Oracle® Mobile Application Framework

Developing Mobile Applications with Oracle Mobile Application Framework

2.4.0.0.0

E81103-01

March 2017

Documentation that describes how to use Oracle JDeveloper to create mobile applications that run natively on devices.

ORACLE

Oracle Mobile Application Framework Developing Mobile Applications with Oracle Mobile Application Framework, 2.4.0.0.0

E81103-01

Copyright © 2014, 2017, Oracle and/or its affiliates. All rights reserved.

Primary Authors: Walter Egan, Sujatha Joseph

Contributing Authors: Puneeta Bharani

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 xxii

What's New in This Guide for MAF Release 2.4.0 xxiii

New and Changed Features for MAF Release 2.4.0 xxiii
Other Significant Changes in this Document for MAF Release 2.4.0 xxv

1 Introduction to Oracle Mobile Application Framework

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

2 Getting Started with MAF Application Development

2.1 Introduction to Declarative Development for MAF Applications 2-1
2.2 Creating a MAF Application 2-2
2.2.1 How to Create a MAF Application 2-3
2.2.2 What Happens When You Create a MAF Application 2-3
2.3 Defining Application Features for a MAF Application 2-4
2.3.1 How to Define an Application Feature 2-5
2.4 Adding Content to an Application Feature 2-6
2.5 Adding Application Features to a MAF Application 2-6
2.5.1 How to Add an Application Feature to a MAF Application 2-7
2.5.2 What You May Need to Know About Feature Reference IDs and Feature IDs 2-7
2.6 Creating MAF AMX Pages and MAF Task Flows 2-8
2.6.1 How to Create a MAF AMX Page 2-11
2.6.2 How to Create MAF Task Flows 2-13
2.6.3 What Happens When You Create MAF AMX Pages and Task Flows 2-14

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 ... 3-3
3.4 Setting Display Properties for an Application Feature ... 3-4

4 Configuring the Application Navigation

4.1 Introduction to the Display Behavior of MAF Applications ... 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 ... 4-5
4.2.4 How to Set the Display Order for Application Features ... 4-5
4.3 What Happens When You Configure the Navigation Options ... 4-6
4.4 What Happens When You Set the Animation for the Springboard ... 4-7
4.5 What You May Need to Know About Custom Springboard Application Features with HTML Content ... 4-8
4.6 What You May Need to Know About Custom Springboard Application Features with MAF AMX Content ... 4-9
4.7 What You May Need to Know About the Runtime Springboard Behavior ... 4-12
4.8 Restarting an Application Feature in a MAF Application ... 4-12
4.8.1 How to Restart an Application Feature ... 4-13
4.8.2 What Happens When You Restart an Application Feature ... 4-13
4.9 Navigating a MAF Application Using Android's Back Button ... 4-13
4.9.1 How to Configure Behavior of the Android System Back Button ... 4-15
4.10 Creating a Sliding Window in a MAF Application ... 4-18
4.11 Using Custom URL Schemes in MAF Applications ... 4-19

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 ... 5-5
5.4 Configuring the Web View of Application Features with AMX Content on iOS ... 5-7
5.5 Selecting External Resources for Use in Application Features ... 5-8

6 Creating the Client Data Model in a MAF Application

6.1 Introduction to the Client Data Model in a MAF Application ... 6-2
6.2 Overview of Creating a Client Data Model in a MAF Application ... 6-3
6.3 Connecting to a REST Service to Create the Client Data Model ... 6-4
6.3.1 How to Connect to the REST Service to Retrieve Data Objects ... 6-4
6.3.2 What You May Need to Know About the MCS Anonymous Access Key ... 6-6
6.4 Discovering Candidate Data Objects for the Client Data Model ... 6-6

6.4.1 How to Discover Data Objects Using a REST Resource URL 6-7
6.4.2 How to Discover Data Objects Using a Sample Payload 6-8
6.4.3 How to Discover Data Objects Using a RAML File 6-9
6.4.4 What You May Need to Know About the Flatten Nested Data Objects Option 6-10
6.4.5 How to Discover Data Objects Using ADF BC REST Describe 6-10
6.5 Selecting and Persisting Data Objects for the Client Data Model 6-11
6.5.1 How to Select and Persist Data Objects 6-11
6.5.2 How to Create New Data Objects 6-13
6.5.3 How to Modify Data Object Attributes 6-13
6.6 Specifying Parent-Child Relationships for Data Objects 6-14
6.6.1 How to Specify a Parent-Child Relationship for Data Objects 6-15
6.7 Defining CRUD REST Resources 6-16
6.8 Specifying CRUD REST Resource Details 6-18
6.8.1 How to Specify GET (Read) Resource Details 6-19
6.8.2 How to Specify Non-GET (Write) Resource Details 6-20
6.8.3 How to Add Custom Resources 6-21
6.8.4 How to Specify Query and Path Parameters 6-22
6.8.5 How to Add HTTP Header Parameters 6-23
6.9 Setting Runtime Options for the Client Data Model 6-23
6.10 Generating the Client Data Model 6-25
6.11 Editing the Client Data Model in a MAF Application 6-26
6.12 Accessing the SQLite Database Using the MAF Client Data Model DBPersistenceManager 6-27
6.13 Defining a Custom Resource 6-30
6.14 Executing Custom Logic After CRUD REST Calls 6-32
6.15 Getting Programmatic Access to Service Objects 6-34
6.16 Understanding Usage of the Primary Key 6-35
6.17 Using Filtered Entity Lists 6-36
6.18 Creating a User Interface from a MAF Client Data Model 6-38
6.18.1 How to Create Data Controls from the Client Data Model 6-39
6.18.2 What Happens When You Create a Data Control from the Client Data Model 6-39
6.18.3 How to Use the MAF User Interface Generator 6-40
6.18.4 What Happens When You Generate a User Interface 6-42
6.19 Synchronizing Offline Transactions from a MAF Application 6-43
6.19.1 How to View Pending Synchronization Actions 6-44
6.19.2 How to Add Custom Logic to Handle Failed Synchronization Actions 6-46
6.19.3 What You May Need to Know About Disabling Automatic Synchronization 6-47
6.20 Understanding the Client Data Model's Support for Data Change Events 6-48
6.21 Forcing Offline Mode in a MAF Application 6-50
6.22 Using a Visual Indicator for Running Background Tasks 6-51

7 Using Oracle Mobile Cloud Service Platform APIs in a MAF Application

7.1 Introduction to Using Oracle Mobile Cloud Service Platform APIs 7-1

7.2 Accessing Oracle Mobile Cloud Service User Information 7-2
7.3 Accessing Files in an Oracle Mobile Cloud Service Storage Collection 7-3
7.3.1 How to Create the StorageObjectService Bean Data Control 7-4
7.3.2 What Happens When You Create the StorageObject Bean Data Control 7-5
7.3.3 How to Retrieve All Files from an Oracle Mobile Cloud Service Storage Collection . 7-6
7.3.4 How to Filter the List of Storage Objects from an MCS Storage Collection 7-7
7.3.5 How to Retrieve a Single File from an Oracle Mobile Cloud Service Storage Collection 7-8
7.3.6 How to Associate Storage Objects with Data in your MAF Application 7-8
7.4 Sending Analytics Information to Oracle Mobile Cloud Service 7-13
7.4.1 How to Configure the Transfer of Analytics to Oracle Mobile Cloud Service 7-14
7.4.2 How to Programmatically Send Analytics to Oracle Mobile Cloud Service 7-16
7.4.3 How to Send Context Events to Oracle Mobile Cloud Service 7-18
7.4.4 How to Send Analytics to Other Repositories 7-20
7.4.5 How to Add Location Information to Events 7-21
7.4.6 MAF Framework Events that Capture Analytics Information 7-21
7.5 Sending Diagnostic Information to Oracle Mobile Cloud Service 7-24

8 Localizing MAF Applications

8.1 Introduction to MAF Application Localization 8-1
8.2 Setting Resource Bundle Options for a MAF Application 8-2
8.2.1 How to Set the Resource Bundle Options for a MAF Application 8-2
8.3 Defining Text Resources in a Base Resource Bundle 8-3
8.3.1 How to Define a Text Resource in a Base Resource Bundle 8-4
8.3.2 What Happens When You Define a Text Resource in a Base Resource Bundle 8-5
8.4 Creating Locale-Specific Resource Bundles 8-7
8.4.1 How to Create a Locale-Specific Resource Bundle 8-8
8.5 Editing Resources in Resource Bundles 8-9
8.6 Localizing Image Files in a MAF Application 8-10
8.7 MAF Support of Languages 8-12
8.8 Localizable MAF Properties 8-13

9 Skinning MAF Applications

9.1 Introduction to MAF Application Skins 9-1
9.1.1 About the maf-config.xml File 9-3
9.1.2 About the maf-skins.xml File 9-4
9.2 Adding a Custom Skin to an Application 9-6
9.3 Specifying a Skin for an Application to Use 9-6
9.4 Registering a Custom Skin 9-7
9.5 Versioning MAF Skins 9-8
9.6 What Happens When You Version Skins 9-8
9.7 Overriding the Default Skin Styles 9-9
9.8 What You May Need to Know About Skinning 9-11

9.9 Adding a New Style Sheet to a Skin 9-12
9.10 Enabling End Users Change an Application's Skin at Runtime 9-12
9.11 What Happens at Runtime: How End Users Change an Application's Skin 9-15

10 Using Plugins in MAF Applications

10.1 Introduction to Using Plugins in MAF Applications 10-1
10.2 Enabling a Core Plugin in Your MAF Application 10-2
10.2.1 How to Enable a Core Plugin in Your MAF Application 10-3
10.2.2 What Happens When You Enable a Core Plugin in Your MAF Application 10-3
10.3 Registering Additional Plugins in Your MAF Application 10-3
10.3.1 How to Register an Additional Plugin 10-3
10.3.2 What Happens When You Register an Additional Plugin for Your MAF Application 10-4
10.4 Deploying Plugins with Your MAF Application 10-5
10.5 Importing Plugins from a Feature Archive File 10-6
10.6 Using a Plugin in a MAF Application 10-6
10.7 Providing Usage Descriptions for Plugins that Access Device Capabilities on iOS 10-9

11 Customizing MAF Application Artifacts with MDS

11.1 Introduction to Applying MDS Customizations to MAF Files 11-1
11.2 Customizing MAF Applications with MDS 11-2
11.3 Configuring Customization Layers 11-4
11.3.1 How to Configure the Layer Values Globally 11-5
11.3.2 How to Configure the Application-Level Layer Values 11-6
11.4 Creating Customization Classes 11-8
11.5 Consuming Customization Classes 11-11
11.6 Understanding a Customization Developer Role 11-15
11.6.1 How to Switch to the Customization Developer Role in JDeveloper 11-16
11.6.2 What You May Need to Know About the Tip Layer 11-16
11.7 Enabling Customizations in Resource Bundles 11-17
11.7.1 How to Create an Application Resource Bundle 11-17
11.7.2 How to Create a Project Resource Bundle 11-18
11.8 Upgrading a MAF Application with Customizations 11-20
11.8.1 What Happens in JDeveloper When You Upgrade Applications 11-22
11.8.2 What You May Need to Know About Upgrading FARs 11-23

12 Using Lifecycle Listeners in MAF Applications

12.1 Introduction to Lifecycle Listeners in MAF Applications 12-1
12.2 Registering a Lifecycle Listener for a MAF Application or an Application Feature 12-4
12.3 What Happens When You Register a Lifecycle Listener 12-5

