstances.

Before you begin:

Deploy the application to the iOS simulator from OEPE.

To open the generated project directly in Xcode:

1. Navigate to the *workspace_directory*\deploy*deployment profile name*\temporary_xcode_project\.
2. Open the Xcode project called `Oracle_ADFmc_Container_Template.xcodeproj`.

If you are debugging your MAF application using Xcode, you cannot see the Java output in the IDE (on neither OEPE console nor Xcode console). Instead, the output is redirected to a file (see Section 30.6, "Using and Configuring Logging in MAF Applications"). By adding the following argument to your application's schema, you can disable this behavior and enable access to the Java, JavaScript, and Objective-C log output in Xcode in real time when debugging on either an iOS-powered device or simulator:

```
-consoleRedirect=FALSE
```

30.6.5 How to Access the Application Log

Using the following APIs, you can access the application log information:

- `oracle.adfmf.framework.api.PerfMon`
- `oracle.adfmf.framework.api.LogEntry`

- `oracle.adfmf.util.HOTS`

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

30.6.6 How to Disable Logging

You can prevent the logging output from being directed to the application log file, in which case the log file either remains blank or is not created in the first place. When logging is disabled, trace statements are absent from the application log and any output directed to `stderr` and `stdout` is redirected to either a `null` location or other location that is not accessible to the end user.

To disable all logging, set the `disableLogging` property to `true` in the application's `adf-config.xml` file, as follows:

```
<adf-property name="disableLogging" value="true"/>
```

By default, logging is enabled in MAF applications and the `disableLogging` property is set to `false`.

For information on the `adf-config.xml` file, see Appendix C.1, "Introduction to MAF Application and Project Files."

30.7 Measuring MAF Application Performance

MAF assists you in monitoring and measuring the performance of your MAF application. You can, for example, measure the time it takes for the following events in your application to complete:

- An action that a button invokes to complete
- A page to load
- A REST call to return a response

In addition, you can print statistics that show the mean and standard deviation time of operations that you monitor in your MAF application.

You enable performance measurement by configuring logging levels for the following performance loggers in your application's `logging.properties` file. The values assigned to the loggers in the following example are for illustrative purposes. See Table 30–1 for descriptions of all possible values.

```
oracle.adfmf.amx.useParentHandlers=false
oracle.adfmf.amx.handlers=oracle.adfmf.util.logging.ConsoleHandler
oracle.adfmf.amx.level=SEVERE

# used to control what monitors are captured in the list of monitors
oracle.maf.performance.monitor.captured.level = FINEST

# used to control what monitors are reported in the dumpStatistics
oracle.maf.performance.monitor.reported.useParentHandlers=false
oracle.maf.performance.monitor.reported.handlers=oracle.adfmf.util.logging.Console
Handler
oracle.maf.performance.monitor.reported.level = FINEST

# used to control what monitor observations (start/stop times) are logged.
oracle.maf.performance.monitor.observations.reported=false
oracle.maf.performance.monitor.observations.reported.handlers=oracle.adfmf.util.lo
gging.ConsoleHandler
oracle.maf.performance.monitor.observations.reported.level = FINEST
```

Once performance measurement completes, you can review the data that MAF collected while it measured the performance of your application. The following example shows an extract of the output that MAF produces after it monitors performance. In the example, the monitor observed 5 occurrences of a `Process AMX event` that had a mean completion time of `1435.5` milliseconds with a standard deviation of `1990.567`….

```
INFO - oracle.maf.performance.monitor.reported - MonitorFactory - dumpStatistics]
PERFMON-JAVA STATS: Monitor 'com.company.WorkBetter.**Perf_
Monitor**.Springboard.Container.Process AMX event' description:
'Time to process event'  observations: 5  mean: 1435.4 standard deviation:
1990.5674567821106
```

Table 30–1 describes the available performance monitor levels that you can set in the logging.properties file.

Table 30–1 Performance Monitor Levels

Level	Description
`Level.INFO`	This is the coarsest monitor level. It monitors events and actions. Using this level, you can, for example, monitor the loading of a page or the completion of an end user action, such as a button click. Use of this monitor level has minimal impact on performance.
`Level.FINE`	This level monitors more performance indicators that it orders into the following categories: ▪ JavaScript and rendering (System Level) ▪ Business logic processing (Application Level) ▪ Framework processing (System Level) ▪ External data access ▪ REST Calls (System Level) ▪ Database access (System Level) Use of this monitor level impacts the performance of your application and should not be enabled by default. Consider using this level to gain insight into how your code executes with a view to restructuring or refining your application.
`Level.FINER`	Use to monitor significant performance issues, such as how it long it takes to process nodes in an AMX page, data change events or EL expressions.
`Level.FINEST`	Use this monitor level to debug the performance of your application.

The monitor level that you specify in logging.properites takes effect when you first start the application.

> **Note:** MAF provides a number of setPerformanceMonitor methods in the oracle.adfmf.framework.api package's PerfMon class that enable you to change the performance monitor level at runtime. For more information, see the *Java API Reference for Oracle Mobile Application Framework*.

In addition to specifying a monitor level in the `logging.properties` file, you can add monitors to your application to collect the performance data. MAF uses the Java class,

`oracle.adfmf.performance.Monitor` (monitor), to collect performance data. A monitor is a stop watch that can be started, stopped and can add observations. By adding observations, you can use monitors to gain insights, such as the standard deviation for a given measurement. Each monitor has a unique ID and an optional description.

Monitor exposes a number of `addObservation()` methods that enable a monitor to measure the duration of an event or the number of occurrences of an event. When measuring duration, start the monitor before the event. After the event occurs, invoke an `addObservation()` method from the monitor. This stops the monitor. Duration is the time between the `start()` and the `addObservation()` method. You can restart a monitor that was never stopped. As you might expect, you cannot stop a monitor that was never started. An attempt to stop such a monitor logs an error.

A monitor that measures the number of occurrences of an event does not need to be started or stopped. Use the `addObservation(double duration)` method that does not stop the monitor, as demonstrated in the following example that counts the number of occurrences of JSON serialization in the application where you monitor performance. The `duration` parameter specifies the time since the monitor last added an observation.

The following example demonstrates how you create a monitor to measure the duration of an event. The example also shows a sample of the statistical information that this monitor produces.

```
import oracle.adfmf.performance.Monitor;
....

 public void measurePerformance()
  {
    Monitor monitor = null;

    try
    {
      //// Check that the appropriate monitor level is set before you create the
monitor
      if (Utility.PerformanceMonitorCaptured.isLoggable(Level.INFO))
      {
        monitor = MonitorFactory.getInstance().getMonitor("REST call", Level.INFO,
"REST call timing");
        monitor.start();
      }

      //
      // Perform your custom logic here:
      //
    }
    finally
    {
      if (monitor != null)
      {
        monitor.addObservation();
      }
    }
  }

  public void countCalls()
  {
    Monitor monitor = null;
```

```
    try
    {
      if (Utility.PerformanceMonitorCaptured.isLoggable(Level.FINE))
      {
        monitor = MonitorFactory.getInstance().getMonitor("Call count",
Level.FINE, "Count number of calls");
        monitor.start();
      }

      //
      // Perform your custom logic here:
      //
    }
    finally
    {
      if (monitor != null)
      {
        monitor.addObservation(1);
      }
    }
  }
```

To create a monitor that collects performance data in your application, you configure the logging.properties file to enable the performance monitor capture level to collect the data. In the example aboves, MAF will not collect performance data for the monitor if the application's logging.properties file does not contain the following entry:

```
# used to control what monitors are captured in the list of monitors
oracle.maf.performance.monitor.captured.level = FINEST
```

For more information about monitor (`oracle.adfmf.performance.Monitor`), see *Java API Reference for Oracle Mobile Application Framework*.

In addition to monitor, MAF also provides `oracle.adfmf.performance.Story` (story). A story allows you to start and end the collection of performance data. Once you end collection of the performance data for a story, MAF presents a hierarchical view of the collected performance data using a story ID that you assigned when you started the story. The hierarchical view shows the individual timing measurements for events that took place during the story. In addition, MAF performs a health of the system (HOTS) checkpoint at the end of the story. As part of this HOTS checkpoint, standard deviations for all of the monitor data collected during the story will be computed, so you can gain insight into how individual story events measure up statistically. All monitor data will then be cleared from the `MonitorFactory`.

Note: You can perform a HOTS checkpoint independently of a story by invoking the oracle.adfmf.util.HOTS.checkpoint() method. This determines information for your application, such as total memory used by the JVM, free memory, used memory (total minus free), and the number of active features. The following shows a sample of the data returned by `checkpoint()`:

```
HOTS.memory.used (N/A) count: 1.0335056E7
HOTS.memory.free (N/A) count: 1.365112E7
HOTS.memory.total (N/A) count: 2.3986176E7
HOTS.memory.max (N/A) count: 4.9152E7
HOTS.threads.active (N/A) count: 20.0
HOTS.features.active (N/A) count: 6.0
```

The following example shows a managed bean that exposes methods to start a story with a story ID (`**Perf_Monitor**`) and stop the story. This story could be started and ended from a button in the UI of the application that you want to measure the performance. For example, you may want to measure the loading of a page, so you expose a UI button that enables you to start the story prior to navigating to the page and another button that ends the story once the page loads.

```
package mobile;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.performance.Story;

public class PerfBean
{
  public PerfBean()
  {
    super();
  }
  public void startStory(ActionEvent ae)
  {
    Story.startStory("**Perf_Monitor**");
  }

  public void endStory(ActionEvent ae)
  {
    Story.endStory();
  }

}
```

The performance monitor level that you specify in the logging.properties file determines how much data the story captures. That is, a `logging.properties` file entry of `oracle.maf.performance.monitor.reported.level = FINEST` produces a more verbose story than an entry of `INFO`. When the story ends, all performance data captured during a story is sorted based on timing. This sorting presents monitor observations corresponding to JavaScript events at a correct time, since the time when these observations were recorded might be different from the time when the measured events took place. MAF then iterates through the sorted records and produces indented output, as demonstrated in the following example.

MAF writes the story into the application log file. A sample output for an application that navigates from one AMX page to another AMX page might look like the following:

Example 30–1 MAF Performance Monitor Data Generated using Story

```
[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
                      PERFMON-JAVA START: **Perf_
Monitor**.Navigation.Embedded.Story **Perf_Monitor** (Story Book) at 1452728427473

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
                      PERFMON-JAVA START: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                      /view1.amx event of type action on node cb2 (Time to process
event) at 1452728427416

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
                      PERFMON-JAVA STOP:  **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                      /view1.amx event of type action on node cb2 (Time to process
```

```
event) took: 14897.0ms
                     (started at 1452728427416)

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
                     PERFMON-JAVA START: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                     /view1.amx event of type action on node cb1 (Time to process
event) at 1452728442323

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
PERFMON-JAVA STOP:
                     **Perf_Monitor**.Navigation.Container.Process AMX event
Page:
                     /view1.amx event of type action on node cb1 (Time to process
event) took:
                     586.0ms (started at 1452728442323)

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
PERFMON-JAVA START:
                     **Perf_Monitor**.Navigation.Container.Load page /view2.amx
                     (Time to fully render the page) at 1452728442441

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
PERFMON-JAVA STOP:
                      **Perf_Monitor**.Navigation.Container.Load page /view2.amx
                     (Time to fully render the page) took: 468.0ms (started at
1452728442441)

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
PERFMON-JAVA START:
                      **Perf_Monitor**.Navigation.Embedded.Evaluate method
expression #{myBean2.endStory}
                      (UserSpace) at 1452728450665

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
PERFMON-JAVA STOP:
                      **Perf_Monitor**.Navigation.Embedded.Evaluate method
expression #{myBean2.endStory}
                      (UserSpace) took: 78.0ms (started at 1452728450665)

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - start]
PERFMON-JAVA START:
                      **Perf_Monitor**.Navigation.Container.Process AMX event
Page: /view2.amx
                      event of type action on node cb2 (Time to process event) at
1452728450626

[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
PERFMON-JAVA STOP:
                      **Perf_Monitor**.Navigation.Container.Process AMX event
Page: /view2.amx
                      event of type action on node cb2 (Time to process event)
                      took: 85.0ms (started at 1452728450626)
[INFO - oracle.maf.performance.monitor.observations.reported - Monitor - stop]
PERFMON-JAVA STOP:
                      **Perf_Monitor**.Navigation.Embedded.Story **Perf_Monitor**
(Story Book)
                      took: 23367.0ms (started at 1452728427473)

[INFO - oracle.maf.performance.monitor.reported - MonitorFactory - dumpStatistics]
```

```
PERFMON-JAVA STATS:
                      Monitor 'com.company.aPerfMonDocApp.**Perf_
Monitor**.Navigation.Container.Process AMX event'
                      description: 'Time to process event'  observations: 3
                      mean: 5189.333333333333 standard deviation:
8410.817102596711

[INFO - oracle.maf.performance.monitor.reported - MonitorFactory - dumpStatistics]
PERFMON-JAVA STATS:
                      Monitor 'com.company.aPerfMonDocApp.**Perf_
Monitor**.Navigation.Container.Load page'
                      description: 'Time to fully render the page'  observations:
1
                      mean: 468.0 standard deviation: NaN

[INFO - oracle.maf.performance.monitor.reported - MonitorFactory - dumpStatistics]
PERFMON-JAVA STATS:
                      Monitor 'com.company.aPerfMonDocApp.**Perf_
Monitor**.Navigation.Embedded.Evaluate
                      method expression' description: 'UserSpace'  observations:
1  mean: 78.0
                      standard deviation: NaN

[INFO - oracle.maf.performance.monitor.reported - MonitorFactory - dumpStatistics]
PERFMON-JAVA STATS:
                      Monitor 'com.company.aPerfMonDocApp.**Perf_
Monitor**.Navigation.Embedded.Story'
                      description: 'Story Book'  observations: 1  mean: 23367.0
standard deviation: NaN

1452728427473  0001.0001 [0000] Start: **Perf_Monitor**.Navigation.Embedded.Story
**Perf_Monitor** (INFO)
1452728442323    0002.0002 [0001] Start: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                                                        /view1.amx event of type
action on node cb1 (INFO)

1452728442441      0003.0003 [0002] Start: **Perf_
Monitor**.Navigation.Container.Load page /view2.amx (INFO)
1452728442909      0003.0003 [0002] Stop: **Perf_
Monitor**.Navigation.Container.Load page /view2.amx
                                                                (took = 468.0)
started at 1452728442441 (INFO)

1452728442909    0002.0002 [0001] Stop: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                                              /view1.amx event of type action on
node cb1 (took = 586.0)
                                              started at 1452728442323 (INFO)

1452728450626    0004.0002 [0001] Start: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                                              /view2.amx event of type action on
node cb2 (INFO)

1452728450665      0005.0003 [0004] Start: **Perf_
Monitor**.Navigation.Embedded.Evaluate
                                              method expression
#{myBean2.endStory} (INFO)
```

```
1452728450711    0004.0002 [0001] Stop: **Perf_
Monitor**.Navigation.Container.Process AMX event Page:
                                                /view2.amx event of type action on
node cb2 (took = 85.0)
                                                started at 1452728450626 (INFO)

1452728450743      0005.0003 [0004] Stop: **Perf_
Monitor**.Navigation.Embedded.Evaluate method
                                               expression #{myBean2.endStory} (took
= 78.0)
                                               started at 1452728450665 (INFO)

1452728450840  0001.0001 [0000] Stop: **Perf_Monitor**.Navigation.Embedded.Story
**Perf_Monitor**
                                               (took = 23367.0) started at
1452728427473 (INFO)
```