13 Creating MAF AMX Pages

13.1 Introduction to the MAF AMX Application Feature 13-1

13.2 Creating Task Flows 13-1
13.2.1 How to Create a Task Flow 13-2
13.2.2 What You May Need to Know About Task Flow Activities and Control Flows 13-9
13.2.3 What You May Need to Know About the ViewController-task-flow.xml File 13-10
13.2.4 What You May Need to Know About the MAF Task Flow Diagrammer 13-11
13.2.5 How to Add and Use Task Flow Activities 13-11
13.2.6 How to Define the Data Control Context Depth for Task Flows 13-27
13.2.7 How to Define Control Flows 13-28
13.2.8 What You May Need to Know About MAF Support for Back Navigation 13-31
13.2.9 How to Enable Page Navigation by Dragging 13-32
13.2.10 How to Specify Action Outcomes Using UI Components 13-32
13.2.11 How to Create and Reference Managed Beans 13-32
13.2.12 How to Specify the Page Transition Style 13-38
13.2.13 What You May Need to Know About Bounded and Unbounded Task Flows 13-40
13.3 Creating Views 13-48
13.3.1 How to Work with MAF AMX Pages 13-49
13.3.2 How to Add UI Components to a MAF AMX Page 13-72
13.3.3 How to Add Data Controls to a MAF AMX Page 13-86
13.3.4 What You May Need to Know About the Server Communication 13-118

14 Creating the MAF AMX User Interface

14.1 Introduction to Creating the User Interface for MAF AMX Pages 14-1
14.2 Designing the Page Layout 14-2
14.2.1 How to Use a View Component 14-6
14.2.2 How to Use a Panel Page Component 14-6
14.2.3 How to Use a Panel Group Layout Component 14-7
14.2.4 How to Use a Panel Form Layout Component 14-7
14.2.5 How to Use a Panel Stretch Layout Component 14-9
14.2.6 How to Use a Panel Label And Message Component 14-9
14.2.7 How to Use a Facet Component 14-10
14.2.8 How to Use a Popup Component 14-16
14.2.9 How to Use a Panel Splitter Component 14-21
14.2.10 How to Use a Spacer Component 14-22
14.2.11 How to Use a Table Layout Component 14-23
14.2.12 How to Use a Masonry Layout Component 14-24
14.2.13 How to Use an Accessory Layout Component 14-27
14.2.14 How to Use a Deck Component 14-28
14.2.15 How to Use a Flex Layout Component 14-29
14.2.16 How to Use the Fragment Component 14-30
14.3 Creating and Using UI Components 14-32
14.3.1 How to Use the Input Text Component 14-34
14.3.2 How to Use the Input Number Slider Component 14-39
14.3.3 How to Use the Input Date Component 14-40

14.3.4 How to Use the Output Text Component 14-40
14.3.5 How to Use Buttons 14-41
14.3.6 How to Use Links 14-48
14.3.7 How to Display Images 14-49
14.3.8 How to Use the Checkbox Component 14-50
14.3.9 How to Use the Select Many Checkbox Component 14-51
14.3.10 How to Use the Choice Component 14-53
14.3.11 How to Use the Select Many Choice Component 14-55
14.3.12 How to Use the Boolean Switch Component 14-56
14.3.13 How to Use the Select Button Component 14-57
14.3.14 How to Use the Radio Button Component 14-58
14.3.15 How to Use List View and List Item Components 14-59
14.3.16 How to Use a Carousel Component 14-86
14.3.17 How to Use the Film Strip Component 14-90
14.3.18 How to Use Verbatim Component 14-92
14.3.19 How to Use an Output HTML Component 14-94
14.3.20 How to Enable Iteration 14-95
14.3.21 How to Refresh Contents of UI Components 14-96
14.3.22 How to Use the Action Listener 14-100
14.3.23 How to Use the Set Property Listener 14-102
14.3.24 How to Use the Client Listener 14-104
14.3.25 How to Convert Date and Time Values 14-107
14.3.26 How to Convert Numeric Values 14-111
14.3.27 How to Enable Drag Navigation 14-113
14.3.28 How to Use the Loading Indicator 14-117
14.4 Creating Custom UI Components 14-119
14.5 Enabling Gestures 14-121
14.6 Implementing Application Shortcuts for Use on iOS Devices with 3D Touch Support 14-123
14.7 Providing Data Visualization 14-125
14.7.1 How to Create an Area Chart 14-129
14.7.2 How to Create a Bar Chart 14-132
14.7.3 How to Create a Range Chart 14-133
14.7.4 How to Create a Bubble Chart 14-134
14.7.5 How to Create a Combo Chart 14-136
14.7.6 How to Create a Line Chart 14-137
14.7.7 How to Create a Pie Chart 14-140
14.7.8 How to Create a Scatter Chart 14-143
14.7.9 How to Create a Spark Chart 14-145
14.7.10 How to Create a Funnel Chart 14-146
14.7.11 How to Create a Stock Chart 14-148
14.7.12 How to Style Chart Components 14-151
14.7.13 How to Use Events with Chart Components 14-152
14.7.14 What You May Need to Know About Customization of Chart Tooltips 14-153

14.7.15 How to Enable Sorting of Charts with Categorical Axis 14-153
14.7.16 How to Define the Initial Zooming of Charts 14-153
14.7.17 How to Define Stacking of Specific Chart Series 14-153
14.7.18 How to Enable Split Dual-Y Axis in Charts 14-154
14.7.19 How to Create a LED Gauge 14-154
14.7.20 How to Create a Status Meter Gauge 14-154
14.7.21 How to Create a Dial Gauge 14-156
14.7.22 How to Create a Rating Gauge 14-157
14.7.23 How to Define Child Elements for Chart and Gauge Components 14-159
14.7.24 How to Create a Geographic Map Component 14-163
14.7.25 How to Create a Thematic Map Component 14-168
14.7.26 How to Use Events with Map Components 14-175
14.7.27 How to Create a Treemap Component 14-176
14.7.28 How to Create a Sunburst Component 14-180
14.7.29 How to Create a Timeline Component 14-182
14.7.30 How to Create an NBox Component 14-186
14.7.31 How to Create a Picto Chart 14-188
14.7.32 How to Define Child Elements for Map Components, Sunburst, Treemap, Timeline, Picto Chart, and NBox 14-190
14.7.33 How to Create Databound Data Visualization Components 14-191
14.7.34 How to Create Data Visualization Components Based on Static Data 14-202
14.7.35 How to Enable Interactivity in Chart Components 14-202
14.7.36 How to Create Polar Charts 14-203
14.8 Styling UI Components 14-203
14.8.1 How to Use Component Attributes to Define Style 14-203
14.8.2 What You May Need to Know About Skinning 14-206
14.8.3 What You May Need to Know About Using CSS ID Selectors for Skinning 14-206
14.8.4 How to Style Data Visualization Components 14-206
14.9 Understanding MAF Support for Accessibility 14-209
14.9.1 How to Configure UI and Data Visualization Components for Accessibility 14-210
14.9.2 What You May Need to Know About the Basic WAI-ARIA Terms 14-213
14.9.3 What You May Need to Know About the Oracle Global HTML Accessibility Guidelines 14-216
14.10 Validating Input 14-216
14.11 Using Event Listeners 14-219
14.11.1 What You May Need to Know About Constrained Type Attributes for Event Listeners 14-222

15 Using Bindings and Creating Data Controls in MAF AMX

15.1 Introduction to Bindings and Data Controls 15-1
15.2 About Object Scope Lifecycles 15-2
15.2.1 What You May Need to Know About Object Scopes and Task Flows 15-3
15.3 Creating EL Expressions 15-4

15.3.1 About Data Binding EL Expressions 15-5
15.3.2 How to Create an EL Expression 15-6
15.3.3 What You May Need to Know About MAF Binding Properties 15-12
15.3.4 How to Enable Retention of Data Provider State Across Iterators 15-12
15.3.5 How to Reference Binding Containers 15-13
15.3.6 About the Categories in the Expression Builder 15-15
15.3.7 About EL Events 15-23
15.3.8 How to Use EL Expressions Within Managed Beans 15-23
15.4 Creating and Using Managed Beans 15-24
15.4.1 How to Create a Managed Bean in JDeveloper 15-25
15.4.2 What Happens When You Use JDeveloper to Create a Managed Bean 15-26
15.5 Exposing Business Services with Data Controls 15-27
15.5.1 How to Create Data Controls 15-27
15.5.2 What Happens in Your Project When You Create a Data Control 15-27
15.5.3 Data Control Built-in Operations 15-29
15.6 Creating Databound UI Components from the Data Controls Panel 15-31
15.6.1 How to Use the Data Controls Panel 15-32
15.6.2 What Happens When You Use the Data Controls Panel 15-34
15.7 What Happens at Runtime: How the Binding Context Works 15-35
15.8 Configuring Data Controls 15-36
15.8.1 How to Edit a Data Control 15-37
15.8.2 What Happens When You Edit a Data Control 15-37
15.8.3 What You May Need to Know About MDS Customization of Data Controls 15-39
15.9 Working with Attributes 15-39
15.9.1 How to Designate an Attribute as Primary Key 15-39
15.9.2 How to Define a Static Default Value for an Attribute 15-40
15.9.3 How to Set UI Hints on Attributes 15-40
15.9.4 What Happens When You Set UI Hints on Attributes 15-41
15.9.5 How to Access UI Hints Using EL Expressions 15-42
15.10 Creating and Using Bean Data Controls 15-42
15.10.1 What You May Need to Know About Serialization of Bean Class Variables 15-43
15.11 Sharing Instances of Data Controls Across Application Features 15-43
15.11.1 How to Share Instances of Data Controls Across Application Features 15-44
15.11.2 What Happens When You Share Instances of Data Controls Across Application Features 15-44
15.12 Using the DeviceFeatures Data Control 15-45
15.12.1 How to Use the getPicture Method to Enable Taking Pictures 15-48
15.12.2 How to Use the SendSMS Method to Enable Text Messaging 15-53
15.12.3 How to Use the sendEmail Method to Enable Email 15-55
15.12.4 How to Use the createContact Method to Enable Creating Contacts 15-58
15.12.5 How to Use the findContacts Method to Enable Finding Contacts 15-62
15.12.6 How to Use the updateContact Method to Enable Updating Contacts 15-65
15.12.7 How to Use the removeContact Method to Enable Removing Contacts 15-68

15.12.8 How to Use the startLocationMonitor Method to Enable Geolocation 15-70
15.12.9 How to Use the displayFile Method to Enable Displaying Files 15-73
15.12.10 How to Use the addLocalNotification and cancelLocalNotification Methods to Manage Local Notifications 15-75
15.12.11 What You May Need to Know About Device Properties 15-78
15.13 Validating Attributes 15-82
15.13.1 How to Add Validation Rules 15-84
15.13.2 What You May Need to Know About the Validator Metadata 15-86
15.14 Using Background Threads 15-87
15.15 Working with Data Change Events 15-87

16 Configuring End Points Used in MAF Applications

16.1 Introduction to Configuring End Points in MAF Applications 16-1
16.2 Defining the Configuration Service End Point 16-1
16.3 Creating the User Interface for the Configuration Service 16-3
16.4 About the URL Construction 16-4
16.5 Setting Up the Configuration Service on the Server 16-4

17 Using Web Services in a MAF Application

17.1 Introduction to Using Web Services in a MAF Application 17-1
17.2 Creating a Rest Service Adapter to Access Web Services 17-1
17.2.1 Accessing Input and Output Streams 17-4
17.2.2 Support for Non-Text Responses 17-5
17.3 Accessing Secure Web Services 17-6
17.3.1 How to Enable Access to Web Services 17-7
17.3.2 What Happens When You Enable Access to Web Services 17-8
17.3.3 What You May Need to Know About Credential Injection 17-9
17.3.4 What You May Need to Know About Cookie Injection 17-11
17.4 Configuring the Browser Proxy Information 17-11

18 Using the Local Database in MAF AMX

18.1 Introduction to the Local SQLite Database Usage 18-1
18.1.1 Differences Between SQLite and Other Relational Databases 18-2
18.2 Using the Local SQLite Database 18-4
18.2.1 How to Connect to the Database 18-4
18.2.2 How to Use SQL Script to Initialize the Database 18-5
18.2.3 How to Initialize the Database on a Desktop 18-7
18.2.4 What You May Need to Know About Commit Handling 18-8
18.2.5 Limitations of MAF SQLite JDBC Driver 18-8
18.2.6 How to Use the VACUUM Command 18-8
18.2.7 How to Encrypt and Decrypt the Database 18-9

19 Customizing MAF AMX Application Feature Artifacts

19.1 Introduction to Customizing MAF AMX Pages and Artifacts 19-1
19.2 Customizing MAF AMX Pages and Artifacts 19-2

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 20-8
20.4.2 What Happens When You Enable the Browser Navigation Buttons for a Remote URL Application Feature 20-9

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 21-5
21.1.2 What Happens When You Create an Application-Level Preference Page 21-15
21.2 Creating User Preference Pages for Application Features 21-15
21.3 Using EL Expressions to Retrieve Stored Values for User Preference Pages 21-16
21.3.1 What You May Need to Know About preferenceScope 21-18
21.3.2 Reading Preference Values in iOS Native Views 21-19
21.4 Platform-Dependent Display Differences 21-19

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 22-4
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

23 Enabling and Using Notifications

23.1 Introduction to Notifications 23-1
23.2 Enabling Push Notifications 23-3
23.2.1 How to Register a Device for Push Notifications from MCS 23-5

23.2.2 What You May Need to Know About the Push Notification Payload 23-6
23.3 Managing Local Notifications 23-7
23.3.1 How to Manage Local Notifications Using Java 23-7
23.3.2 How to Manage Local Notifications Using JavaScript 23-8
23.3.3 How to Manage Local Notifications Using the DeviceFeatures Data Control 23-10
23.3.4 What You May Need to Know About Local Notification Options and the Application Behavior 23-11
23.4 Determining Application State When MAF Triggers a Notification Event 23-12

24 Displaying Error Messages in MAF Applications

24.1 Introduction to Error Handling in MAF Applications 24-1
24.2 Displaying Error Messages and Stopping Background Threads 24-2
24.2.1 How Applications Display Error Message for Background Thread Exceptions 24-3
24.3 Localizing Error Messages 24-4

25 Deploying MAF Applications

25.1 Introduction to Deployment of MAF Applications 25-1
25.2 Working with Deployment Profiles 25-2
25.2.1 About Automatically Generated Deployment Profiles 25-2
25.2.2 How to Create a Deployment Profile 25-7
25.2.3 What Happens When You Create a Deployment Profile 25-8
25.3 Deploying a MAF Application to the Android Platform 25-9
25.3.1 How to Configure Gradle Proxy Settings 25-11
25.3.2 How to Sign a MAF Application that You Deploy to the Android Platform 25-12
25.3.3 What You May Need to Know About Credential Storage 25-14
25.3.4 How to Deploy a MAF Application to the Android Platform 25-14
25.3.5 What Happens When You Deploy a MAF Application to the Android Platform 25-15
25.3.6 How to Add a Custom Image to an Android Application 25-17
25.3.7 What Happens When JDeveloper Deploys Images for Android Applications 25-18
25.4 Deploying an iOS Application 25-19
25.4.1 How to Create an iOS Deployment Profile 25-21
25.4.2 How to Deploy an iOS Application to an iOS Simulator 25-29
25.4.3 How to Deploy an Application to an iOS-Powered Device 25-30
25.4.4 What Happens When You Deploy an Application to an iOS Device 25-33
25.4.5 What You May Need to Know About Deploying an Application to an iOS-Powered Device 25-33
25.4.6 How to Distribute an iOS Application to the App Store 25-35
25.5 Deploying a MAF Application to the Universal Windows Platform 25-37
25.5.1 How to Deploy a MAF Application to the Universal Windows Platform 25-39
25.5.2 What Happens When You Deploy a MAF Application to the Universal Windows Platform 25-39
25.6 Overview of MAF Quick Deployment of Applications 25-41
25.6.1 About the Artifacts That Support Quick Deployment 25-42

25.6.2 About Requirements for Quick Deployment 25-43
25.6.3 What Happens During a Quick Deployment Session 25-43
25.6.4 How to Start the Full Deployment of an Application 25-44
25.6.5 How to Force the Full Deployment of an Application 25-44
25.6.6 What You May Need to Know About Quick Deployment Limitations 25-45
25.7 Deploying Feature Archive Files (FARs) 25-45
25.7.1 How to Create a Deployment Profile for a Feature Archive 25-46
25.7.2 How to Deploy the Feature Archive Deployment Profile 25-48
25.7.3 What Happens When You Deploy a Feature Archive File Deployment Profile 25-50
25.8 Creating a Mobile Application Archive File 25-51
25.8.1 How to Create a Mobile Application Archive File 25-52
25.9 Creating a New Application from an Application Archive 25-60
25.9.1 How to Create a New Application from an Application Archive 25-61
25.9.2 What Happens When You Import a MAF Application Archive File 25-62
25.10 Deploying MAF Applications from the Command Line Using OJDeploy 25-63

26 Understanding Secure Mobile Development Practices

26.1 Weak Server-Side Controls 26-1
26.2 Insecure Data Storage on the Device 26-2
26.2.1 Encrypting the SQLite Database 26-2
26.2.2 Securing the Device's Local Data Stores 26-2
26.2.3 About Security and Application Logs 26-3
26.3 Insufficient Transport Layer Protection 26-3
26.4 Side-Channel Data Leakage 26-4
26.5 Poor Authorization and Authentication 26-4
26.6 Broken Cryptography 26-5
26.7 Client-Side Injection From Cross-Site Scripting 26-5
26.7.1 Protecting MAF Applications from Injection Attacks Using Device Access Permissions 26-6
26.7.2 About Injection Attack Risks from Custom HTML Components 26-6
26.7.3 About SQL Injections and XML Injections 26-7
26.8 Security Decisions From Untrusted Inputs 26-7
26.9 Improper Session Handling 26-8
26.10 Lack of Binary Protections Resulting in Sensitive Information Disclosure 26-8

27 Securing MAF Applications

27.1 Introduction to MAF Security 27-1
27.2 About the User Login Process 27-2
27.3 Overview of the Authentication Process for MAF Applications 27-4
27.4 Configuring MAF Connections 27-5
27.4.1 How to Create a MAF Login Connection 27-5
27.4.2 How to Create a Multi-Tenant Aware MAF Login Connection 27-8
27.4.3 How to Configure Basic Authentication 27-10

27.4.4 How to Configure OAuth Authentication ... 27-13
27.4.5 How to Configure Single Sign-On in a MAF Application ... 27-16
27.4.6 How to Configure a Placeholder Connection for MAF Application Login ... 27-19
27.4.7 How to Update Connection Attributes of a Named Connection at Runtime ... 27-21
27.4.8 How to Store Login Credentials ... 27-23
27.4.9 What Happens When You Create a Connection for a MAF Application ... 27-25
27.4.10 What Happens When You Create a Multi-Tenant Aware Connection ... 27-26
27.4.11 What You May Need to Know About the Login Connection Configuration ... 27-27
27.4.12 What You May Need to Know About Multiple Identities for Local and Hybrid Login Connections ... 27-27
27.4.13 What You May Need to Know About Migrating a MAF Application and Authentication Modes ... 27-27
27.4.14 What You May Need to Know About Custom Headers ... 27-28
27.4.15 What Happens at Runtime: When MAF Calls a REST Web Service ... 27-28
27.4.16 What You May Need to Know About Injecting Basic Authentication Headers ... 27-28
27.4.17 What You May Need to Know About Web Service Security ... 27-29
27.4.18 How to Configure Access Control ... 27-30
27.4.19 What You May Need to Know About the Access Control Service ... 27-32
27.4.20 How to Alter the Application Loading Sequence ... 27-34
27.4.21 How to Configure Login Credentials Programmatically Prior to Authentication 27-35
27.5 Configuring Security for MAF Applications ... 27-38
27.5.1 How to Enable Application Features to Require Authentication ... 27-38
27.5.2 How to Designate the Login Page ... 27-39
27.5.3 How to Create a Custom Login HTML Page ... 27-42
27.5.4 What You May Need to Know About Login Pages ... 27-43
27.5.5 What You May Need to Know About Login Page Elements ... 27-46
27.5.6 What Happens in JDeveloper When You Configure Security for Application Features ... 27-46
27.6 Allowing Access to Device Capabilities ... 27-47
27.7 Enabling Users to Log Out from Application Features ... 27-47
27.8 Using MAF Authentication APIs ... 27-48
27.9 Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL ... 27-49
27.10 Registering SSL Certificate File Extensions in a MAF Application ... 27-51

28 Testing and Debugging MAF Applications

28.1 Introduction to Testing and Debugging MAF Applications ... 28-1
28.2 Testing MAF Applications ... 28-2
28.2.1 How to Perform Accessibility Testing on iOS-Powered Devices ... 28-2
28.3 Configuring JDeveloper and MAF Applications to Debug Code ... 28-2
28.3.1 What You May Need to Know About the Debugging Configuration ... 28-3
28.3.2 How to Enable Debugging of Java Code and JavaScript ... 28-5
28.3.3 How to Debug the MAF AMX Content ... 28-6
28.4 Debugging MAF Applications Deployed on the Android Platform ... 28-6

28.4.1 How to Debug Java Code on the Android Platform 28-7
28.4.2 How to Debug UI Code on the Android Platform 28-7
28.5 Debugging MAF Applications Deployed on the iOS Platform 28-8
28.5.1 How to Debug Java Code on the iOS Platform 28-8
28.5.2 How to Debug UI Code on the iOS Platform 28-8
28.6 Debugging MAF Applications Deployed on the Universal Windows Platform 28-16
28.6.1 How to Debug Java Code on the Universal Windows Platform 28-16
28.6.2 How to Debug UI Code on the Universal Windows Platform 28-19
28.7 Using and Configuring Logging in MAF Applications 28-21
28.7.1 How to Configure Logging Using the Properties File 28-24
28.7.2 How to Use JavaScript Logging 28-25
28.7.3 How to Use Embedded Logging 28-26
28.7.4 How to Use Xcode for Debugging and Logging on the iOS Platform 28-27
28.7.5 How to Access the Application Log 28-27
28.7.6 How to Disable Logging 28-29
28.8 Measuring MAF Application Performance 28-29
28.9 Inspecting Web Service Calls in a MAF Application 28-36

29 Reusing MAF Application Content with a Feature Archive File

29.1 Introduction to Feature Archive Files 29-1
29.2 Using FAR Content in a MAF Application 29-1
29.3 What Happens When You Add a FAR as a Library 29-3
29.4 What Happens When You Add a FAR as a View Controller Project 29-6
29.5 What You May Need to Know About Enabling the Reuse of Feature Archive Resources 29-8