30.8 Sending Diagnostic Information to Oracle Mobile Cloud Service

MAF applications that access REST web services use `RestServiceAdapter` to access these services. If your application accesses REST services hosted by MCS and you want to use MCS Diagnostics to monitor and/or debug your application's calls to REST services hosted by MCS, create a `McsRestServiceAdapter` to send the following information to MCS:

- Mobile diagnostic session ID.

  This attribute maps an application session on a specific device. The application sends this information through the `Oracle-Mobile-DIAGNOSTIC-SESSION-ID` HTTP request header.

- Mobile device ID

  Correlates the REST API requests sent to MCS with the physical device that makes the request. The mobile application supplies this information through the `Oracle-Mobile-Device-ID` HTTP request header.

- Client request time,

  Indicates the API call time stamp that is captured on the client side immediately before the application submits the request. The mobile application supplies this information using the HTTP request header `Oracle-Mobile-CLIENT-REQUEST-TIME` attribute.

The following example shows the type of information that a MAF application using this type of adapter inserts into HTTP request headers:

```
Oracle-Mobile-Diagnostic-Session-ID: 19975
Oracle-Mobile-Device-ID: d09379504b0a3247
Oracle-Mobile-Client-Request-Time: 2016-02-09T09:03:17.777Z
```

The following example shows how you create the `McsRestServiceAdapter`:

```
...
import oracle.maf.api.dc.ws.rest.RestServiceAdapterFactory;
import oracle.maf.api.dc.ws.rest.RestServiceAdapter;

...
RestServiceAdapterFactory factory = RestServiceAdapterFactory.newFactory();
RestServiceAdapter mcsRestServiceAdapter = factory.createMcsRestServiceAdapter();
```

30.9 Sending Analytics Information to Oracle Mobile Cloud Service

A MAF application with one or more mobile backends (MBE) hosted on MCS can send analytics information about application usage to MCS Analytics.

Analytics information that MAF generates to send to MCS provides information about the application lifecycle and an end user's interaction with the MAF application. MAF categorizes analytics events into MAF Framework events and business events. You send information captured in response to MAF Framework events to MCS by configuring properties in your application's `logging.properties` file. Examples of MAF framework events includes application start, feature events like activate and feature navigation, and user authentication events. MAF logs these events by default when you configure your application to send analytics information to MCS.

The following example shows the payload that MAF generates and transfers to MCS for a `FeatureNavigation` MAF framework event that occurs when an end user is redirected to a login application feature (`LF1`) before navigating to secured application features (`secure-feature-1`, and `secure-feature-2`). MAF logs all MAF framework events within a session that starts when a MAF application that is configured to send analytics information activates. The session ends when the MAF application deactivates. In the following example, the `sessionID` property identifies the session.

OEPE's Log window displays this payload when you deploy your MAF application to a device in debug mode.

```
"name":"FeatureNavigation",
"properties":{"SourceId":"null","DestinationId":"LF1"}, "type":"custom",
"timestamp":"2015-11-06T20:35:27.384Z",

"sessionID":"com.company.MafAnalytics_736ad3d4-3443-4f65-8378-4e653ade2d30_
160121114922"

"name":"FeatureNavigation",
"properties":{"SourceId":"LF1","DestinationId":"secure-feature-1"},
"type":"custom",
             "timestamp":"2015-11-06T20:35:27.384Z",
"sessionID":"com.company.MafAnalytics_736ad3d4-3443-4f65-8378-4e653ade2d30_
160121114922"

"name":"FeatureNavigation",
"properties":{"SourceId":"secure-feature-1","DestinationId":"secure-feature-2"},
"type":"custom",
              "timestamp":"2015-11-06T20:35:27.384Z",
"sessionID":"com.company.MafAnalytics_736ad3d4-3443-4f65-8378-4e653ade2d30_
160121114922"
```

Business events are events that you (the application developer) define in your application. You capture the analytics information for the event using the APIs that MAF provides in `oracle.maf.api.analytics.AnalyticsUtilities`. MAF also exposes a data control method (`fireEventListener`) on the ApplicationFeatures data control. Drag and drop this data control method to an AMX page where you can configure it to listen for the event that you define. These APIs can also be used to send analytics from MAF Framework events to MCS.

MAF also enables you to send context event information (device model, country, time zone, and so on) to MCS or to another repository that you choose.

You can also send analytics to repositories other than MCS.

For more information, see:

- Section 30.9.1, "How to Configure the Transfer of Analytics to Oracle Mobile Cloud Service"
- Section 30.9.2, "How to Programmatically Send Analytics to Oracle Mobile Cloud Service"
- Section 30.9.3, "How to Send Context Events to Oracle Mobile Cloud Service"
- Section 30.9.4, "How to Send Analytics to Other Repositories"
- Section 30.9.5, "MAF Framework Events that Capture Analytics Information" (Describes the MAF Framework events that MAF provides.)
- For information about the classes in the `oracle.maf.api.analytics` package that you can extend and customize, see *Java API Reference for Oracle Mobile Application Framework*.

30.9.1 How to Configure the Transfer of Analytics to Oracle Mobile Cloud Service

MAF populates the logging.properties file in a new MAF application with the properties that you need to configure to send analytics information from your MAF application to MCS Analytics.

The following example shows the ready-to-use entries that the logging.properties file contains when you create a new MAF application.

```
# Configure the analytics logger
# Analytics events are logged only if oracle.maf.api.analytics.level=ALL
# Set to OFF or any level other than ALL to disable analytics
oracle.maf.api.analytics.level=ALL
oracle.maf.api.analytics.handlers=oracle.maf.api.analytics.LoggerAnalyticsHandler,
oracle.maf.api.analytics.McsAnalyticsHandler
oracle.maf.api.analytics.custom.level=INFO
oracle.maf.api.analytics.LoggerAnalyticsHandler.level=INFO

# Configure MCSHandler
oracle.maf.api.analytics.McsAnalyticsHandler.level=INFO
oracle.maf.api.analytics.McsAnalyticsHandler.connectionId=Mcs_Connection_Id
oracle.maf.api.analytics.McsAnalyticsHandler.batchSize=25
oracle.maf.api.analytics.McsAnalyticsHandler.offlineWrite=false
oracle.maf.api.analytics.McsAnalyticsHandler.recordUsername=false
oracle.maf.api.analytics.McsAnalyticsHandler.contextProviderClassName=oracle.maf.a
pi.analytics.McsContextProvider
```

Of the properties listed in the previous example, only `connectionId` is mandatory. Table 30–2 describes the optional properties. The `connectionId` property is where you define a valid connection to a MCS MBE. Use the `connectionId` defined in your application's `connection.xml` as the value for this property. If you do not specify a valid `connectionId`, MAF does not send events to MCS. Instead, MAF appends these events on disk until the maximum number of events allowed by the device is reached.

A valid `connectionId` in your application's connection.xml file uses the ID of the MBE (`oracle-mobile-backend-id`) and the application key (`oracle-mobile-application-key`) that is generated when the MAF application registers with the MBE. MAF adds these two values to the HTTP header that creates a connection to MCS. These values identify the MBE to which the MAF application sends analytic events and identify the application from which the analytics events originate.

Note: You do not need to register your MAF application as a client on MCS for all 3 platforms (Android, iOS, and Universal Windows Platform). Register the MAF application for one platform and use the generated application key as a value for `oracle-mobile-application-key`.

If the connection to the MCS MBE uses HTTP basic authentication type, associate `oracle/wss_http_token_client_policy` with the connection. For connections that use Oauth, associate `oracle/http_oauth2_token_mobile_client_policy` with the connection. If you do not associate the correct policy with the connection, MAF does not flush analytics events to MCS. For more information about associating a security policy with a connection, see Section 15.3, "Accessing Secure Web Services.".

The following example shows extracts of an application `connections.xml` file with values for the properties just discussed.

```
<References xmlns="http://xmlns.oracle.com/adf/jndi">
  <Reference name="Mcs_Connection_Id"
className="oracle.adf.model.connection.url.HttpURLConnection"
             adfCredentialStoreKey="McsLoginConn" xmlns="">
    ...
    <RefAddresses>
      <XmlRefAddr addrType="Mcs_Connection_Id">
        <Contents>
          <urlconnection name="Mcs_Connection_Id" url="http://mcs_
instance.oracle.com:7201"/>
    ...
  </Reference>
  <Reference name="McsLoginConn"
className="oracle.adf.model.connection.adfmf.LoginConnection"
             adfCredentialStoreKey="McsLoginConn" partial="false"
manageInOracleEnterpriseManager="true"
             deployable="true" xmlns="">
    <Factory
className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
    <RefAddresses>
      <XmlRefAddr addrType="adfmfLogin">
        <Contents>
          <login url="http://mcs_
instance.oracle.com:7201/mobile/platform/users/login"/>
          <logout url="http://mcs_
instance.oracle.com:7201/mobile/platform/users/logout"/>
          <customAuthHeaders>
            <header name="oracle-mobile-backend-id"
value="0e4a9dfa-046a-4aaa-b8dd-331044ad81f4"/>
            <header name="oracle-mobile-application-key"
value="be53201a-8674-48d7-96d0-bb02f4cd06c5"/>
          </customAuthHeaders>
    ...
</References>
```

Configure the following optional entries in the `logging.properties` file to implement the described functionality.

Table 30–2 Optional Properties to Manage the Transfer of Analytics from a MAF Application

Property	Description
`batchSize`	An optional property that determines the number of events saved locally before the MAF application sends them to an associated MCS instance. Events will be uploaded in batches to MCS. There is a limit on maximum batch size (65). If `batchSize` is not provided or exceeds the maximum limit of 65, a default `batchSize` of 25 applies.
`offlineWrite`	An optional property that determines if offline buffering of events should be enabled or not. It is possible that events get generated while a device is offline or the client is not able to establish a connection with MCS. Configure the `offlineWrite` property if you do not want to lose these events. Once the connection is re-established, MAF flushes these saved events to MCS. The `offlineWrite` property ensures that those events get cached to disk to enable their buffering while offline. MAF provides support for up to 250 events. Events will be saved on a rolling basis. That is, MAF saves the last 250 events. This ensures offline buffering of the latest events. MAF also flushes events to MCS when the application deactivates, irrespective of whether `batchSize` has been reached or not. By default, `offlineWrite` is disabled. Set it to True to enable it.
`recordUsername`	An optional property that determines if username should be captured or not. Set to `True` so that MAF captures the username when a user logs into a secured feature. If an application does not contain any secured feature or if the user has not yet logged into the secured feature then username remains null. If the application contains several secured features, then username will be updated based upon the credentials (username) used to log into that feature. The username captured will be sent as one of the fields in the context event. Therefore, if you intend to capture username, configure `logging.properties` so that the `recordUsername` property is True and `contextProviderClassName` has a valid class name for generating the context event. For example, `oracle.maf.api.analytics.McsAnalyticsHandler.recordUsername=true`.
`contextProviderClassName`	Optional. Provide a value for this property when the context event is generated. The value you specify determines the class that generates the context event for MCS. The context event contains information like timezone offset, geolocation and device information. It can also contain username if `recordUsername` is set to `True`. The `contextProviderClassName` property is disabled by default. The following example shows how you enable it: `oracle.maf.api.analytics.McsAnalyticsHandler.contextProviderClassName=oracle.maf.api.analytics.McsContextProvider`

30.9.2 How to Programmatically Send Analytics to Oracle Mobile Cloud Service

MAF provides an API in `oracle.maf.api.analytics.AnalyticsUtilities` that you can use to send events to MCS.