30 Reusing MAF Application Content with a MAF Shared Library

30.1 Introduction to Reusing MAF Application Content with a MAF Shared Library 30-1
30.2 Creating a MAF Application for Shared Library 30-2
30.2.1 How to Create a MAF Application for Shared Library 30-2
30.2.2 What Happens When You Create a MAF Application for Shared Library 30-3
30.2.3 How to Set the Shared Library Root Directory 30-4
30.2.4 How to Describe the Shared Library Content to Consumers 30-5
30.3 Adding Artifacts to a MAF Application for Shared Library 30-6
30.4 Deploying a MAF Application for Shared Library 30-7
30.4.1 How to Deploy a MAF Application for Shared Library 30-8
30.4.2 What Happens When You Deploy a MAF Application for Shared Library 30-8
30.5 Consuming a Shared Library in a MAF Application 30-8
30.5.1 How to Consume a Shared Library in a MAF Application 30-9
30.5.2 What Happens When You Consume a Shared Library in a MAF Application 30-12
30.6 Deploying a MAF Application with Duplicate Shared Library Archive File Names 30-13
30.7 Using Cordova Plugins in Shared Libraries 30-15

31 Integrating MAF Applications with EMM Solutions

31.1 Introduction to the AppConfig Community 31-1
31.2 About the MAF Approach to Enterprise Mobile Applications 31-2
31.3 Access Control for MAF Applications with EMM Solutions 31-3
31.4 How to Manage MAF Application Configurations with EMM Solutions 31-4
31.5 Managing MAF Applications with the AirWatch EMM Solution 31-4
31.6 Managing MAF Applications with the MobileIron EMM Solution 31-6
31.7 Managing MAF Applications with the Blackberry EMM Solution 31-8
31.8 Configuring Properties in MAF Applications for Use by EMM Solutions 31-9

A Troubleshooting MAF Applications

A.1 Problems with Input Components on iOS Simulators A-1
A.2 Code Signing Issues Prevent Deployment A-2
A.3 The credentials Attribute Causes Deployment to Fail 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-3
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-10
B.2.10 resetApplication B-10
B.2.11 gotoSpringboard B-11
B.2.12 showSpringboard B-12
B.2.13 hideSpringboard B-13
B.2.14 showNavigationBar B-13
B.2.15 hideNavigationBar B-14
B.2.16 showPreferences B-15
B.2.17 invokeMethod B-15
B.2.18 invokeContainerMethod B-16
B.2.19 invokeContainerJavaScriptFunction B-17
B.2.20 sendEmail B-18
B.2.21 sendSMS B-19
B.2.22 Application Icon Badging B-19
B.3 Accessing Files Using the getDirectoryPathRoot Method B-19

B.3.1 Accessing Platform-Independent Download Locations B-20

C MAF Application and Project Files

C.1 Introduction to MAF Application and Project Files C-1
C.2 About the Application Controller Project-Level Resources C-3
C.3 About the View Controller Project Resources C-7
C.4 About the MAF Application Configuration File C-8
C.5 About the MAF Application Feature Configuration File C-9

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.2.2 maf_arrays.xml D-6
D.2.3 maf_strings.xml D-7
D.3 Converting Preferences for iOS D-8
D.4 Converting Preferences for Windows D-8

Preface

Welcome to the *Developing Mobile Applications with Oracle Mobile Application Framework*.

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 documents:

- *Installing Oracle Mobile Application Framework*
- *Installing Oracle JDeveloper*
- *Developing Applications with Oracle JDeveloper*
- *Developing Extensions for Oracle JDeveloper*
- *Understanding Oracle Web Services Manager*
- *Administering Web Services*
- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*
- *JSDoc Reference for Oracle Mobile Application Framework*

- *Java API Reference for Oracle Web Services Manager*
- *Oracle JDeveloper 12c Online Help*
- *Oracle JDeveloper 12c Release Notes* (link included with your Oracle JDeveloper 12*c* installation and on Oracle Technology Network)

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.
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 2.4.0

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 2.4.0

Oracle MAF Release 2.4.0 includes the following new and changed development features, which are described in this guide.

- MAF now uses Gradle to build and deploy MAF applications to the Android platform. MAF downloads and installs Gradle during the initial deployment of a MAF application to Android. You may need to configure Gradle proxy settings to ensure a successful installation of Gradle. See How to Configure Gradle Proxy Settings. You also need to install the Android Support Repository package in the Android SDK to successfully deploy a MAF application to the Android platform. See Setting Up Development Tools for the Android Platform in *Installing Oracle Mobile Application Framework*and Deploying a MAF Application to the Android Platform.

- MAF applications can now share one instance of a data control across multiple application features. You must designate data controls that you want to share, as described in Sharing Instances of Data Controls Across Application Features.

- MAF now allows you to share individual artifacts (for example, AMX page fragments, task flows, and data controls) between multiple MAF applications using a shared library. For more information about how to create, deploy, and consume a shared library, see Reusing MAF Application Content with a MAF Shared Library.

- MAF excludes shared libraries from application features that you to deploy to a Feature Archive File (FAR). If a FAR that you consume in your MAF application depends on a shared library for a resource, you need to obtain the shared library separately. For more information, see What Happens When You Deploy a Feature Archive File Deployment Profile.

- Using FAR Content in a MAF Application has been revised to note that MAF no longer adds a FAR to a MAF application when the FAR contains a connection that is already defined in the application. Previously, MAF added the FAR, but logged a warning message in the Log window for you to verify the configuration of the connections.

- MAF applications that you deploy to iOS devices with 3D Touch support can implement application shortcuts, as described in Implementing Application Shortcuts for Use on iOS Devices with 3D Touch Support.
- This release updates the Cordova engine versions that MAF uses (Android: 6.0.0, iOS: 4.3.0, and Windows: 4.4.3). For more information about Cordova plugins in MAF applications, see Introduction to Using Plugins in MAF Applications.
- Android 4.4 (API Level 19) is now the minimum version of Android that MAF supports for the deployment of MAF applications on the Android platform and the target SDK API level has increased to Android 6's API Level 23 from API Level 21. For more information about deploying MAF applications to Android, see Deploying a MAF Application to the Android Platform.
- You can now restart an application feature, as described in Restarting an Application Feature in a MAF Application.
- MAF has changed the location where it outputs the application log file on Android devices to `/sdcard/Android/data/<app.package>/files/<app.name>.txt` or `/data/data/<package>/files/<AppName>.txt` if the former location is not accessible. End users with Android 6+ devices must grant Storage permission to the MAF application to use `/sdcard/Android/data/<app.package>/files/<app.name>.txt`. See Using and Configuring Logging in MAF Applications. How to Access the Application Log describes how you can programmatically retrieve the location of the application log file.
- The `RestServiceAdapter` exposes a new API (`getHttpURLConnection`) that you can use to initialize and return a `java.net.HttpURLConnection` or `javax.net.ssl.HttpsURLConnection`, as described in Creating a Rest Service Adapter to Access Web Services.
- The List View component exposes new attributes (`maxRows` and `preserveState`). Use the `maxRows` attribute to specify the maximum number of list items to display. Set the `preserveState` attribute to `none` to prevent MAF applying the past state of the List View component when the end user navigates back to the page that renders the component. For more information about these attributes, see *Tag Reference for Oracle Mobile Application Framework* and Configuring Paging and Dynamic Scrolling.
- The Panel Splitter component exposes new attributes (`navigatorDisplay` and `navigatorDisclosureState`) that you can use to configure the display and disclosure state of the navigator. For more information about these attributes, see *Tag Reference for Oracle Mobile Application Framework* and How to Use a Panel Splitter Component.
- The `dvtm:marker` component supports new attributes (`markerIconDisplay` and `selectedLabelStyle`) and its `labelPosition` attribute supports a new value (`bubble`) that can be used to render information bubbles in a Geographic Map component. The CompGallery sample application, described in MAF Sample Applications, renders information bubbles in a Geographic Map component. For more information about the Geographic Map component, see How to Create a Geographic Map Component and *Tag Reference for Oracle Mobile Application Framework*.

Other Significant Changes in this Document for MAF Release 2.4.0

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

- How to Configure Single Sign-On in a MAF Application revised to describe the configuration changes you need to implement in a Cordova plugin if want to use MCS single sign-on in a MAF application that you deploy to the Universal Windows Platform.
- Content describing the Oracle Mobile Security Suite integration in MAF has been removed. Support for this integration was deprecated in MAF 2.3.3. Consider using AppConfig, described in Integrating MAF Applications with EMM Solutions, as an alternative solution.
- Removed the Caching Data in a MAF Application chapter which described the transport level data caching that was deprecated in a previous release. We recommend that customers use the MAF client data model, described in Creating the Client Data Model in a MAF Application, to cache data. The Caching Data in a MAF Application chapter continues to be available in the documentation libraries for previous releases of MAF.

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:

- Introduction to Mobile Application Framework
- About the MAF Runtime Architecture
- About Developing Applications with MAF
- MAF Sample Applications

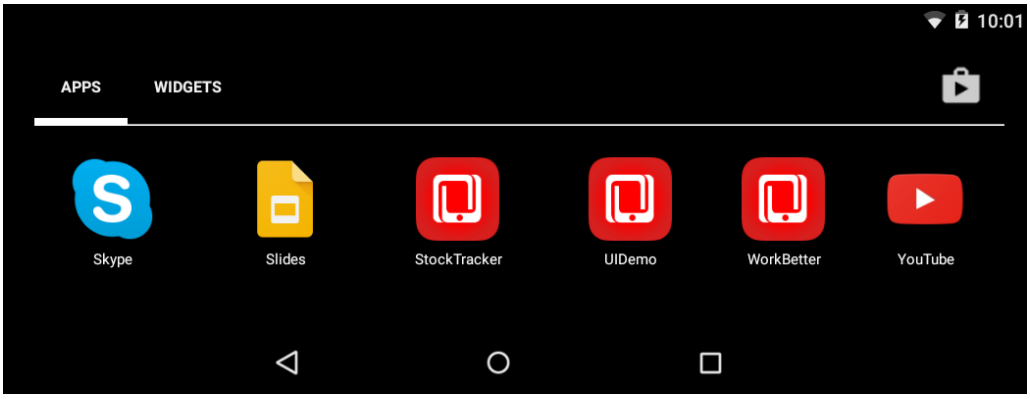
1.1 Introduction to Mobile Application Framework

MAF is a hybrid mobile architecture, one that uses HTML5 and CSS to render the user interface. MAF uses Java for the application business logic, and Apache Cordova to access device features such as GPS, camera, and email.

Because MAF uses these cross-platform technologies, you can build an application that runs on Android, iOS and Universal Windows Platform (UWP) 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 the Android SDK. Furthermore, 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.

A MAF 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), application features are independent of one another. They provide you with a way to separate the UI of your application and can be used at the same time by the end user in the MAF application. Taking our example of the customer contacts and product inventory application features, this means that end users can edit a customer contact in a MAF application, switch to check a product in the inventory, and return to edit the customer contact at the point where they left to check the inventory.

One way to think of application features is as tabs in a web browser. You open multiple tabs in a web browser when you want multiple concurrent UIs to view web pages. If you don't need multiple concurrent UIs, you use one tab to navigate from web page to web page. Given that each application feature incurs the expense of loading the MAF UI framework and all of the Cordova plugins associated with the MAF application, you should use application features sparingly in your MAF application. If you do not have a requirement for one or more concurrent UIs, use a single application feature that makes task flow calls to perform the tasks you want your MAF application to accomplish.

As application 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 independent software vendors to create applications that can be configured by specific customers.