The `AnalyticsUtilities` class provides the following API:

```
public static void fireEvent(Level level, String category, String eventName)
// Send an event without a payload

public static void fireEvent(Level level, String category, String eventName,
```

```
JSONObject payload)
// Send an event with a JSON payload


level:     The logging level of the event. Set to any standard level supported by
Java logging.

category:  The category of the event. Set to 'custom' for all events except
context,
           sessionStart(MAF Framework event) and sessionEnd(MAF Framework event).
Set the latter events to 'system'.

eventName: Provide your own event name if you do not use an event provided in the
AnalyticsUtilities class.
           Throws an exception if null.

payload:   A JSONObject that contains key-value pairs for custom events. The
JSONObject must be of type String.
           No other data type is supported.
```

Configure your application's logging.properties file with the values necessary to transfer analytics to MCS before you use this API. Specify, for example, the MCS `connectionID`.

The following example demonstrates how to use this API to send analytics from a MAF application to MCS.

```
//  Sending events from your application.

        //  The following logs event when there is no payload to register for an
event.
        AnalyticsUtilities.fireEvent(Level.WARNING, AnalyticsUtilities.CATEGORY_
CUSTOM, "EVENT_VIDEO_ACTIONS");

        //  The following logs event when there is a JSON payload to send for a
custom event.
        try
                {
                  JSONObject   payload = new JSONObject();
                  payload.put("PAYLOAD_VIDEONAME", getFileName());
                  payload.put("PAYLOAD_ACTION", getAction());
                  // Creating a custom event 'EVENT_VIDEO_ACTIONS' of level INFO
                  AnalyticsUtilities.fireEvent(Level.INFO,
AnalyticsUtilities.CATEGORY_CUSTOM, "EVENT_VIDEO_ACTIONS" , payload);
                }

           catch(Throwable t)
                {
                // log the error
                }
```

You can set different log levels to capture events based on user interaction. For example, the `EVENT_VIDEO_ACTIONS` event in the previous example could be of `INFO` level when your end user performs a play action. Alternatively, it could be set to `WARNING` level to capture events in case of failure. You manage the logging level in the logging.properties file. For example, to manage the logging of `EVENT_VIDEO_ACTIONS` events, configure the logging.properties file as follows:

```
// Set to  WARNING to log events for the play action. Set to INFO (or a lower
level)
// to log events for the play action plus failure events
```

```
oracle.maf.api.analytics.custom.EVENT_VIDEO_ACTIONS.level=WARNING

// Disable logging of EVENT_VIDEO_ACTIONS
oracle.maf.api.analytics.custom.EVENT_VIDEO_ACTIONS.level=OFF
```

30.9.3 How to Send Context Events to Oracle Mobile Cloud Service

MAF applications can capture context events and send the collected information to MCS or to a repository other than MCS.

A context event defines the context of subsequent events until another context event is logged. An event can be logged from a MAF Framework event or from events that you define in your application.

Table 30–3 lists key names that MCS accepts in the key-value pairs of the JSON object(s) that transmit context events from the MAF application. MCS requires that all properties be of type String. All properties in the following table are optional.

> **Note:** MCS translates latitude and longitude information to city, state, country, and postal code. Provide latitude and longitude information or locality, region, postalCode and country, but not both.

Table 30–3 Valid Key Names to Send Context Event Information to MCS

Key Name	Description
userName	The user of the device who was logged in to the secured feature when the context events were logged.
timezone	Device's offset from UTC in seconds
model	Device's model name
osName	Device operating system name
osVersion	Device operating system version
latitude	Device's GPS latitude
longitude	Device's GPS longitude
locality	Device's locality
region	Device's region
postalCode	Device's postal code
country	Device's country

To send context event information from your MAF application to MCS, configure the following entry in your `logging.properties` file:

```
oracle.maf.api.analytics.McsAnalyticsHandler.contextProviderClassName=
oracle.maf.api.analytics.McsContextProvider
```

With this entry configured in your logging.properties file, MAF sends the context event information listed in Table 30–3 to MCS. You also need to set `recordUsername` to `True` in the logging.properties file if you want the MAF application to send the user name to MCS.

If you want to generate context events that contain fields you define, configure the following entry in your `logging.properties` file:

```
oracle.maf.api.analytics.McsAnalyticsHandler.contextProviderClassName=oracle.maf.d
```

```
emo.CustomContextProvider
```

Where `oracle.maf.demo.CustomContextProvider` is a class that implements `oracle.maf.api.analytics.ContextProvider`, as shown in Example 29-2. The generated context event contains the timezone, carrier, and manufacturer information.

Example 30–2 Custom Context Provider to Send Context Information

```
package oracle.maf.demo;

import java.util.Date;
import java.util.TimeZone;
import oracle.maf.api.analytics.ContextProvider;
import oracle.adfmf.json.JSONObject;

public class CustomContextProvider
  implements ContextProvider
  {
  public CustomContextProvider()
  {
    super();
  }

  public JSONObject generateContext()
  {
    JSONObject myCustomCtx = new JSONObject();

    //
    // TimeZone - Mobile device's offset from UTC in seconds
    //
    Date date = new Date();
    int offset = TimeZone.getDefault().getOffset(date.getTime()) / 1000;

    try
    {
      myCustomCtx.put("timezone", new Integer(offset).toString());
      myCustomCtx.put("carrier", "AT&T");
      myCustomCtx.put("manufacturer", "Apple");
    }
    catch(Exception ex)
    {
      ex.printStackTrace();
    }

    return myCustomCtx;
  }
  }
```

30.9.4 How to Send Analytics to Other Repositories

MAF applications can send analytics to repositories other than MCS.

To achieve this, you write a custom class that extends oracle.maf.api.analytics.McsAnalyticsHandler and overrides `processEvent()` method, as shown in the following example.

```
package oracle.maf.demo;

import java.io.IOException;
import oracle.adfmf.framework.exception.AdfException;
```

```
import oracle.adfmf.json.JSONArray;
import oracle.adfmf.resource.CDCErrorBundle;
import oracle.adfmf.util.ResourceBundleHelper;
import oracle.maf.api.analytics.McsAnalyticsHandler;
import oracle.maf.api.dc.ws.rest.RestServiceAdapter;
import oracle.maf.api.dc.ws.rest.RestServiceAdapterFactory;

public class CustomHandler extends McsAnalyticsHandler
{
  public CustomHandler()
  {
    super();
  }

  //
  // Establish the connection to a different repository and flush the events.
  //
  protected void processEvents() throws AdfException
  {
    // Extract the events to be flushed
    _events = super.getEvents();

    RestServiceAdapter restAdapter =
RestServiceAdapterFactory.newFactory().createRestServiceAdapter();
    restAdapter.clearRequestProperties();

    // Get valid connectionId of the repository.
    restAdapter.setConnectionName(getConnectionId());
    restAdapter.setRequestMethod(RestServiceAdapter.REQUEST_TYPE_POST);
    restAdapter.addRequestProperty("Content-Type", "application/json");
    restAdapter.setRequestURI(_ANOTHER_REPOSITORY_URI);
    restAdapter.setGenerateAnalyticsEvents(false);

    // make REST call to send events
    try
    {
      String responseMessage = restAdapter.send(_events.toString());
    }
    catch (IOException ex)
    {
      throw new AdfException(AdfException.ERROR, ResourceBundleHelper.CDC_ERROR_
BUNDLE,
                             CDCErrorBundle.ERR_ANALYTICS_FLUSH_EVENTS, new
Object[] { ex });
    }
  }

  private JSONArray _events = new JSONArray();
  private static final String _ANOTHER_REPOSITORY_URI = "_repository_uri";
}
```

You also need register the custom class in the logging.properties file. For example, to register CustomHandler, configure logging.properties as shown in the following example:

```
# Configure the analytics logger
oracle.maf.api.analytics.level=ALL
oracle.maf.api.analytics.handlers=oracle.maf.demo.CustomHandler
oracle.maf.api.analytics.custom.level=INFO
```

```
# Configure CustomHandler
oracle.maf.demo.CustomHandler.level=INFO
oracle.maf.demo.CustomHandler.connectionId=RepositoryConn
oracle.maf.demo.CustomHandler.batchSize=7
oracle.maf.demo.CustomHandler.offlineWrite=true
oracle.maf.demo.CustomHandler.recordUsername=false
oracle.maf.demo.CustomHandler.contextProviderClassName=oracle.maf.api.analytics.McsContextProvider
```

30.9.5 MAF Framework Events that Capture Analytics Information

MAF provides a range of MAF Framework events that capture analytics information.

These events are grouped into two categories (custom and system). Configure your application's logging.properties file so that the application sends these events to MCS. First, configure your application's `logging.properties` file to enable the sending of analytics information by, among other things, specifying the MCS `connectionID`. You then specify the events that you want to send. Use the information in the following tables for this latter task.

Table 30–4 lists the MAF Framework events (custom category) that you can configure in the `logging.properties` file to send analytics information to MCS from your application. You can disable each event by setting it to OFF or to a higher logging level than the level that enables logging of the event.

Table 30–4 MAF Framework Events (Custom Category)

Event Name	Logging Level	Enable or Disable Logging
`UpdateAuthenticationStatus`	FINE	`oracle.maf.api.analytics.custom.UpdateAuthenticationStatus.level= FINE` `oracle.maf.api.analytics.custom.UpdateAuthenticationStatus.level= OFF`
`FeatureNavigation`	INFO	`oracle.maf.api.analytics.custom.FeatureNavigation.level=INFO` `oracle.maf.api.analytics.custom.FeatureNavigation.level=OFF`
`Login`	INFO	`oracle.maf.api.analytics.custom.Login.level=INFO` `oracle.maf.api.analytics.custom.Login.level=OFF`
`LoginCallback`	FINER	`oracle.maf.api.analytics.custom.LoginCallback.level=FINER` `oracle.maf.api.analytics.custom.LoginCallback.level=OFF`

`Timer`	INFO for Operations: Warning and Expired
	FINE for Operation: Adjust
		`oracle.maf.api.analytics.custom.Timer.level=INFO`
		`oracle.maf.api.analytics.custom.Timer.level=FINE`
		`oracle.maf.api.analytics.custom.Timer.level=OFF`

Table 30–4 (Cont.) MAF Framework Events (Custom Category)

Event Name	Logging Level	Enable or Disable Logging
`Logout`	INFO	`oracle.maf.api.analytics.custom.Logout.level=INFO` `oracle.maf.api.analytics.custom.Logout.level=OFF`
`LogoutCallback`	FINER	`oracle.maf.api.analytics.custom.LogoutCallback.level=FINER` `oracle.maf.api.analytics.custom.LogoutCallback.level=OFF`
`PageNavigation`	INFO	`oracle.maf.api.analytics.custom.PageNavigation.level=INFO` `oracle.maf.api.analytics.custom.PageNavigation.level=OFF`
`FeatureTransition`	INFO	`oracle.maf.api.analytics.custom.FeatureTransition.level=INFO` `oracle.maf.api.analytics.custom.FeatureTransition.level=OFF`
`ApplicationTransition`	INFO	`oracle.maf.api.analytics.custom.ApplicationTransition.level=INFO` `oracle.maf.api.analytics.custom.ApplicationTransition.level=OFF`
`RESTWebService`	INFO	`oracle.maf.api.analytics.custom.RESTWebService.level=INFO` `oracle.maf.api.analytics.custom.RESTWebService.level=OFF`

Table 30–5 lists the MAF Framework events (system category) that you can configure in the logging.properties file to send analytics information to MCS from your application.

Table 30–5 MAF Framework Events (System Category)

Event Name	Logging Level	Description
`sessionStart`	INFO	MAF reserves the use of this event name to identify the session that it starts when a MAF application activates and starts to log analytics information. Do not define events in your MAF application that use this event name.
`sessionEnd`	INFO	MAF reserves the use of this event name to identify the session that it stops when a MAF application deactivates and stops logging analytics information. Do not define events in your MAF application that use this event name. MAF flushes events to MCS when sessionEnd occurs, irrespective of whether `batchSize` has been reached or no

Table 30–5 (Cont.) MAF Framework Events (System Category)

Event Name	Logging Level	Description
`context`	INFO	Sends context event information (for example, the timezone, carrier, and manufacturer of the device). Enable this event as follows: `oracle.maf.api.analytics.McsAnalyticsHandler .contextProviderClassName=oracle.maf.api .analytics.McsContextProvider` For more information about the context event information, see Section 30.9.3, "How to Send Context Events to Oracle Mobile Cloud Service."

A

Troubleshooting MAF Applications

This appendix describes problems with various aspects of mobile applications, as well as how to diagnose and resolve them.

This appendix includes the following sections:

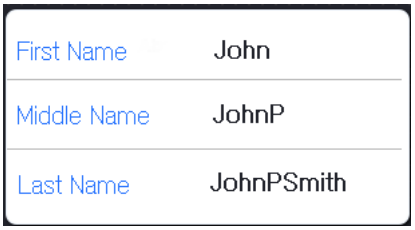
- Section A.1, "Problems with Input Components on iOS Simulators"
- Section A.2, "Code Signing Issues Prevent Deployment"

A.1 Problems with Input Components on iOS Simulators

Issue:

On mobile applications deployed to iOS simulators, text entered into one `<amx:inputText>` component field becomes attached to the beginning of the text entered in subsequent field when navigating from one field to another using a mouse. For example, on a page with First Name, Middle Name, and Last Name input text fields, if you enter *John* in the First Name field, then click the Middle Name field, and enter *P*, the text displays as *JohnP*. Likewise, when you click the Last Name field, and enter Smith, the text in that field displays as *JohnPSmith*, as shown in Figure A–1.

Figure A–1 Text Values Concatenate in Subsequent <amx:inputText> fields



Note: This behavior only occurs on iOS simulators and in web pages, not on actual devices.

Solution:

Use the keyboard on the simulator to traverse the input text fields rather than the mouse.

A.2 Code Signing Issues Prevent Deployment

Issue:

In some iOS development environments, mobile application deployment fails because of code signing errors.

Solution:

To ensure that the mobile application is signed, add code signing data to the Mach-O (Mach object) file by configuring the environment with `CODESIGN_ALLOCATE`. For example, enter the following from the Terminal:

```
export CODESIGN_
ALLOCATE="/Applications/Xcode.app/Contents/Developer/usr/bin/codesign_allocate"
```

For more information, *see codesign_allocate(1) OS X Manual Page* and *OS X ABI Mach-O File Format Reference*, both available from the iOS Developer Library (http://developer.apple.com/library/ios/navigation/).

B

Local HTML and Application Container APIs

This chapter describes the MAF JavaScript API extensions, the MAF Container Utilities API, and how to use the `AdfmfJavaUtilities` API for HTML application features, including custom HTML springboard applications.

This chapter includes the following sections:

- Section B.1, "Using MAF APIs to Create a Custom HTML Springboard Application Feature"
- Section B.2, "The MAF Container Utilities API"
- Section B.3, "Accessing Files Using the getDirectoryPathRoot Method"

B.1 Using MAF APIs to Create a Custom HTML Springboard Application Feature

Using JavaScript to call the JavaScript API extensions enables you to add the navigation functions to a custom springboard page authored in HTML. As stated in Section 4.5, "What You May Need to Know About Custom Springboard Application Features with HTML Content" you can enable callbacks and leverage Apache Cordova by including methods in the JavaScript `<script>` tag. The example below illustrates using this tag to call Cordova.

```
...
<script type="text/javascript">if (!window.adf) window.adf = {};
                                adf.wwwPath =
 "/~maf.device~/www/";</script>
 <script type="text/javascript" src="/~maf.device~/www/js/base.js"></script>
...
```

It is recommended that you use the virtual path `/~maf.device~/` when including `base.js` so that the browser will identify the request as being for a MAF resource and not for the remote server. This approach works in both remote as well as local HTML pages and is the best way to include `base.js` in an HTML feature (regardless of where it is being served from). For more information, see Section 20.2, "Enabling Remote Applications Access Container Services."

Tip: To access (and determine the location of) the `www/js` directory, you must first deploy a MAF application and traverse to the `deploy` directory. The `www/js` directory resides within the platform-specific artifacts generated by the deployment. For iOS deployments, the directory is located within the `temporary_xcode_project` directory. For Android deployments, this directory is located in the `assets` directory of the Android application package (`.apk`) file. See also Section 4.5, "What You May Need to Know About Custom Springboard Application Features with HTML Content."

Note: Because the path does not exist during design time, OEPE notes the JavaScript includes in the source editor as an error by highlighting it with a red, wavy underline. This path is resolved at runtime.

The MAF extension to the Cordova API enables the mobile device's API to access the configuration metadata in the `maf-feature.xml` and `maf-application.xml` files, which in turn results in communication between the mobile device and MAF's infrastructure. These extensions also direct the display behavior of the application features.

For information on the default MAF springboard page, `springboard.amx`, and about the ApplicationFeatures data control that you can use to build a customized springboard, see Section 4.6, "What You May Need to Know About Custom Springboard Application Features with MAF AMX Content."

B.1.1 About Executing Code in Custom HTML Pages

The example below illustrates a script defining the `showpagecomplete` event on the `handlePageShown` callback function. By listening to this event using standard DOM (Document Object Model) event listening, custom HTML pages (such as login pages) can invoke their own code after MAF has loaded and displayed the page for the first time.

```
<script>
        function handlePageShown()
        {
          console.log("Page is shown!");
        }
        document.addEventListener("showpagecomplete", handlePageShown, false);
</script>
```

Note: The `showpagecomplete` event guarantees the appropriate MAF state; other browser and third-party events, such as `load` and Cordova's `deviceready`, may not. Do not use them.

B.2 The MAF Container Utilities API

The methods of the MAF Container Utilities API provide MAF applications with such functionality as navigating to the navigation bar, displaying a springboard, or displaying application features. You can use these methods at the Java and JavaScript layers of MAF.

In Java, the Container Utilities API is implemented as static methods on the `AdfmfContainerUtilities` class, which is located in the `oracle.adfmf.framework.api` package. The example below illustrates calling the `gotoSpringboard` method. For more information on `oracle.adfmf.framework.api.AdfmfContainerUtilities`, see *Java API Reference for Oracle Mobile Application Framework*.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilities;
...
AdfmfContainerUtilities.gotoSpringboard();
...
```

B.2.1 Using the JavaScript Callbacks

The signatures of Java and JavaScript both match. In Java, they are synchronous and return results directly. Because JavaScript is asynchronous, there are two callback functions added for every function: a `success` callback that returns the results and a `failed` callback that returns any exception that is thrown. Within a Java method, the `success` value is returned from the function, or method, and the exception is thrown directly from the method. The pseudocode in the example below illustrates how a call with no arguments, `public static functionName() throws`, is executed within Java using `try` and `catch` blocks.

```
...
 try {
       result = AdfmfContainerUtilities.functionName();
    }
       catch() {
        ...
       }

...
```

Because JavaScript calls are asynchronous, the return is required through the callback mechanism when the execution of the function is complete. The pseudocode in the example below illustrates the signature of the JavaScript call.

```
adf.mf.api.functionName(
           function(successFunction, failureFunction ) { alert("functionName complete"); },
           function(successFunction, failureFunction ) { alert("functionName failed with " +
                                                          adf.mf.util.stringify(failureFunction); }
);
```

JavaScript calls cannot return a result because they are asynchronous. They instead require a callback mechanism when the execution of the function has completed. The signature for both the success and failed callbacks is `function(request, response)`, where the `request` argument is a JSON representation for the actual request and the `response` is the JSON representation of what was returned by the method (in the case of `success` callback functions) or, for `failed` callback functions, a JSON representation of the thrown exception.

> **Note:** The callback functions must be invoked before subsequent JavaScript calls can be made to avoid problems related to stack depth or race conditions.

The pseudocode in illustrates how a call with one or more arguments, such as `public static <return value> <function name>(<arg0>, <arg1>, ...) throws <exceptions>`, is executed within Java using a `try-catch` block.

```
try {
       result = AdfmfContainerUtilities.<function_name>(<arg0>, <arg1>, ...);
    }
       catch(<exception>) {
        ...
       }
```

JavaScript calls cannot return a result because they are asynchronous. They instead require a callback mechanism when the execution of the function has completed. The signature for both the success and failed callbacks is `function(request, response)`, where the `request` argument is a JSON representation for the actual request and the `response` is the JSON representation of what was returned by the method (in the case of `success` callback functions) or, for `failed` callback functions, a JSON representation of the thrown exception.

> **Note:** The callback functions must be invoked before subsequent JavaScript calls can be made to avoid problems related to stack depth or race conditions.

B.2.2 Using the Container Utilities API

The Container Utilities API provides the following methods:

- getApplicationInformation—Retrieves the metadata for the mobile application.
- gotoDefaultFeature—Displays the default application feature.
- gotoFeature—Displays a specific application feature.
- getFeatures—Retrieves the application features.
- getFeatureByName—Retrieves information about the application feature using the application feature's name.
- getFeatureById—Retrieves an application feature using its ID.
- resetFeature—Resets the application feature to the same state as when it was loaded.
- resetApplication—Resets the application.
- gotoSpringboard—Displays the springboard.
- showSpringboard—Shows the springboard.
- hideSpringboard—Hides the springboard
- showNavigationBar—Displays the navigation bar.
- hideNavigationBar—Hides the navigation bar.
- showPreferences—Displays the preferences page.
- invokeMethod—Invokes a Java method.
- invokeContainerMethod—Invokes a native method on the specified class with the given arguments.
- invokeContainerJavaScriptFunction—Invokes a JavaScript method.
- sendEmail—Displays the mobile device's email interface.

- sendSMS—Displays the mobile device's text messaging (SMS) interface.

The Container Utilities API also include methods for placing badges and badge numbers on applications. For more information, see Section B.2.22, "Application Icon Badging."

B.2.3 getApplicationInformation

This method returns an `ApplicationInformation` object that contains information about the application. This method returns such metadata as the application ID, application name, version, and the vendor of an application.

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.ApplicationInformation
              getApplicationInformation()
              throws oracle.adfmf.framework.exception.AdfException
```

The following example illustrates calling this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        ApplicationInformation ai =
AdfmfContainerUtilities.getApplicationInformation();
        String applicationId = ai.getId();
        String applicationName = ai.getName();
        String vendor = ai.getVendor();
        String version = ai.getVersion();
        ...
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getApplicationInformation(success, failed)
```

The `success` callback must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value, which is the `ApplicationInformation` object containing application-level metadata. This includes the application name, vendor, version, and application ID.

The `failed` callback must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The next example illustrates using these callback functions to retrieve the application information.

```
adf.mf.api.getApplicationInformation(
           function(req, res) { alert("getApplicationInformation complete"); },
           function(req, res) { alert("getApplicationInformation failed with " +
                                      adf.mf.util.stringify(res); }
                                    );
```

B.2.4 gotoDefaultFeature

This method requests that MAF display the default application feature. The default application feature is the one that is displayed when the mobile application is started.

> **Note:** This method may not be able to display an application feature if it has authentication- or authorization-related problems.

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoDefaultFeature(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the error.

This example illustrates using these callbacks to call the default application feature.

```
adf.mf.api.gotoDefaultFeature(
          function(req, res) { alert("gotoDefaultFeature complete"); },
          function(req, res) { alert("gotoDefaultFeature failed with " +
                                     adf.mf.util.stringify(res); }
                             );
```

B.2.5 gotoFeature

This method requests that MAF display the application feature identified by its ID.

> **Note:** This method may not be able to display an application feature if it has authentication- or authorization-related problems.

Within Java, this method is called as follows:

```
public static void gotoFeature(java.lang.String featureId)
                  throws oracle.adfmf.framework.exception.AdfException
```

This method's parameter, as shown in the example below, is the ID of the application feature.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        AdfmfContainerUtilities.gotoFeature("feature.id");
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoFeature(featureId, success, failed)
```

The `featureId` parameter is the application feature ID. This parameter activates the `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The next example illustrates using these callback functions to call an application feature.

```
adf.mf.api.gotoFeature("feature0",
          function(req, res) { alert("gotoFeature complete"); },
function(req, res) { alert("gotoFeature failed with " +
                           adf.mf.util.stringify(res); }
   );
```

B.2.6 getFeatures

This method returns an array of `FeatureInformation` objects that represent the available application features. The returned metadata includes the feature ID, the application feature name, and the file locations for the image files used for the application icons. This call enables a custom springboard implementation to access the list of application features that are available after constraints have been applied. (These application features would also display within the default springboard.)

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.FeatureInformation[] getFeatures()
              throws oracle.adfmf.framework.exception.AdfException
```

The next example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        FeatureInformation[] fia = null;
        fia = AdfmfContainerUtilities.getFeatures();

        for(int f = 0; f < fia.length; ++f) {
           FeatureInformation fi = fia[i];
           String featureId = fi.getId();
           String featureName = fi.getName();
           String featureIconPath = = fi.getIcon();
           String featureImagePath = fi.getImage();
           ...
        }
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned values and the exceptions to be passed back to the JavaScript calling code as follows:

```
public void getFeatures(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument

contains the associated `AdfmfContainerUtilities` method's return value (the array of `FeatureInformation` objects).

The `failed` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the error (`AdfException`).

```
adf.mf.api.getFeatures(
          function(req, res) { alert("getFeatures complete"); },
          function(req, res) { alert("getFeatures failed with " +
                                adf.mf.util.stringify(res); }
                        );
```

B.2.7 getFeatureByName

This method returns information about the application feature using the passed-in name of the application feature.