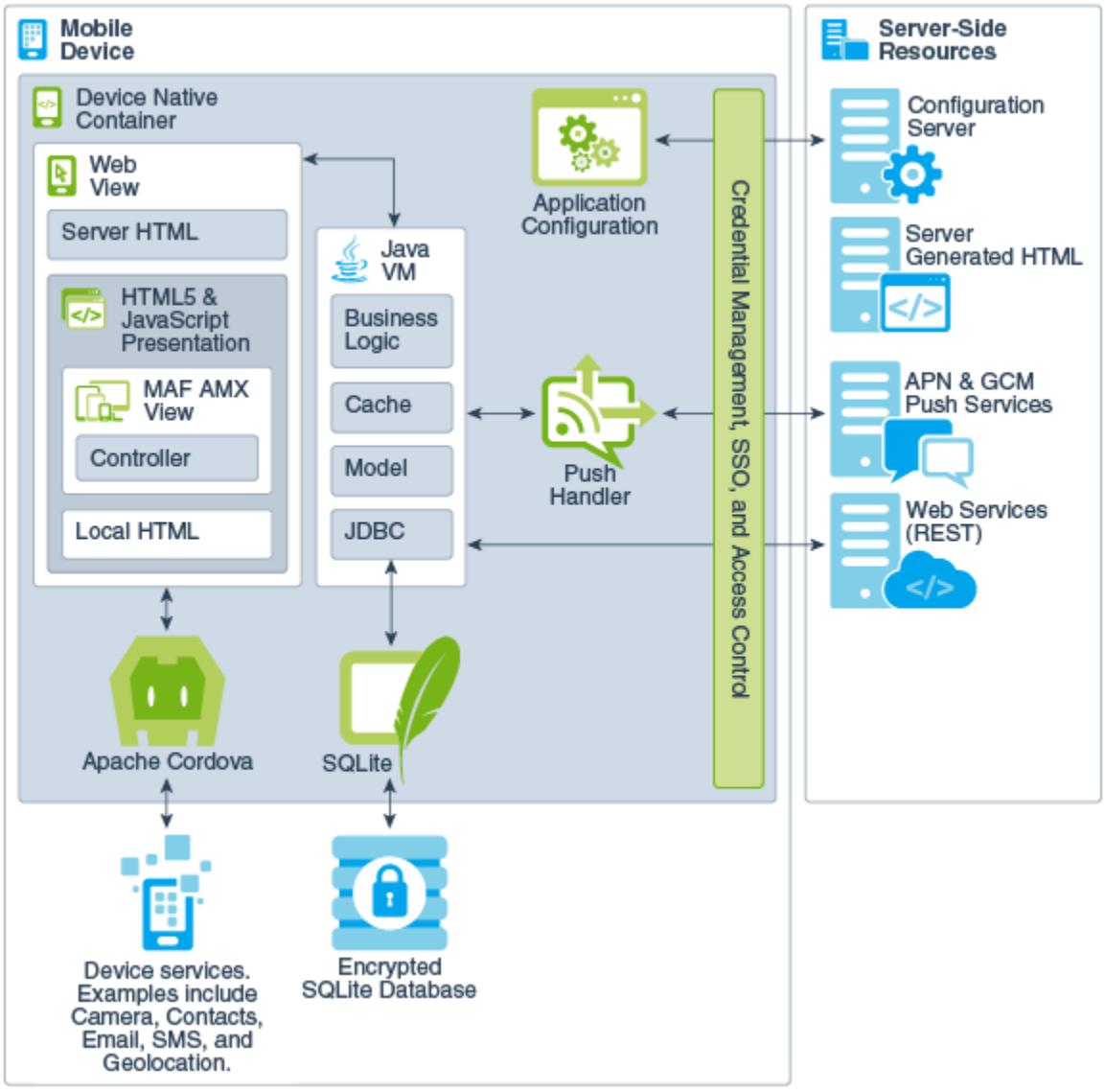
Different types of application feature can be created based on the type of content that you configure the application feature to render. The available options are AMX pages, local HTML pages, or content from a web application (remote URL option). The default content type is AMX pages. This content type allows you to take advantage of many of the other features that MAF provides to facilitate the development of your mobile application, such as an extensive suite of UI and data visualization components, task flows, and data controls that access device features. For information about these content types, including how to configure them in your mobile application, see Defining the Content Type of MAF Application Features .

1.2 About the MAF Runtime Architecture

MAF is a thin native container that is deployed to a device and 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 illustrates the MAF runtime architecture.

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 information, see the following:

- Creating MAF AMX Pages
- Creating the MAF AMX User Interface
- Using Bindings and Creating Data Controls in MAF AMX
- Using Web Services in MAF AMX
- Configuring End Points Used in MAF Applications
- Using the Local Database in MAF AMX
- Customizing MAF AMX Application Feature Artifacts
- Creating Custom 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. See 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. See 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 native features of the device 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. See 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-based web services.
  - **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. See 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. See 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 that are used by the application configuration services. The configuration server is delivered as a reference implementation. Any common WebDav services that are

hosted on a J2EE server can be used for this purpose. See Configuring End Points Used in MAF Applications .

- **Server-Generated HTML**—Web content that is hosted on remote servers that are used for browser-based application features. See 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 client data model, the Java layer, or through data controls. See Creating the Client Data Model in a MAF Application and Using Web Services in MAF AMX .

1.3 About Developing Applications with MAF

Developing a MAF application involves activities such as gathering requirements, design, development, and testing.

These activities can be grouped into the following tasks:

- Creating a mock-up of the user interface

  Start with a visual design of the MAF application so that you know what data to consume on which page and what data you want to modify. Completing this task allows you optimize the design of the REST service(s) that deliver the data.

- Create a mobile backend layer where you optimize the REST services your MAF application consumes

  MAF recommends REST services that use JSON as the payload format. This type of service is considered the best architectural choice for integration between your MAF application and the mobile backend layer. Oracle Mobile Cloud Service (MCS) provides a comprehensive solution for the creation of mobile backends that includes services for analytics, push notifications, file storage and security.

- Create the model and persistence layer. This layer contains:

  – Java classes and associated files that represent the data model of your application

  – SQLite database that stores data for offline usage

  – Data objects (also known as entities) that hold the data coming from the SQLite database and/or the REST services

  – Service objects that perform CRUD actions and other custom actions on the data objects

  – Object-relation information mapping how database tables and their columns map to entity classes and their attributes, and how entity classes and attributes map to attributes in the REST resource response payload

- Create the user interface layer

  You typically create a bean data control for each service class in the model layer. This helps you build MAF AMX pages using drag and drop from the Data

Control panel in JDeveloper. To quickly test your model layer and/or perform user prototyping, use the MAF User Interface Generator. This wizard creates a user interface and the associated task flows, AMX pages and data bindings.

MAF provides the MAF Client Data Model From REST Web Service and the MAF User Interface Generator wizards to simplify the above tasks. You can consume REST services from an MCS backend, create your model and persistence layer, and generate a MAF application with a prototype user interface complete with support for offline transactions and data synchronization once the application returns online. Use of these wizards is described in Creating the Client Data Model in a MAF Application.

> **Tip:** The Consuming and persisting REST/JSON services with Oracle MAF and the Client Data Model (CDM) tutorial takes you step-by-step through the creation of a MAF application that consumes a REST service using the aforementioned wizards.

Although the components of a MAF application may be created by a single developer, an application may 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 task flow of the application. 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 is 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.

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 launch icons used for the application in different

orientations (landscape or portrait) and on different devices (phone or tablets). See 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 JDeveloper and platform-specific tools. See 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

   The MAF 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 user roles of the application. See 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.

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.

1.4 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 JDeveloper 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.

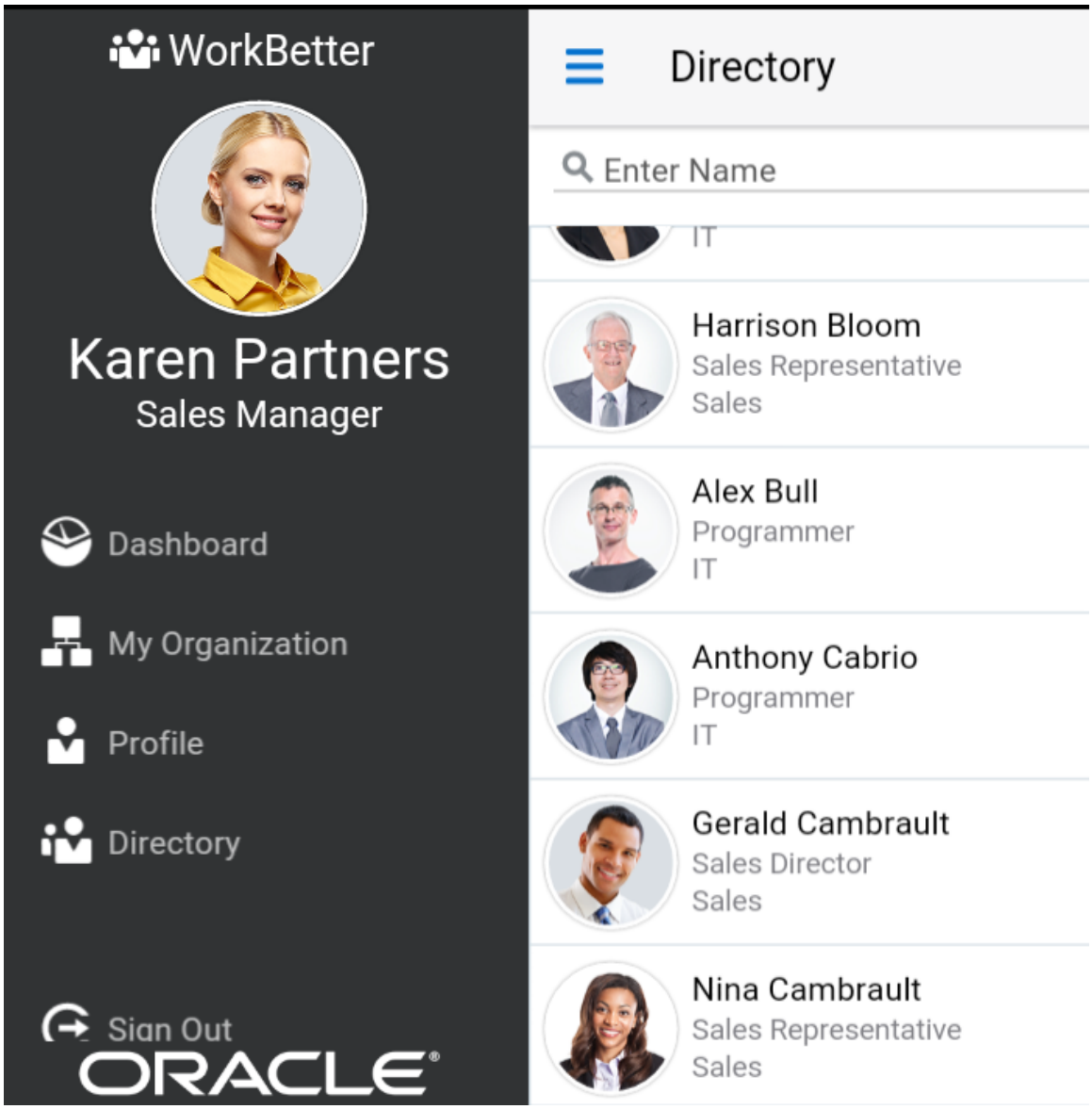
A `HelloWorld` sample application demonstrates how to implement a single application feature with a local HTML file. We suggest that you use the `HelloWorld` application to verify that your development environment is set up correctly to compile and deploy an application.