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.FeatureInformation getFeatureByName(java.lang.String
                                                                          featureName)
                                  throws oracle.adfmf.framework.exception.AdfException
```

This method's parameter, as shown in the example below, is the name of the application feature.

```
 ...
     try {
        FeatureInformation fi =
AdfmfContainerUtilities.getFeatureByName("feature.name");
        String featureId = fi.getId();
        String featureName = fi.getName();
        String featureIconPath = = fi.getIcon();
        String featureImagePath = fi.getImage();
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getFeatureByName(featureName, success, failed)
```

The `featureName` parameter is the name of the application feature. The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The next example illustrates using these callback functions.

```
adf.mf.api.getFeatureByName("feature.name",
          function(req, res) { alert("getFeatureByName complete"); },
          function(req, res) { alert("getFeatureByName failed with " +
                                     adf.mf.util.stringify(res); }
);
```

B.2.8 getFeatureById

This method retrieves an application feature using its application ID.

Within Java, this method is called as follows:

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;
```

This method's parameter, as shown in this example, is the ID of the application feature.

```
   try {
        FeatureInformation fi
=AdfmfContainerUtilities.getFeatureById("feature.id");
   }catch(AdfException e) {
        // handle the exception
   }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getFeatureById(featureId, success, failed)
```

The `featureId` parameter is the ID of the application feature. The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The example below illustrates using these callback functions to retrieve an application feature.

```
adf.mf.api.getFeatureById("feature.id",
          function(req, res) { alert("getFeatureById complete"); },
          function(req, res) { alert("getFeatureById failed with " +
                                     adf.mf.util.stringify(res); }
   );
```

B.2.9 resetFeature

This method resets the state of the application feature. It resets the Java-side model for the application feature and then restarts the user interface presentation as if the mobile application had just been loaded and displayed the application feature for the first time.

Within Java, this method is called as follows:

```
public static void resetFeature(java.lang.String featureId)
                  throws oracle.adfmf.framework.exception.AdfException
```

The method's parameter, as shown in the example below, is the ID of the application feature that is to be reset.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        AdfmfContainerUtilities.resetFeature("feature.id");
     }catch(AdfException e) {
        // handle the exception
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and exception to be passed back to the JavaScript calling code as follows:

```
public void resetFeature(featureId, success, failed)
```

The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (The ID of the application feature).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The example below illustrates using these callback functions to call an application feature.

```
adf.mf.api.resetFeature("feature0",
          function(req, res) { alert("resetFeature complete"); },
          function(req, res) { alert("resetFeature failed with " +
                                    adf.mf.util.stringify(res); }
                     );
```

B.2.10 resetApplication

This method resets the running application and it should be used only when resetting individual application features is not sufficient. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

Within Java, this method is called as follows:

```
public static void resetApplication(java.lang.String message)
```

The method's parameter, as shown in the following example, is either a message describing the reason for which the application is being restarted, or null if no message is required.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
   try {
      AdfmfContainerUtilities.resetApplication("New content is available");
   }
   catch(Exception e) {
      // handle the exception
}
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and exception to be passed back to the JavaScript calling code as follows:

```
public void resetApplication(message, success, failed)
```

The success callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (The ID of the application feature).

The failed callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The following example illustrates using these callback functions to call an application.

```
adf.mf.api.resetApplication("message1",
          function(req, res) { alert("resetApplication complete"); },
          function(req, res) { alert("resetApplication failed with " +
                                    adf.mf.util.stringify(res); }
```

```
);
```

B.2.11 gotoSpringboard

This method requests that MAF activate the springboard.

> **Note:** This method may not be able to display the springboard if it has not been designated as a feature reference in the `maf-application.xml` file, or if it has authentication or authorization-related problems. See also Section 4.2, "Configuring Application Navigation."

Within Java, this method is called as follows:

```
public static void gotoSpringboard()
```

The example below illustrates using this method

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

...
  try {
     AdfmfContainerUtilities.gotoSpringboard();
  }catch(AdfException e) {
        // handle the exception
  }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoSpringboard(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

This example illustrates using these callback functions.

```
adf.mf.api.gotoSpringboard(
    function(req, res) { alert("gotoSpringboard complete"); },
    function(req, res) { alert("gotoSpringboard failed with " +
                               adf.mf.util.stringify(res); }
);
```

B.2.12 showSpringboard

This method requests that MAF display the springboard.

Within Java, this method is called as follows:

```
public static void showSpringboard()
```

The following example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
   try {
```

```
      AdfmfContainerUtilities.showSpringboard();
   }
   catch(Exception e) {
      // handle the exception
   }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void showSpringboard(success, failed)
```

The success callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The following example illustrates using these callback functions.

```
adf.mf.api.showSpringboard(
           function(req, res) { alert("showSpringboard complete"); },
           function(req, res) { alert("showSpringboard failed with " +
                                      adf.mf.util.stringify(res); }
);
```

B.2.13 hideSpringboard

This method requests that MAF hide the springboard.

Within Java, this method is called as follows:

```
public static void hideSpringboard()
```

The following example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
   try {
      AdfmfContainerUtilities.hideSpringboard();
   }
   catch(Exception e) {
      // handle the exception
   }
```

In JavaScript, the success and failed callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void hideSpringboard(success, failed)
```

The success callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (void).

The failed callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The following example illustrates using these callback functions.

```
adf.mf.api.hideSpringboard(
       function(req, res) { alert("hideSpringboard complete"); },
       function(req, res) { alert("hideSpringboard failed with " +
                                  adf.mf.util.stringify(res); }
);
```

B.2.14 showNavigationBar

This method requests that MAF display the navigation bar.

Within Java, this method is called as follows:

```
public static void showNavigationBar()
```

This example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

...
  try {
     AdfmfContainerUtilities.showNavigationBar();
  }catch(Exception e) {
     // handle the exception
  }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void showNavigationBar(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The example below illustrates using these callback functions.

```
adf.mf.api.showNavigationBar(
           function(req, res) { alert("showNavigationBar complete"); },
           function(req, res) { alert("showNavigationBar failed with " +
                                      adf.mf.util.stringify(res); }
);
```

B.2.15 hideNavigationBar

This method requests that MAF hide the navigation bar.

Within Java, this method is called as follows:

```
public static void hideNavigationBar()
```

This example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
  try {
     AdfmfContainerUtilities.hideNavigationBar();
   }catch(Exception e) {
     // handle the exception
  }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void hideNavigationBar(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

This example illustrates using these callback functions.

```
adf.mf.api.hideNavigationBar(
       function(req, res) { alert("hideNavigationBar complete"); },
       function(req, res) { alert("hideNavigationBar failed with " +
                                  adf.mf.util.stringify(res); }
);
```

B.2.16 showPreferences

This method requests that MAF display the preferences page.

Within Java, this method is called as follows:

```
public static void showPreferences()
```

The following example illustrates using this method.

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
   try {
      AdfmfContainerUtilities.showPreferences();
   }
   catch(Exception e) {
      // handle the exception
   }
```

n JavaScript, the success and failed callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void showPreferences(success, failed)
```

The success callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (void).

The failed callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

The following example illustrates using these callback functions.

```
adf.mf.api.showPreferences(
          function(req, res) { alert("showPreferences complete"); },
          function(req, res) { alert("showPreferences failed with " +
                                     adf.mf.util.stringify(res); }
);
```

B.2.17 invokeMethod

This method is not available in Java. The example below illustrates using the JavaScript callback methods to invoke a Java method from any class in a classpath.

```
adf.mf.api.invokeMethod(classname,
                        methodname,
```

```
param1,
param2,
 ...
paramN,
successCallback,
failedCallback);
```

Table B–1 lists the parameters taken by this method.

Table B–1 Parameters Passed to invokeJavaMethod

Parameter	Description
`classname`	The class name (including the package information) that MAF uses to create an instance when calling the Java method.
`methodname`	The name of the method that should be invoked on the instance of the class specified by the `classname` parameter.

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value.

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Examples of using this method with multiple parameters are as follows:

```
adf.mf.api.invokeMethod("TestBean", "setStringProp", "foo", success, failed);

adf.mf.api.invokeMethod("TestBean", "getStringProp", success, failed);
```

An example of using an integer parameter is as follows:

```
adf.mf.api.invokeMethod("TestBean", "testSimpleIntMethod", "101", success, failed);
```

The following illustrates using complex parameters:

```
adf.mf.api.invokeMethod("TestBean", "testComplexMethod",
           {"foo":"newfoo","baz":"newbaz",".type":"TestBeanComplexSubType"}, success, failed);
```

The following illustrates using no parameters:

```
adf.mf.api.invokeMethod("TestBean", "getComplexColl", success, failed);
```

The following illustrates using `String` parameters:

```
adf.mf.api.invokeMethod("TestBean", "testMethodStringStringString", "Hello ",
                        "World", success, failed);
```

B.2.18 invokeContainerMethod

The invokeContainerMethod invokes a native method on the specified class with the given arguments. Table B-2 lists the parameters passed by this method.

Table B–2 Parameters Passed to invokeContainerMethod

Parameter	Description
`className`	The class name (including the package information) that MAF uses to create an instance.
`methodName`	The name of the method that should be invoked.

Table B–2 (Cont.) Parameters Passed to invokeContainerMethod

Parameter	Description
`args`	An array of arguments that are passed to the method. Within this array, these arguments should be arranged in the order expected by the method.

This method returns an `Object`.

```
public static java.lang.Object invokeContainerMethod(java.lang.String className,
                                                      java.lang.String methodName)
                                                      java.lang.Object[] args)
```

B.2.19 invokeContainerJavaScriptFunction

The `invokeContainerJavaScriptFunction` invokes a JavaScript method. Table B–3 lists the parameters passed by this method.

Table B–3 Parameters Passed to invokeContainerJavaScriptFunction

Parameter	Description
`featureId`	The ID of the application feature used by MAF to determine the context for the JavaScript invocation. The ID determines the web view in which this method is called.
`method`	The name of the method that should be invoked.
`args`	An array of arguments that are passed to the method. Within this array, these arguments should be arranged in the order expected by the method.

This method returns a JSON object.

> **Note:** The `invokeContainerJavaScriptFunction` API expects the JavaScript function to finish within 15 seconds for applications running on an Android-powered device or emulator, or it will return a timeout error.

```
public static java.lang.Object invokeContainerJavaScriptFunction(java.lang.String featureId,
                                                                  java.lang.Object[] args)
                                throws oracle.adfmf.framework.exception.AdfException
```

The pseudocode in the next example illustrates a JavaScript file called `appFunctions.js` that is included in the application feature, called `feature1`. The JavaScript method, `application.testFunction`, which is described within this file, is called by the `invokeContainerJavaScriptFunction method`, shown in the second example below. Because the application includes a command button that is configured with an action listener that calls this function, a user sees the following alerts after clicking this button:

- APP ALERT 0
- APP ALERT 1
- APP ALERT 2

```
(function()
    {
       if (!window.application) window.application = {};
```

```
      application.testFunction = function()
      {
         var args = arguments;

         alert("APP ALERT " + args.length + " ");
         return "application.testFunction - passed";
      };
   })();
```

The pseudocode in the next example illustrates how the `invokeApplicationJavaScriptFunction` method calls the JavaScript method (`application.testFunction`) that is described in the previous example.

```
invokeApplicationJavaScriptFuntions
    public void invokeApplicationJavaScriptFuntions(ActionEvent actionEvent) {
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",

"application.testFunction",
                                                                  new Object[] {}
);
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",

"application.testFunction",
                                                                  new Object[]
{"P1"} );
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",

"application.testFunction",
                                                                  new Object[]
{"P1", "P2"} );
    }
```

For more information, see *Java API Reference for Oracle Mobile Application Framework* and the APIDemo sample application. This sample application is available from **File** > **New** > **MAF Examples**.

B.2.20 sendEmail

For information, see Section 13.10.3, "How to Use the sendEmail Method to Enable Email."

B.2.21 sendSMS

For information, see Section 13.10.2, "How to Use the SendSMS Method to Enable Text Messaging."

B.2.22 Application Icon Badging

The `AdfmfContainerUtilities` class includes methods to place or retrieve a badge number on a mobile application icon. Table B–4 describes these methods.

Table B–4 Icon Badging Methods

Method	Description	Parameters
`getApplicationIconBadgeNumber`	Gets the current badge value on the mobile application icon. Returns zero (0) if the application icon is not badged.	None

Table B–4 (Cont.) Icon Badging Methods

Method	Description	Parameters
`setApplicationIconBadgeNumber`	Sets the badge number on a mobile application icon.	The value of the badge (`int badge`).

Note: Application icon badging is not supported on Android.

B.3 Accessing Files Using the getDirectoryPathRoot Method

The `AdfmfJavaUtilties` API includes the `getDirectoryPathRoot` method. This method, which can only be called from the Java layer, enables access to files on the iOS, Android, and Windows systems. As shown in the following example, this method enables access to the location of the temporary files, application files (on iOS systems), and the cache directory on the device using the `TemporaryDirectory`, `ApplicationDirectory`, and `DeviceOnlyDirectory` constants, respectively. Files stored in the `DeviceOnlyDirectory` location are not synchronized when the device is connected.

Note: Verify that any directories or files accessed by an application exist before the application attempts to access them.

For more information on `oracle.adfmf.framework.api.AdfmfJavaUtilities`, see *Java API Reference for Oracle Mobile Application Framework*.

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;

...

public void getDirectoryPathRoot()
   {
 // returns the directory for storing temporary files

     String tempDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.TemporaryDirectory);

 // returns the directory for storing application files

     String appDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.ApplicationDirectory);

 // returns the directory for storing cache files

     String deviceDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DeviceOnlyDirectory);

 // returns the directory for storing downloaded files

AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DownloadDirectory);
   }
```

B.3.1 Accessing Platform-Independent Download Locations

File storage requirements differ by platform. The Android platform does not prescribe a central location from which applications can access files; instead, an application can

write a file to any location to which it has write permission. iOS platforms, on the other hand, generally store files within an application directory. In Windows, application do not access external files or directories: files are stored within the application package. Because of these differences, passing `ApplicationDirectory` to the `getDirectoryPathRoot` method can return the file location needed to display attachments for applications running on iOS-powered or Windows-powered devices, but not on Android-powered devices. Rather than writing platform-specific code to retrieve these locations for applications intended to run on both iOS- and Android-powered devices, you can enable the `getDirectoryPathRoot` method to return the paths to both the external storage location and the default attachments directory by passing it `DownloadDirectory`. This constant (an `enum` type) reflects the locations used by the `displayFile` method of the `DeviceManager` API, which displays attachments by using platform-specific functionality to locate these locations.

On Android, `DownloadDirectory` refers to the path returned by the `Environment.getExternalStorageDirectory` method (which retrieves the external Android storage directory, such as an SD card). For mobile applications running on iOS-powered devices, it returns the same location as `ApplicationDirectory`. For more information on the `getExternalStorageDirectory`, see the package reference documentation, available from the Android Developers website (`http://developer.android.com/reference/packages.html`). See also *Files System Programming Guide*, available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

C

MAF Application and Project Files

This appendix provides a reference for the files that OEPE generates when you create a MAF application using the Mobile Application Framework Application template.

This appendix includes the following section:

- Section C.1, "Introduction to MAF Application and Project Files"
- Section C.2, "About the Assembly-Level Resources"
- Section C.3, "About the Application Project Resources"
- Section C.4, "About the View Project Resources"

C.1 Introduction to MAF Application and Project Files

By default. OEPE creates a MAF application with three projects:

- The top-level or assembly project. This holds all of the artifacts required for packaging and deployment of the application. In addition, the assembly project tracks the version of the MAF run time that the project uses, if you are migrating an application from an earlier version of MAF.
- The application project, *applicationApplication*
- The view controller-type project, *applicationView*. This contains the source folder (`src`) and the `ViewContent` folder.

The data controls are:

- The Device Features data control. The Apache Cordova Java API is abstracted through this data control, thus enabling the application features implemented as MAF AMX to access various services embedded on the device.
- The Application Features data control, which enables you to build a springboard page.

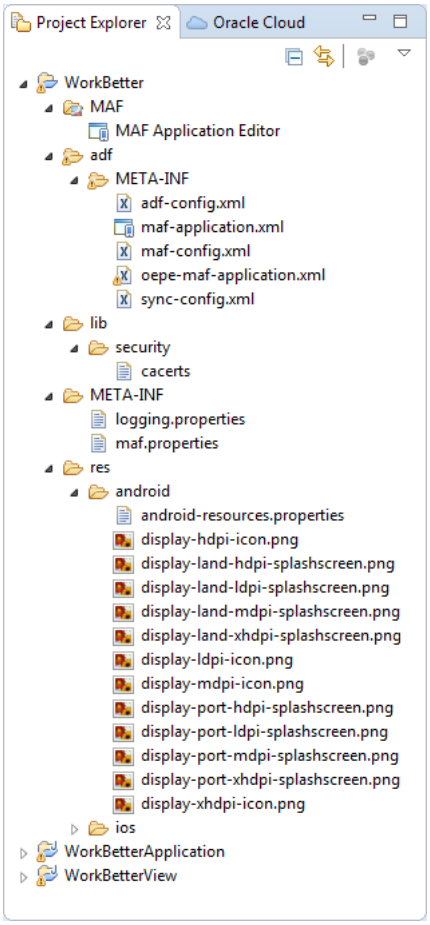
For more information about these projects, see Section C.2, "About the Assembly-Level Resources", Section C.3, "About the Application Project Resources", and Section C.4, "About the View Project Resources."

OEPE also generates files within these projects that you use to configure your MAF application and application features or files that your MAF application needs when you deploy it to the targeted platform. In almost every case, you do not work with the generated files directly. Instead you use the editors, such as the MAF Application Editor, the MAF Features Editor and the Data Control Manager.

C.2 About the Assembly-Level Resources

OEPE generates the files for the MAF application in the assembly project. These files, described in Table C–1, contain configuration files for describing the metadata of the MAF application. You access these files from the Project Explorer pane shown in Figure C–1.

Figure C–1 Mobile Application Artifacts



The assembly project (generated with the default name, *application*), which contains the application-wide resources, provides the presentation layer of the MAF application in that it includes metadata files for configuring how the application will display on a mobile device. This project dictates the security for the MAF application and can include the application's login page, an application-wide resource. The application controller project is essentially a consumer of the view controller project, which defines the application features and their content.

You edit application-level artifacts using the MAF Application Editor which edits `maf-application.xml`. It is used for configuring the MAF application itself, such as its name, the application lifecycle listener (`LifeCycleListenerImpl.java`), the login server connections for the embedded application features.

Table C–1 Mobile Application-Level Artifacts Accessed Through Assembly Project

Artifact(s)	File Location	Description
`maf-application.xml`	*assembly project directory*`\adf\Meta-INF` For example: `workspace\application name\adf\META-INF\maf-application.xml`	A stub XML application descriptor file that enables you to define the MAF application. Similar to the application descriptors for ADF Fusion Web applications, this enables you to define the content for an application, its navigation behavior, and its user authentication requirements. You edit this file using the MAF Application Editor.
`maf-config.xml`	*assembly project directory*`\adf\Meta-INF` For example: `workspace\application name\adf\META-INF\maf-config.xml`	Use to configure the default skin used for MAF applications.
Application images	*assembly project directory*`\res\android` *assembly project directory*`\res\ios` For example: `workspace\application name\res\ios\Default.png`	A set of images required for the deployment of iOS and Android applications. These include PNG images for application icons and splash screens. Deployment to an iOS-powered device, such as an iPhone, requires a set of images in varying sizes. The default iOS images provided with the project include: ▪ images used when the device is in both landscape and portrait orientations ▪ images used for retina displays (that is, `icon.png` and `icon@2x.png`) ▪ an iPad image (`icon-72.png`) To override these images, see Section 27.3.1.5, "How to Add a Custom Image to an Android Application."
`cacerts`	*assembly project directory*`\lib\security` For example: `workspace\application name\security\cacerts`	The `cacerts` certificate file, a system-wide keystore that identifies the CA certificates to JVM 1.4. You can update this file using the Java keytool utility. You can create a custom certificate file using keytool as described in Section 29.11, "Configuring a MAF Application to Enable Two-Way SSL for Authentication."Any certificate file must reside within the `Security` directory.
`logging.properties`	*assembly project directory*`\META-INF` For example: `workspace\application name\META-INF\logging.properties`	Enables you to set the application error logging, such as the logging level and logging console. For more information, see Section 30.6, "Using and Configuring Logging in MAF Applications."

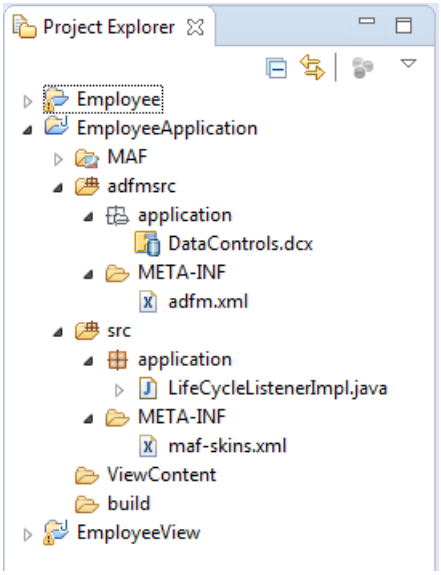
Table C–1 (Cont.) Mobile Application-Level Artifacts Accessed Through Assembly Project

Artifact(s)	File Location	Description
`maf.properties`	*assembly project directory*`META-INF` For example: `workspace\application name\META-INF\maf.properties`	The configuration file for the Java virtual machine, JVM 1.4. Use this file to configure the application startup and heap space allotment, as well as Java and JavaScript debugging options. For more information, see Section 30.3.2, "How to Enable Debugging of Java Code and JavaScript."
`adf-config.xml`	*assembly project directory*`\adf\META-INF` For example: `workspace\application\adf\META-INF\adf-config.xml`	Used to configure application-level settings, including the Configuration Service parameters. See also Chapter 14, "Configuring End Points Used in MAF Applications."
`connections.xml`	*assembly project directory*`\adf\META-INF` For example: `workspace\application\adf\META-INF\connections.xml`	The repository for all of the connections defined in the MAF application.
`sync-config.xml`	*assembly project directory*`\.adf\META-INF` For example: `workspace\application\META-INF\sync-config.xml`	The configuration file for the offline cache of data returned from REST web services. Caching not only improves the user experience by boosting performance, but allows users to read and view data when they are working in an off-line mode.

C.3 About the Application Project Resources

Within the application project itself (which is generated with the default name, *application*`Application`) OEPE creates the following artifacts, listed in Table C–2.

Figure C–2 Application Project



You edit application-level artifacts using the MAF Features Editor which edits `maf-feature.xml`. It describes which application features comprise the MAF application.

Table C–2 Application Project Artifacts

Artifact(s)	File Location	Description
`LifeCycleListenerImpl.java`	*application project directory*`\src\application` For example: `workspace\application project\src\application\LifeCycleListenerImpl.java`	The default application lifecycle listener (ALCL) for the MAF application.
`maf-skins.xml`	*application project directory*`\src\META-INF` For example: `workspace\application project\src\META-INF\maf-skins.xml`	Defines the available skins and also enables you to define new skins. For more information, see Chapter 7, "Skinning MAF Applications."
`adfm.xml`	*application project directory* `\adfmsrc\META-INF` For example: `workspace\application project\adfmsrc\META-INF\adfm.xml`	Maintains the paths (and relative paths) for the `.cpx`, `.dcx`, `.jpx`, and `.xcfg` files (registries of metadata).
`DataControls.dcx`	*application project directory*`\adfmsrc\application` For example: `workspace\application project\adfmsrc\application\DataControls.dcx`	The data controls registry. For information on using the DeviceFeatures data control, which leverages the services of the device, see Chapter 13, "Using Bindings and Creating Data Controls in MAF AMX." For information on the ApplicationFeatures data control, which enables you to create a springboard page that calls the embedded application features, see Section 4.6, "What You May Need to Know About Custom Springboard Application Features with MAF AMX Content."

C.4 About the View Project Resources

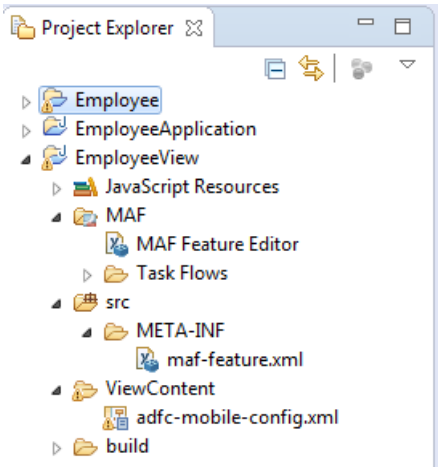
The view project (which is generated with the default name, *applicationView*), as illustrated in Figure C–3) houses the resources for the application features. Unlike the application project described in Section C.2, "About the Assembly-Level Resources," the view project's metadata files describe the resources at the application feature-level, in particular the various application features that can be aggregated into a MAF application so that they can display on a mobile device within the springboard of the MAF application itself or its navigation bar at runtime. Further, the application feature metadata files describe whether the application feature is comprised of HTML or MAF AMX pages. In addition, the view controller project can include these application pages as well as application feature-level resources, such as icon images to represent the application feature on the springboard and navigation bar defined for the MAF application.

Tip: Store code specific to an application feature within the view controller project. Use the application controller project as the location for code shared across application features, particularly those defined in separate view controller projects.

The view controller project can be decoupled from the application controller project and deployed as an archive file for reuse in other mobile applications as described in Section 8.1, "Introduction to Feature Archive Files,"." In rare cases, an application controller project can consume more than one view controller project.

Note: Adding a MAF view controller project as a dependency of another MAF view controller project, or as a dependency of a MAF application controller project, prevents the deployment of a MAF application.

Figure C–3 View Project



These resources include the configuration file for application features called `maf-feature.xml`, which you edit using the MAF Feature Editor.