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 you install the MAF extension, you can extract the sample applications from the `PublicSamples.zip` file within the `jdev_install/jdeveloper/jdev/extensions/oracle.maf/Samples` directory. 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. Consult the `ReadMe.txt` file in the extracted directory for a description of each sample application. The Oracle Mobile Application Framework Samples page that you can access on the following Oracle Technology Network page also provides information about the sample applications. See http://www.oracle.com/technetwork/developer-tools/maf/learnmore/mafsamples-2227357.html.

Figure 1-3 WorkBetter Sample Application



2

Getting Started with MAF Application Development

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

This chapter includes the following sections:

- Introduction to Declarative Development for MAF Applications
- Creating a MAF Application
- Defining Application Features for a MAF Application
- Adding Content to an Application Feature
- Adding Application Features to a MAF Application
- Creating MAF AMX Pages and MAF Task Flows

2.1 Introduction to Declarative Development for MAF Applications

Use the overview editors and wizards that MAF provides to develop an application, define its features, add content to features, and deploy the application to a test environment or a device.

The Oracle Mobile Application Framework (MAF) extension in JDeveloper provides a number of overview editors and other 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 the JDeveloper Applications window where a number of the items that you use to develop MAF applications are identified:

1. The overview editor for the `maf-features.xml` file opens by default when you create a new MAF application. Use this overview editor to define the application features that your MAF application contains.

2. Use the overview editor for the `maf-application.xml` file used to, among other things, specify the name of the MAF application, the default navigation menus (navigation bar or springboard) that the application renders, security, and device access options for the application.

3. By default, JDeveloper creates a MAF application with two data controls (ApplicationFeatures and DeviceFeatures). These data controls expose operations that you can drag to a MAF AMX page where JDeveloper displays context menus to complete configuration of the operation when you drop it on the page. For

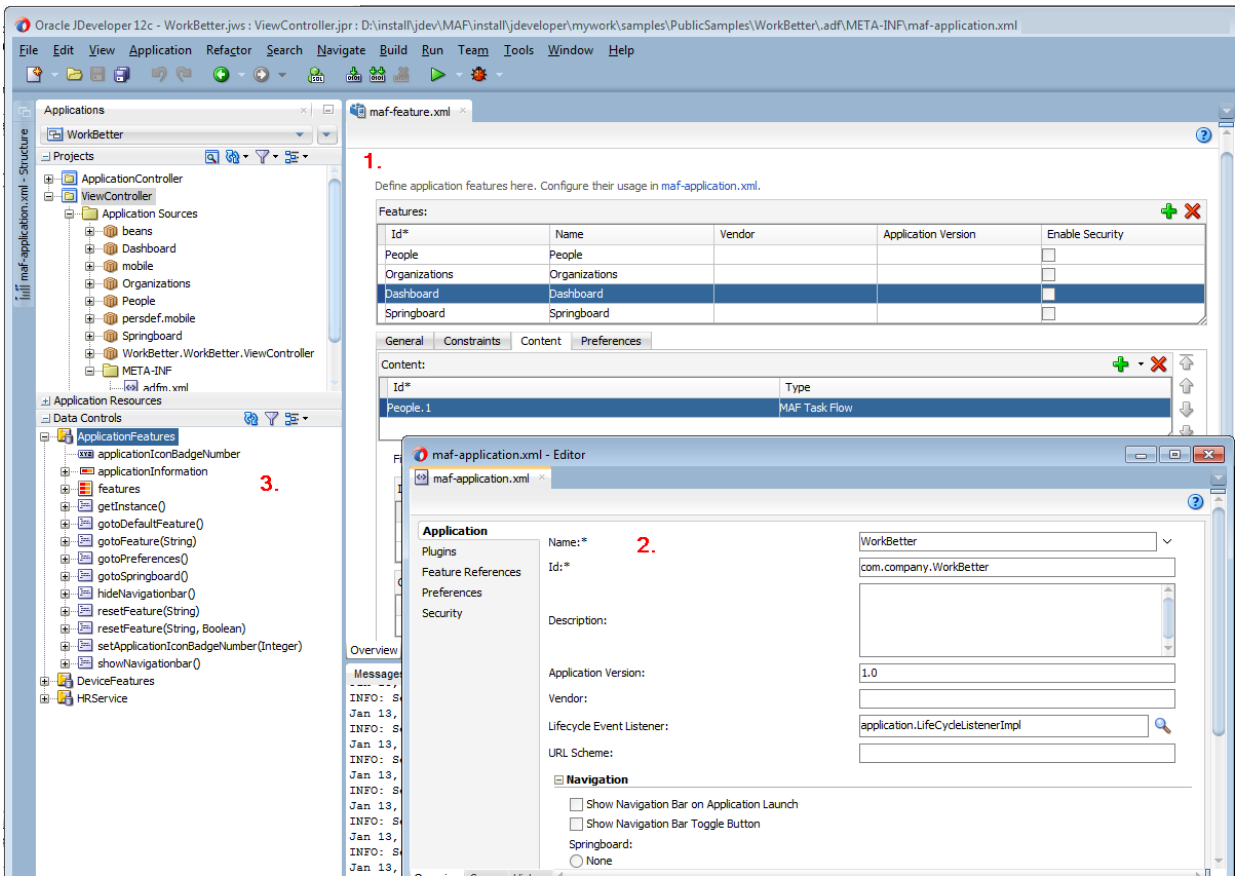
example, dragging the `hideNavigationBar()` operation to a page prompts JDeveloper to display a context menu where you configure a control for users to hide the navigation bar of an application.

The WorkBetter sample application is one of a number of sample applications that MAF provides to demonstrate how to create mobile applications using MAF. See MAF Sample Applications.

JDeveloper 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 Creating a MAF Application.
2. Define an application feature for the MAF application, as described in Defining Application Features for a MAF Application.
3. Add content to the application feature, as described in Adding Content to an Application Feature.

Figure 2-1 Overview Editors for Application Features and Application



2.2 Creating a MAF Application

Access and use the Create Mobile Application Framework Application wizard from the **Mobile Application Framework Application** option in the Applications category in New Gallery to get started with application development in MAF.

Before you can create a MAF application, download, install, and configure the MAF extension in JDeveloper.

See Introduction to Installing the MAF Extension with JDeveloper in *Installing Oracle Mobile Application Framework*. Once you have completed this task, create a MAF application using the creation wizards in JDeveloper.

2.2.1 How to Create a MAF Application

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

To create a MAF application:

1. In the main menu, select **File** and then **Application** > **New**.
2. In the **New Gallery**, in the **Items** list, double-click **Mobile Application Framework Application**.
3. In the **Create Mobile Application Framework Application** wizard, enter application and project details like name, directory, and default packages. For help with the wizard, press F1 or click **Help**.
4. Click **Finish**.

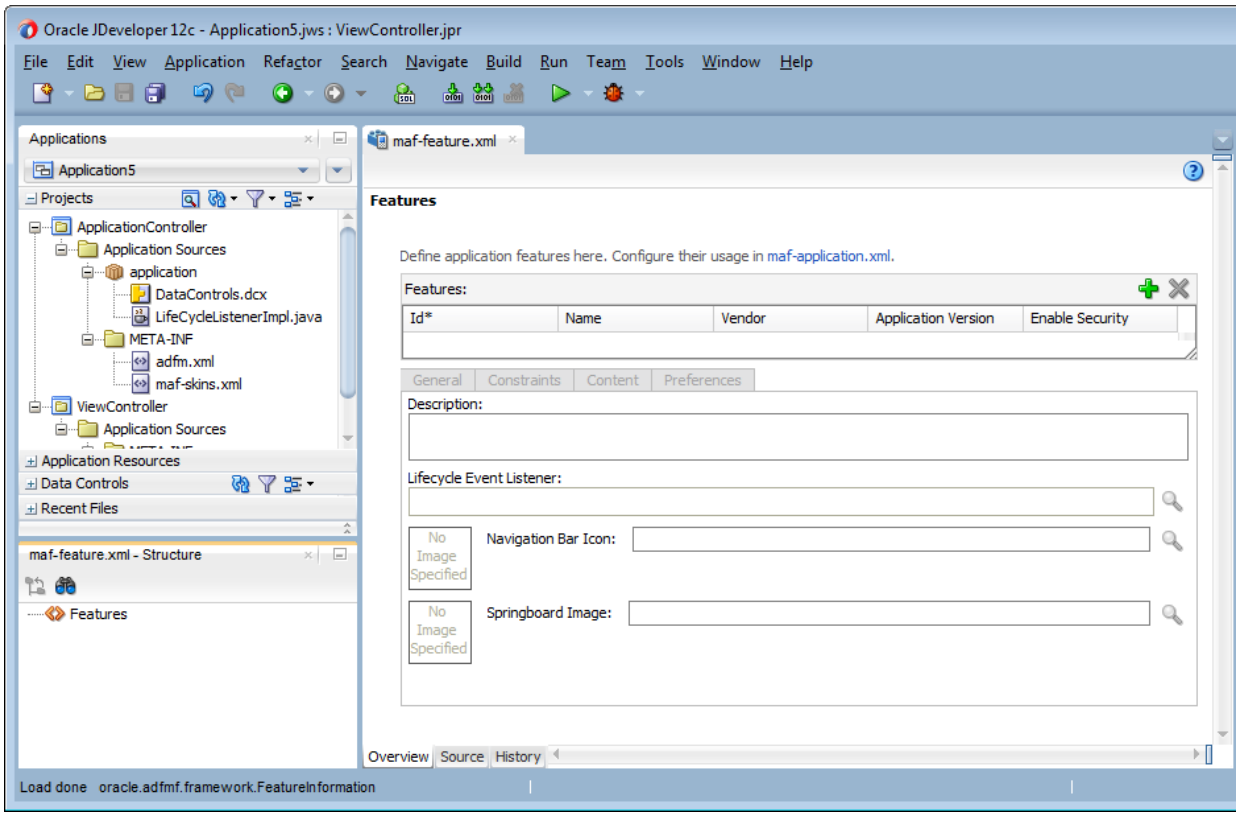
2.2.2 What Happens When You Create a MAF Application

JDeveloper creates a MAF application with two projects (ApplicationController and ViewController) and two data controls (ApplicationFeatures and DeviceFeatures). 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 supported platforms.

By default, JDeveloper opens the overview editor for the `maf-features.xml` file in the ViewController project of the newly-created MAF application, as shown in Figure 2-2. Use this overview editor to add one or more application features to your MAF application. A MAF application must have at least one application feature. See Defining Application Features for a MAF Application.

For information about the files and artifacts that JDeveloper generates when you create a MAF application, see MAF Application and Project Files.