Table C–3 View Controller Artifacts

Artifact(s)	File Location	Description
`maf-feature.xml`	*view project directory*`\src\META_INF` For example: `workspace\application View\src\META-INF`	A stub XML descriptor file that enables you to define application features. After you have configured the Mobile Preferences as described in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse* you can deploy this application using the default deployment configuration settings. For more information, see Chapter 27, "Deploying MAF Applications."
Application-Specific Content	*view project directory*`\public_html` For example: `workspace\application View\public_html`	The application features defined in `maf-feature.xml` display in the `public_html` directory. Mobile content can include MAF AMX pages, CSS files, and task flows. Any custom images that you add to an application feature must be located within this directory. For more information, see Section 5.4, "What You May Need to Know About Selecting External Resources."

D

Converting Preferences for Deployment

This appendix describes how MAF converts user preferences during deployment.

This document includes the following sections:

- Section D.1, "Naming Patterns for Preferences"
- Section D.2, "Converting Preferences for Android"
- Section D.3, "Converting Preferences for iOS"
- Section D.4, "Converting Preferences for Windows"

D.1 Naming Patterns for Preferences

Conversion of MAF application preferences to a mobile-platform representation occurs when a deployment target is invoked. Following conversion, the naming pattern described in Table D–1 ensures that each preference can be uniquely identified on the mobile platform. Each preference element in the `maf-application.xml` and `maf-feature.xml` files must be uniquely identified within the scope of its sibling elements prior to deployment.

The following are examples of identifier values:

- `application.gen.gps.trackGPS`
- `feature.f0.gen.gps.trackGPS`

Table D–1 describes how to generate fully qualified preference identifiers.

Table D–1 MAF Naming Patterns for Preferences

Expression	Description	Syntax
`PreferenceIdentifier`	Represents an identifier value of a preference element that has been converted to a mobile platform representation.	`ApplicationPreferences` | `FeaturePreferences`

`ApplicationPreferences`	Use this expression to build a preference identifier value that is generated from the `maf-application.xml` file.	*application*`.ApplicationElementPath`
		`ApplicationElementPath` represents a dot-separated list of `id` attribute values beginning with the top-most parent element, `<adfmf:preferences>`, and ending with the element that is to be identified. In the following segment from the `maf-application.xml` file, this generated identifier is shown in the comment as `application.gen.gps.trackGPS`.
		`<adfmf:preferences>`
		`  <adfmf:preferenceGroup id="gen">`
		`  <adfmf:preferenceGroup id="gps">`
		`<!-- The mobile-platform identifier would be "application.gen.gps.trackGPS" -->`
		`  <adfmf:preferenceBoolean id="trackGPS"/>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferenceGroup>`
		`</adfmf:preferences>`

`FeaturePreferences`	Use this expression to build a preference identifier value that is generated from the `maf-feature.xml` file.	*feature*`.FeatureElementPath`
		`FeatureElementPath` represents a dot-separated list of `id` attribute values beginning with `<adfmf:feature>`, the top-most parent element, and ending with the element that is to be identified. In the following segment from the `maf-feature.xml` file, this generated identifier is displayed in the comment as `feature.f0.gen.gps.trackGPS`.
		`<adfmf:feature id="f0">`
		`  <adfmf:preferences>`
		`  <adfmf:preferenceGroup id="gen">`
		`  <adfmf:preferenceGroup id="gps">`
		`<!-- The mobile-platform identifier would be "feature.f0.gen.gps.trackGPS" -->`
		`  <adfmf:preferenceBoolean id="trackGPS"/>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferences>`
		`</adfmf:feature>`

The `<adfmf:preferences>` element cited in the code examples in Table D–1 does not have an `id` attribute and is therefore not represented in any preference identifiers.

D.2 Converting Preferences for Android

The MAF deployment uses XML and XLS to transform the user preference pages defined at both the application feature and application-level into the following three XML documents:

- preferences.xml
- arrays.xml
- strings.xml

D.2.1 maf_preferences.xml

This file contains the transformed preferences from both of the maf-feature.xml and maf-application.xml files.

D.2.1.1 Preferences Element Mapping

Table D–2 shows the mapping of MAF's preference definitions to Android template preferences, and Android native preferences:

Table D–2 Mapping MAF Preferences to Android Preferences

MAF Preference Definition	Custom or Android Native Preference Definition (Used by MAF Deployment)	Android Native Preference Definition (Not used by MAF Deployment)
<adfmf:preferenceBoolean>	oracle.adfmf.preferences.AdfMFPreferenceBoolean	CheckBoxPreference
<adfmf:preferenceNumber>	oracle.adfmf.preferences.AdfMFPreferenceText	EditPreferenceText
<adfmf:preferenceText>	oracle.adfmf.preferences.AdfMFPreferenceText	EditTextPreference
<adfdmf:preferenceList>	oracle.adfmf.preferences.AdfMFPreferenceList	ListPreference
<adfmf:PreferenceGroup>	PreferenceCategory	PreferenceCategory
<adfmf:PreferencePage>	PreferenceScreen	PreferenceScreen

D.2.1.2 Preference Attribute Mapping

The Preferences.xml file contains references to string resources contained in both the strings.xml and arrays.xml files. The Android SDK defines the syntax for resources in XML files as *@[<package_name>:]<resource_type>/<resource_name>*. This file contains references to string values as well as the name and value pairs of list preferences. The XSL constructs the following for the strings and list preferences:

- <package_name> is the name of the package in which the resource is located (not required when referencing resources from the same package). This component of the reference will not be used.
- <resource_type> is the R subclass for the resource type. This component will have a value of string if constructing a string reference or array if constructing a list preference.
- <resource_name> is the android:name attribute value in the XML element. The value for this component will be the value of the <PreferenceIdentifier>_title when specifying the android:title attribute (see Section D.1, "Naming Patterns for Preferences." for the definition of <PreferenceIdentifier>.

Table D–3 and Table D–4 show the mapping of MAF attributes for a given MAF preference to the Android preference.

In this table:

- Entries of the form {*X*} (such as {default} in Table D–3) indicate the value of a MAF attribute named *X*.

- Entries having <PreferenceIdentifier> indicate the value of the preference identifier, as defined in Section D.1, "Naming Patterns for Preferences."
- Attributes with an asterisk (*) are custom template attributes defined in a MAF namespace and must appear in the preferences.xml in the form adfmf:<attributeName>. Otherwise, the attributes are part of the Android namespace and must appear in the preferences.xml as android:<attributeName>.

Table D–3 Mapping of MAF Preference Attributes to Android Preferences

MAF Attribute Definition	Template Custom or Android Native Preference Attribute	Android Attribute Value	Applies to
id	key	<PreferenceIdentifier>	AdfMFPreferenceBoolean, AdfMFPreferenceText, AdfMFPreferenceList, PreferenceScreen, PreferenceCategory
default	defaultValue	{default}	AdfMFPreferenceBoolean, AdfMFPreferenceText, AdfMFPreferenceList
label	title	@string/<PreferenceIdentifier>___ title if the given {label} value is not a reference to a string resource bundle. References a string in strings.xml having the given {label}.	AdfMFPreferenceBooleanAdfM, FPreferenceNumber, AdfMFPreferenceText, AdfMFPreferenceList, PreferenceScreen, PreferenceCategory
secret	password	{secret}	AdfMFPreferenceText
min	min*	{min}	AdfMFPreferenceText
max	max*	{max}	AdfMFPreferenceText
name	entryValues	@array/<PreferenceIdentifier>___ entryValues	AdfMFPreferenceList
value	entries	@array/<PreferenceIdentifier>___ entries	AdfMFPreferenceList

D.2.1.3 Attribute Default Values

The MAF Application Editor and MAF Features Editor exclude an attribute name and value from the XML if:

- The attribute type is xsd:boolean.
- The attribute value has a <default> value option.
- The user specifies <default> as the value.

The XSL must know the MAF attributes that are boolean typed and their corresponding default values. The XSL, then, specifies the appropriate Android or template custom attribute value where has been selected by the user.

Table D–4 indicates what the deployment will specify for the android:defaultValue attribute if the MAF preference being transformed does not contain a default attribute:

Table D–4 Transforming Attributes with Non-Default Values

MAF Preference Element	Android Preference Equivalent	Default Attribute Value
preferenceBoolean	AdfMFPreferenceBoolean	false
preferenceText	AdfMFPreferenceText	Empty string
preferenceList	AdfMFPreferenceList	Empty string

D.2.1.4 Preferences Screen Root Element

The preferences.xml file has a root element called <PreferenceScreen>. The Android template requires that this element have the following XML namespace definition:

```
xmlns:adfmf="http://schemas.android.com/apk/res/<Application Package Name>
```

The <Application Package Name> element is defined as the same application package name in the AndroidManifest.xml file. <Android Package Name> defines the definition for the Android package name specified in the AndroidManifest.xml file. For more information, see Section 3.2, "Setting Display Properties for a MAF Application."

The deployment uses the Package Name value from the Android deployment profile if it exists. If it does not exist in the profile, the deployment obtains this value from the application display name and Application Id contained in the maf-application.xml file. The deployment Java code will pass the value to the XSL document as a parameter.

The example below shows MAF preferences contained in the maf-feature.xml file for the sample application PrefDemo, described in Appendix E, "MAF Sample Applications.".

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
                xmlns:adfmf="http://xmlns.oracle.com/jdev/adfmf">
   <adfmf:feature id="oracle.hello"
                  name="Hello"
                  icon="oracle.hello/navbar-icon.png"
                  image="oracle.hello/springboard-icon.png">
      <adfmf:content id="Hello.Generic">
         <adfmf:localHTML url="oracle.hello/index.html"/>
      </adfmf:content>
      <adfmf:preferences>
         <adfmf:preferenceGroup id="prefGroup"
                                label="preference group">
            <adfmf:preferenceBoolean id="boolPref"
                                     label="boolPref perference"
                                     default="true"/>
            <adfmf:preferenceNumber id="numPref"
                                    label="numPref preference"
                                    default="1"
                                    min="1"
                                    max="10"/>
            <adfmf:preferenceText id="textPref"
                                  label="textPref preferences"
                                  default="Foo"/>
            <adfmf:preferenceList id="listPref"
                                  label="listPref preference"
                                  default="value2">
            <adfmf:preferenceValue name="name1"
```

```
                                    value="value1"/>
            <adfmf:preferenceValue name="name2"
                                    value="value2"/>
            </adfmf:preferenceList>
         </adfmf:preferenceGroup>
      </adfmf:preferences>
   </adfmf:feature>
</adfmf:features>
```

D.2.2 arrays.xml

The `arrays.xml` file consists of string-array elements that enumerate the names and values of list preferences that are referenced from the `preferences.xml` file. Each `<preferenceList>` element contained in the `maf-application.xml` and `maf-feature.xml` files is transformed into two string-array elements, one element for the name and one element for the values. For example, the MAF `preferenceList` definition described in the example below results in `<string-array name="feature.oracle.hello.prefGroup.MyList___entry_values">` and `<string-array name="feature.oracle.hello.prefGroup.MyList___entries">` in the `arrays.xml` file shown in the second example below.

```
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/adf/mf">

<adfmf:feature id="oracle.hello" name="Hello" icon="oracle.hello/navbar-icon.png"
image="oracle.hello/springboard-icon.png">
...
   <adfmf:preferences>
      <adfmf:preferenceGroup id="prefGroup">
         <adfmf:preferenceList id="MyList" label="My List">
            <adfmf:preferenceValue name="name1" value="value1"/>
            <adfmf:preferenceValue name="name2" value="value2"/>
            <adfmf:preferenceValue name="name3" value="value3"/>
         </adfmf:preferenceList>
      </adfmf:preferenceGroup>
   </adfmf:preferences>
</adfmf:feature>
...
```

The example below illustrates the pair of string array elements in the `arrays.xml` file that are transformed from a `<preferenceList>` element.

```
<resources xmlns:android="http://schemas.android.com/apk/res/android"
           xmlns:adfmf="http://schemas.android.com/apk/res/oracle.myandroidapp">

   <string-array name="feature_oracle_hello_prefGroup.MyList___entry_values">
      <item>name1</item>
      <item>name2</item>
      <item>name3</item>
   </string-array>

   <string-array name="feature_oracle_hello_prefGroup.MyList___entries">
      <item>value1</item>
      <item>value2</item>
      <item>value3</item>
   </string-array>
</resources>
```

The example below shows the `<string-arrays>` referenced in `preferences.xml`.

```
<oracle.adfmf.preferences.AdfMFPreferenceList
android:key="feature.oracle.hello.MyList"
android:title="@string/feature_oracle_hello_prefGroup.MyList___title"
android:entries="@array/feature_oracle_hello_prefGroup.MyList___entries"
android:entryValues="@array/feature_oracle_hello_prefGroup.MyList___entry_values"
/>
```

D.2.3 Strings.xml

The `strings.xml` file, shown in the example below, consists of string elements that are referenced by the `preferences.xml` file, as well as any resource bundle references defined in the `maf-application.xml` and `maf-feature.xml` files. Each string element has a `name` attribute that uniquely identifies the string and the string value.