Figure 2-2 Overview Editor for Application Features in Newly-Created MAF Application

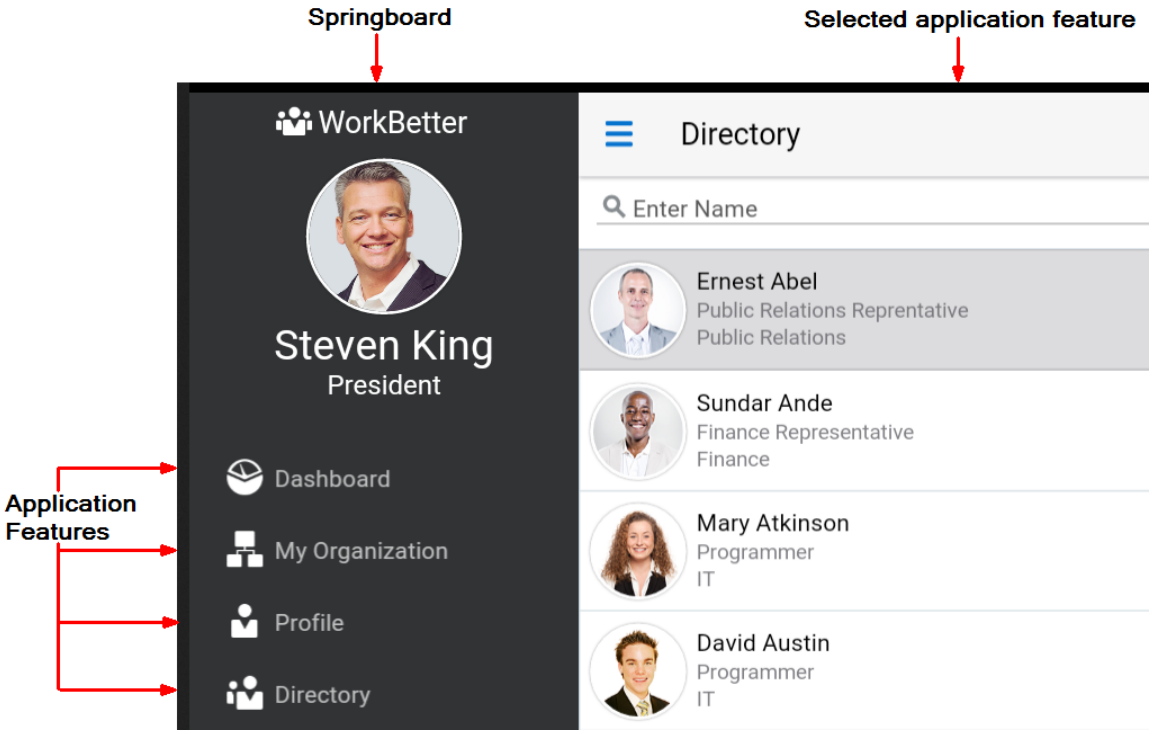


2.3 Defining Application Features for a MAF Application

You must define at least one application feature 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 2-3 shows three of these application features displaying in the custom springboard of that application.

Figure 2-3 Application Features in the Springboard of the WorkBetter Application



2.3.1 How to Define an Application Feature

Use the procedure to define an application feature for a MAF application using the Create MAF Feature dialog of the overview editor for the `maf-features.xml` file.

You define an application feature for a MAF application using the overview editor for the `maf-features.xml` file.

To define an application feature for a MAF application:

1. In the **Applications** window, expand the **ViewController** project and then **Application Sources** and **META-INF**.
2. Double-click the `maf-feature.xml` file.
3. In the **Features** page, click the **Add** icon.
4. Complete entries in the **Create MAF Feature** dialog as follows:
   - **Feature Name**: Enter the display name for the application feature.
   - **Feature ID**: Enter a unique ID for the application feature or accept the value that JDeveloper generates.
   - **Directory**: Specify the directory for the application feature or accept the value that JDeveloper generates.
   - Select the **Add a corresponding feature reference to maf-application.xml** checkbox to add the application feature to the MAF application. By default, JDeveloper selects this checkbox.
5. Click **OK**.

2.4 Adding Content to an Application Feature

After you define an application feature, you can add MAF AMX pages, MAF Task Flows, Local HTML, or Remote HTML as content for the feature.

One of the tasks to do after you define an application feature is to add content to the application feature. Select from among the following types to render content in your application feature:

- **MAF AMX Page**: Select this content type if you want the application feature to render MAF AMX pages.
- **MAF Task Flow**: Select 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**: Select if you want the application feature to render HTML page.
- **Remote URL**: Select 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 select the type of content to add to the application feature in the Content tab of the Features page of the overview editor of the `maf-features.xml` file. For the specific steps for each content type, see Defining the Content Type of MAF Application Features .

2.5 Adding Application Features to a MAF Application

Add a feature to a MAF application that you create or import. A feature can be added to an application as you create it, or after you have created the application.

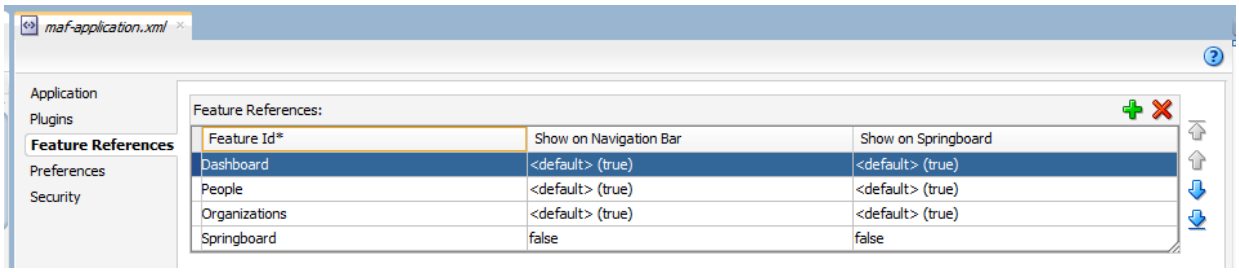
You can automatically add an application feature to a MAF application when you define it by selecting the **Add a corresponding feature reference to maf-application.xml** checkbox in the **Create MAF Feature** dialog, as described in How to Define an Application Feature.

Use the **Feature References** page of the overview editor of the `maf-application.xml` file if you want to add an application feature that you did not add to the MAF application when you created it, you use the **Feature References** page of the overview editor of the `maf-application.xml` file.

You can also add application features to your MAF application that you import from Feature Archive (FAR) files. You must import the application feature into your MAF application before you can add an application feature to the MAF application. See Reusing MAF Application Content .

Figure 2-4 shows the **Feature References** page where you add application features to a MAF application.

Figure 2-4 Adding Application Features Using the Feature References Page



2.5.1 How to Add an Application Feature to a MAF Application

Use the procedure to access and use the Feature References page in the overview editor of the `maf-application.xml` file to add application features to a MAF application.

You use the Feature References page in the overview editor of the `maf-application.xml` file to add application features to a MAF application.

To add an application feature to a MAF application:

1. In the **Applications** window, expand the Application Resources panel.
2. In the **Application Resources** panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the **maf-application.xml** file and in the overview editor that appears, click the **Feature References** navigation tab.
4. In the **Feature References** page, click the **Add** icon.
5. In the **Insert Feature Reference** dialog, select the ID of the application feature from the drop-down list.
6. Click **OK**.

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

Application feature IDs must be unique, and must match feature reference IDs.

JDeveloper 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.

Example 2-1 shows the entries for the People application feature in the `maf-application.xml` and `maf-feature.xml` files of the WorkBetter sample application .

Example 2-1 Feature Reference and Feature ID for an Application Feature in WorkBetter Application

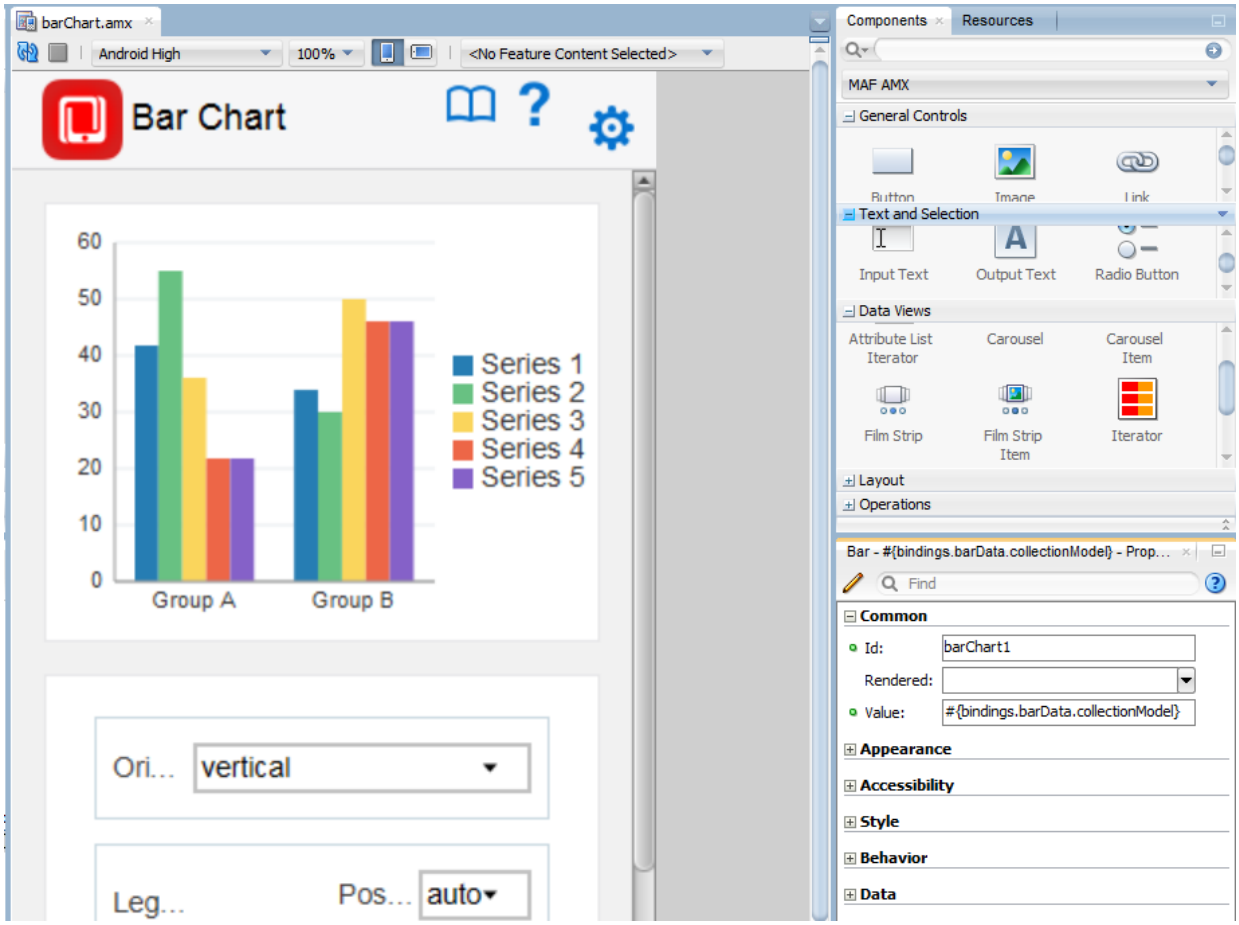
```
<!-- Feature Reference ID in maf-application.xml File -->
<adfmf:featureReference id="fr2" refId="People"/>
...
<!-- Feature ID in maf-feature.xml File -->
<adfmf:feature id="People" name="People" icon="images/people.png" image="images/people.png">
...
```

2.6 Creating MAF AMX Pages and MAF Task Flows

Construct the user interface of an application with MAF AMX pages that can be used by a feature or the whole application. Navigate between AMX pages with MAF task flows that organize MAF AMX view pages and other activities into sequences.

As described in 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 2-5 illustrates the declarative development of a MAF AMX page.