```
<resources xmlns:android="http://schemas.android.com/apk/res/android"
           xmlns:adfmf="http://schemas.android.com/apk/res/oracle.myandroidapp">
...
  <string name="feature.PROD.bundle.FeatureName">Products</string>
  <string name="feature.oracle.hello.prefGroup.MyBooleanPreference___title">My
feature boolean pref</string>
...
</resources>
```

If the source of the string is not a reference to a resource bundle string, the naming convention for the `name` attribute is `<PreferenceIdentifier>___ <androidAttributeName>`.

```
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
   <adfmf:loadBundle basename="mobile.ViewControllerBundle"
                     var="bundle"/>
   <adfmf:feature id="oracle.hello"
                  name="Hello"
                  icon="oracle.hello/navbar-icon.png"
                  image="oracle.hello/springboard-icon.png">
   <adfmf:feature id="PROD"
                  name="#{bundle.FeatureName}"
                  icon="openMore.png"
                  image="G.png"
                  credentials="none">
...
      <adfmf:preferences>
         <adfmf:preferenceGroup id="prefGroup">
            <adfmf:preferenceBoolean default="true"
                                     id="MyBooleanPreference"
                                     label="My feature boolean pref"/>
         </adfmf:preferenceGroup>
      </adfmf:preferences>
   </adfmf:feature>
```

D.3 Converting Preferences for iOS

The MAF deployment transforms the MAF preferences listed in Table D–4 to the preference list (`strict the Display to a `) file representation required by an iOS Settings application.

Table D–5 MAF Preferences and Their iOS Counterparts

MAF Preferences Component	iOS Representation
<adfmf:preferencePage>	PSChildPaneSpecifier
<adfmf:preferenceGroup>	PSGroupSpecifier
<adfmf:preferenceBoolean>	PSToggleSwitchSpecifier
<adfmf:preferenceList>	PSMultiValueSpecifier
<adfmf:preferenceText>	PSTextFieldSpecifier
<adfmf:preferenceNumber>	PSTextFieldSpecifier

For information on the iOS requirement for preference list (`.plist`) files, see *Preferences and Settings Programming Guide*, which is available through the iOS Developer Library (http://developer.apple.com/library/ios/navigation/).

This example shows XML based on the `maf-application.xml` file.

```
<adfmf:preferences>
   <adfmf:preferenceGroup id="gen"
                          label="Oracle Way Cool Mobile App">
      <adfmf:preferenceGroup id="SubPage01"
                             label="Child Page">
   </adfmf:preferenceGroup>
</adfmf:preferences>
```

D.4 Converting Preferences for Windows

Provides information on preferences for the Windows platform

For the Windows platform, from the `maf-feature.xml` and `maf-application.xml` metadata, MAF generates a single `maf-preferences.json` file and multiple `maf-preferences.resjson` files, one each for every supported locale in the application. MAF uses the JSON file, saves it as Windows settings for the application, and also uses it to display the settings screen.

E

MAF Sample Applications

This appendix describes the MAF sample applications.

This appendix includes the following section:

- Section E.1, "Overview of the MAF Sample Applications"

E.1 Overview of the MAF Sample Applications

Mobile Application Framework ships with a set of a sample applications that provide different development scenarios, such as creating the basic artifacts, accessing such device-native features as SMS and e-mail, or performing CRUD (Create, Read, Update, and Delete) operations on a local SQLite database. The sample applications are available from **File** > **New** > **MAF Examples**.

These applications, which are described in Table E–1, are complete. Except where noted otherwise, these applications can be deployed to a simulator after you configure the development environment as described in *Oracle Enterprise Pack Installing Oracle Enterprise Pack for Eclipse*.

> **Tip:** To get an idea of how to create a mobile application, review these applications in the order set forth in Table E–1.

Table E–1 MAF Sample Applications

Application Name	Description	Additional Resources Required to Run the Sample Application
HelloWorld	The "hello world" application for MAF, which demonstrates the basic structure of the framework. This basic application has a single application feature that is implemented with a local HTML file. Use this application to ascertain that the development environment is set up correctly to compile and deploy an application. See also Section 2.3.2, "What Happens When You Create an MAF Application."	
APIDemo	This application demonstrates how to invoke custom JavaScript methods from within a MAF AMX page. Use this approach to invoke the Apache Cordova APIs that are not included in the DeviceFeatures data control. You can also use this approach to add custom JavaScript methods to an application and invoke them as well. This application also demonstrates calling back to Java from the JavaScript methods. For more information, see Section B.2, "The MAF Container Utilities API."	
BarcodeDemo	This application demonstrates how to make use of a Cordova plugin by calling the BarcodeScanner plugin from embedded JavaScript that is invoked from a backing bean. For more information, see Chapter 9, "Using Plugins in MAF Applications."	
BeaconDemo	This application demonstrates how to make use of a Cordova plugin to detect iBeacons.	
CompGallery	This application serves as an introduction to the MAF AMX UI components by demonstrating all of these components. Using this application, you can change the attributes of these components and see the effects of those changes in real time without recompiling and redeploying the application after each change. See generally Chapter 12, "Creating the MAF AMX User Interface."	
ConfigServiceDemo	This application demonstrates the use of the Configuration Service to change the end points used in a MAF application. Changes to end points in `connections.xml` are propagated to the application on the device and the application re-initialized to consume the new end points. For more information, see Section 14.1, "Introduction to Configuring End Points."	
CRUDDemo	This application demonstrates how to build CRUD operations using the SQLite database and Java bean data controls. This application displays a list of employees and allows you to create, update, and delete employees. This application uses a local SQLite database to store its data, persisting the data during CRUD operations. In addition, this application shows how to use CREATE and DELETE operations to add or delete items to and from a collection. For more information, see Chapter 19, "Using the Local Database in MAF AMX."	

Table E–1 (Cont.) MAF Sample Applications

Application Name	Description	Additional Resources Required to Run the Sample Application
DeviceDemo	This application shows you how to use the DeviceFeatures Data Control to expose device features such as geolocation, e-mail, SMS, and contacts, as well as how to query the device for its properties. This feature demonstrates how to use `displayFile` method from DeviceFeatures data control to display various types files like .doc, .ppt, .xls, and .png. For more information, see Section 13.10, "Using the DeviceFeatures Data Control" and Section 13.10.9, "How to Use the displayFile Method to Enable Displaying Files."	You must also run this application on an actual device, because SMS and some of the device properties do not function on an iOS simulator or Android emulator.
ExpandCollapseComponent	This application demonstrates how to create a custom component which can act as a container for any type of AMX component and provides an expand, collapse functionality. For more details look at the `cardview.js` file under the `js` folder. This JavaScript file contains a method (`'expandcollapse.prototype.render'`) which renders the UI of the component. It also contains a method which demonstrates how to render the child components of the custom component. For more information, see Section 18.1, "Introduction to Creating Custom UI Components."	
FakeBeacon	This application demonstrates how to make use of a Cordova plugin to pretend to be an iBeacon. This application can be used in conjunction with the BeaconDemo application if you do not have an actual iBeacon for testing	
FragmentDemo	This application shows how you can use fragments to define reusable artifacts that can used as templates. It demonstrates how you can have multiple content types for each feature, one for tablet, one for phone, and use the fragment so that you don't have to code the list/form each time.	
GestureDemo	This application demonstrates how gestures can be implemented and used in MAF applications. See also Section 12.4, "Enabling Gestures."	
Java8Example	This applications demonstrates use of Java 8 language features in an AMX application.	
LayoutDemo	This application demonstrates various MAF AMX UI constructs and techniques, focusing on a variety of page layouts (flowing, stretch, or split view).	
LifecycleEvents	This application implements lifecycle event handlers on the mobile application itself and its embedded application features. This application shows you where to insert code to enable the applications to perform their own logic at certain points in the lifecycle. See also Section 10.1, "Introduction to Lifecycle Listeners in MAF Applications."	For iOS, the LifecycleEvents sample application logs data to the Console application, located at Applications-Utilities-Console application.

Table E–1 (Cont.) MAF Sample Applications

Application Name	Description	Additional Resources Required to Run the Sample Application
LocalNotificationDemo	This application demonstrates how to schedule and receive local notifications within a MAF application.See also Chapter 24, "Enabling and Using Notifications."	
MAFMCSUtility	This example provides the MAF MCS Utility source code in its MAFMCSUtilityView project.	
MAFMCSUtilitySample	This application demonstrates the use of the MAF MCS utility to access Oracle Mobile Cloud Service (MCS). It is optimized for use with Android and iOS tablets. The MAF MCS utility jar shipped with this application provides a set of APIs that make it easier for MAF developers to use MCS core services like analytics, diagnostics, and storage collections. The source of the utility classes is included for reference.	To run this sample app, you need an Oracle MCS subscription so you can connect to the Oracle Mobile Cloud Service. Within the Oracle MCS instance you need to create at least one mobile user, a mobile backend (MBE) to test against, and one or more sample storage collections that you then assign to the MBE.
Navigation	This application demonstrates the various navigation techniques in MAF, including bounded task flows and routers. It also Chapter 29, "Securing MAF Applications."demonstrates the various page transitions. See also Section 11.2, "Creating Task Flows."	
PrefDemo	This application demonstrates application-wide and application feature-specific user setting pages. See generally Chapter 21, "Enabling User Preferences"	
PushDemo	This application demonstrates how to register for and receive push notifications from the Apple Push Notification and Google Could Messaging servers. For more information, see Section 24.2, "Enabling Push Notifications."	
RangeChangeDemo	This sample demonstrates how to invoke a `rangeChangeListener` by invoking a Java handler method when the Load More Rows link within the List View component is activated or when the List View is scrolled to the end. It also demonstrates that `rangeChangeListener` is called every time new data is being fetched by the List View. It shows how you can use `RangeChangeEvent` to define whether or not more data is available to download on the client.	

Table E–1 (Cont.) MAF Sample Applications

Application Name	Description	Additional Resources Required to Run the Sample Application
RESTEasy	This example used the REST Service Editor feature in OEPE to declaratively build a MAF application that invokes REST services. For more information, see Chapter 16, "Working with REST Services."	To run this example, a mock REST server is provided to test against. See *OEPE_install*/plugins/oracle.eclipse.tools.maf.examples.nnn/examples/MockHRReSTServer.zip.README.txt for details on how to import and run the server. As provided, the example is configured to run in an Android emulator, but can easily be re-configured to run in an iOS simulator. (The difference stems from how the different tools reference a port on the host machine; in practice, a device would always be configured to access a specific server, and not a relative host process.) To switch between Android and iOS, use one of the following methods. In the Connections view (lower left in the default MAF perspective), double click RESTEasy/HRRestService and edit the URL in the Manage Connection dialog (host is 127.0.0.1 for iOS, 10.0.2.2 for Android.) Or, in the example RESTEasy project, edit adf/META-INF/connections.xml and comment/uncomment the appropriate urlconnection elements (note that only one of the elements should be uncommented at any one time; uncommenting both will not work as perhaps expected.
SecurityDemo	This application demonstrates how to secure a MAF application, configure authentication and the login server, use the Access Control Service, and access secure web services. For more information, see Chapter 29, "Securing MAF Applications."	This sample application connects to the ACS.ear application deployed to Oracle Weblogic Server. To deploy the ACS.ear follow the instructions at *OEPE-install*\plugins\oracle.eclipse.tools.maf.examples.v21_2.1.0.2015*nnnnnn*\examples\ACS.ear.README.txt.
SkinningDemo	This application demonstrates how to skin applications and add a unique look and feel by either overriding the supplied style sheets or extending them with their own style sheets. This application also shows how skins control the styling of MAF AMX UI components based on the type of device. It also demonstrates the ability to change skin families (out-of-the-box or custom) at runtime. See also Chapter 7, "Skinning MAF Applications."	
SlidingWindows	This application demonstrates the use of the AdfmfSlidingWindowUtilities API, which can be used to display multiple features on the screen at the same time. This sample shows how you can create a custom springboard or create a global navigation bar using the AdfmfSlidingWindowUtilities API.	

Table E–1 (Cont.) MAF Sample Applications

Application Name	Description	Additional Resources Required to Run the Sample Application
StockTracker	This sample demonstrates how data change events use Java to enable data changes to be reflected in the user interface. It has a variety of layout use cases, gestures, and basic mobile patterns. This sample also demonstrates how to use client-side validations with the `amx:validationGroup` component. Within the Portfolio feature, look at editStock.amx for details. See also Section 13.12, "About Data Change Events."	
URLSchemeDemo	This application demonstrates how to define a custom URL scheme for your application so that you can invoke it from a URL link in a browser or e-mail. See also Section 4.10, "Using Custom URL Schemes in MAF Applications."	
WorkBetter	This human resources application contains two features: People and Organizations.People: This feature includes a search component, which allows you to search for people. It also demonstrates the ability to create custom components as well as how to build reusable layouts as fragments and use them between different features. It demonstrates how to use various DVT visualization components to display performance, compensation, and timeline-related information. Organizations: Like the People feature, this feature demonstrates how to build reusable layouts as fragments and use them between different features. It also demonstrates how to create views for different form factors and configure them. This application is meant to be an end-to-end demo of the various UI techniques and components available. It shows a variety of layout patterns and demonstrates various uses for both common and more complex components. It is not meant to showcase a compete application but rather focus on the user interface. Please consult other samples for things like data-model or web services.	