Figure 2-5 Creating a 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 a 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 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. A 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 Creating Task Flows, a 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 default *List* page of the WorkBetter sample application and the *Detail* page in Figure 2-6) 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 2-6, 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 is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

Figure 2-6 MAF Task Flow

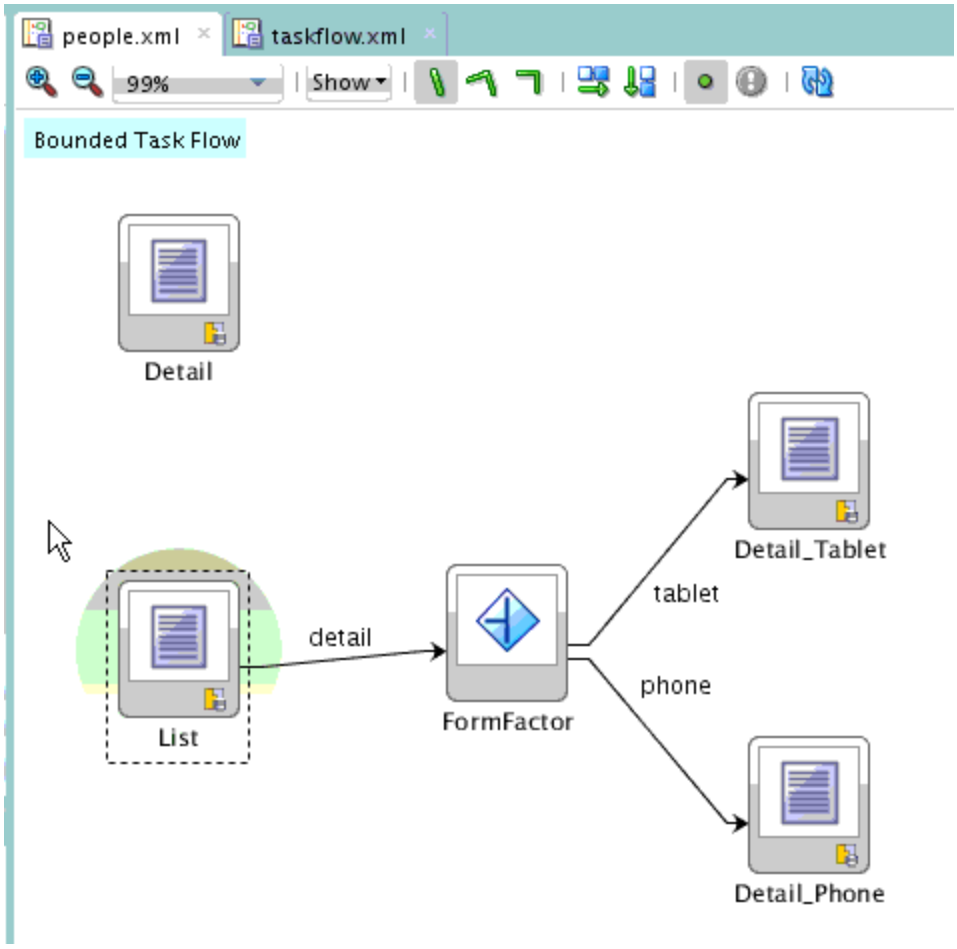
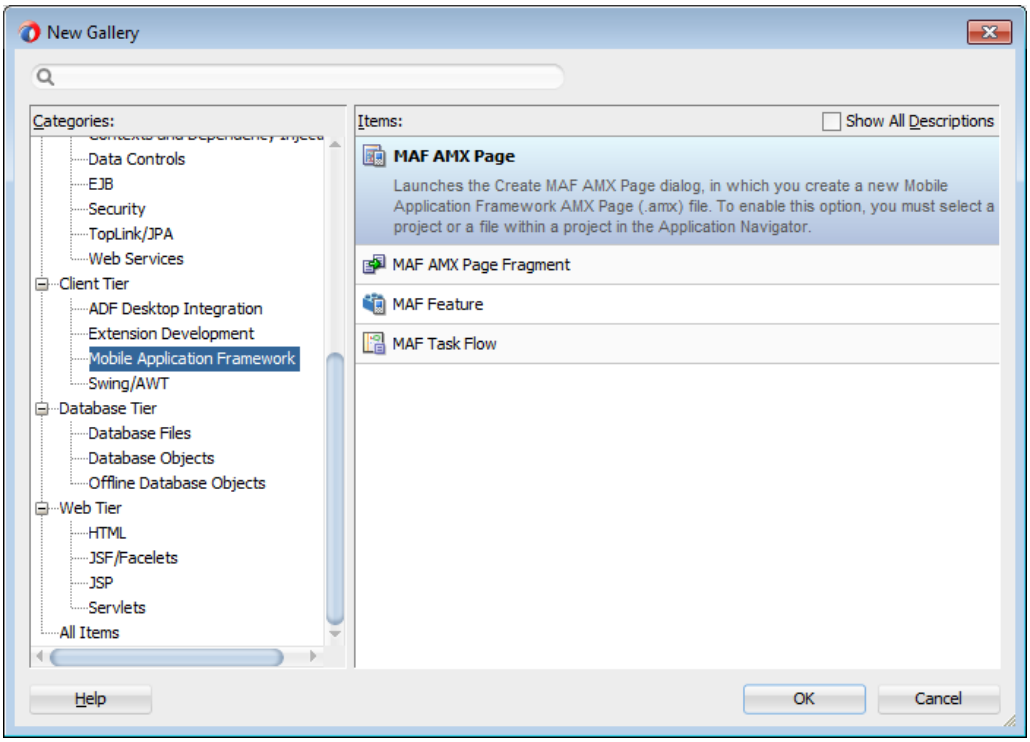


Figure 2-7 shows wizards that MAF provides to add MAF task flows, AMX pages, reusable portions of MAF AMX pages called MAF page fragments, and application features. To access these wizards, select a view controller or application controller project within the Applications window and select **File** > **New**. Select one of the wizards after selecting **Mobile Application Framework** within the **Client Tier**.

Figure 2-7 Wizards for Creating Resources for Application Features



2.6.1 How to Create a MAF AMX Page

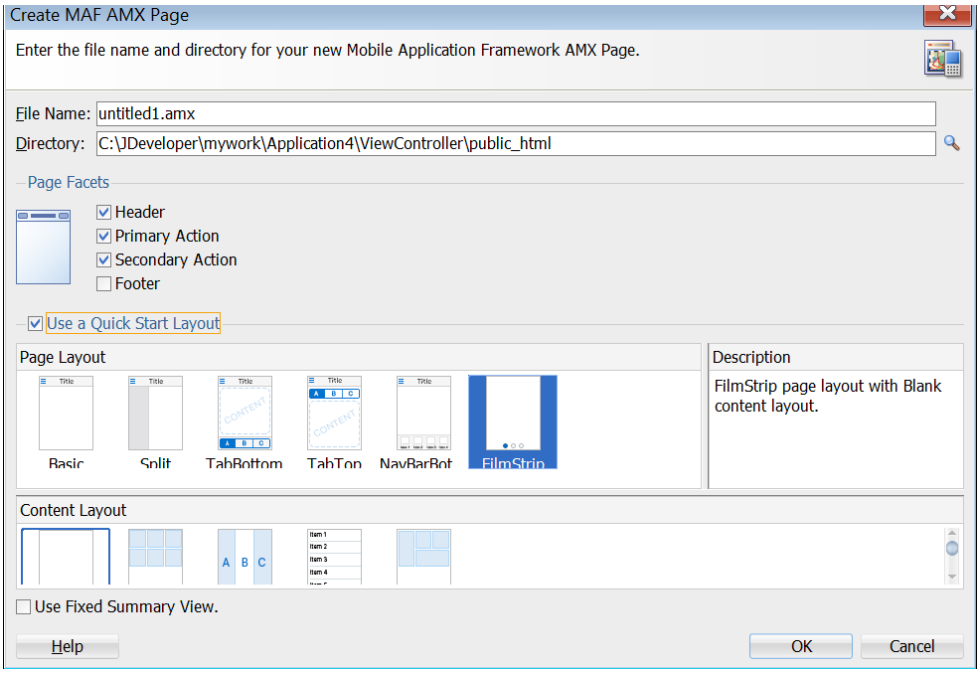
You can use the MAF AMX Page wizard to create AMX pages used for the user interface for an application feature, or as an application-level resource (such as a login page) that can be shared by the application features that comprise the MAF application.

For information about application feature content, see Defining the Content Type of MAF Application Features .

To create a MAF AMX page as content for an application feature:

1. In the Applications window, right-click the view controller project.
2. Select **File** and then **New**.
3. From the Client Tier node in the New Gallery, select **MAF AMX Page** and then click **OK**.
4. Complete the **Create MAF AMX Page** dialog, shown in Figure 2-8, 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 view controller project.

Figure 2-8 Create MAF AMX Page Dialog



5. Select (or deselect) the **Facets** within the Panel Page that are used to create a header and footer. Click **OK**.

   See How to Use a Panel Page Component.

6. Select the **Use a Quick Start Layout** checkbox to enable page layout and the associated content layout options.

   The options in the Content Layout list depend on the Page Layout option you select. MAF generates an AMX page using the layout options you select.

7. Select the **Select Use Fixed Summary View** checkbox to include a Summary Section in the generated AMX page.

   Entries, similar to the following, appear in the generated AMX page to render this section.

```
<amx:facet name="top">
        <amx:panelGroupLayout styleClass="summary-section-style"
layout="horizontal" halign="center" id="pgl1">
          <amx:outputText value="Summary Section" id="ot11"/>
        </amx:panelGroupLayout>
      </amx:facet>
```

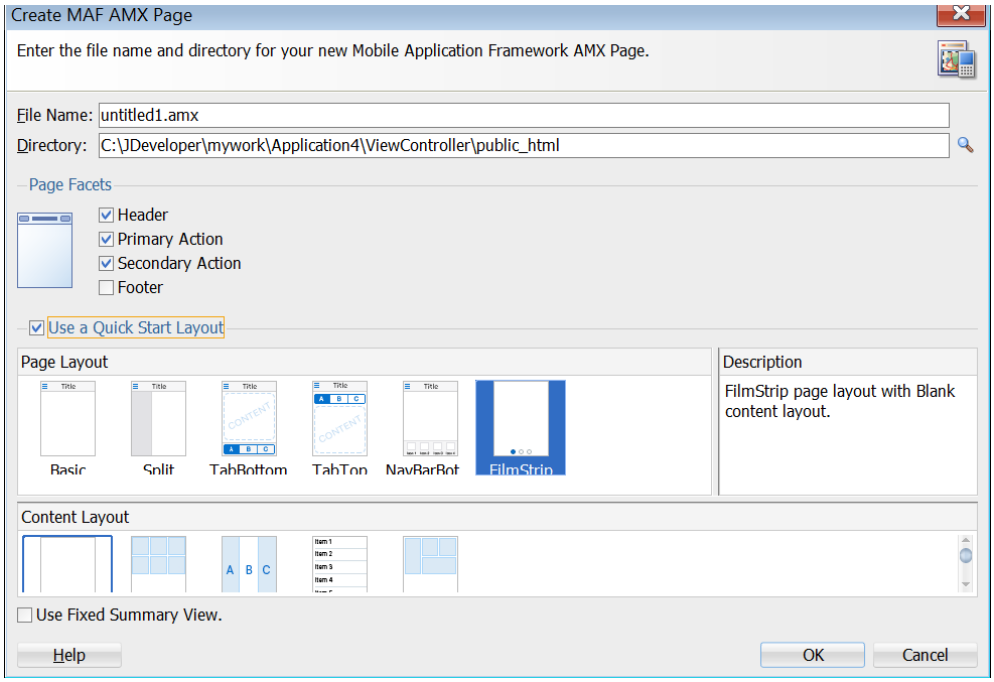
8. Build the MAF AMX page. For information about using the AMX components, see Creating MAF AMX Pages. See also Defining the Application Feature Content as a MAF AMX Page or Task Flow.

To create a MAF AMX page as a resource to a MAF application:

1. In the Applications window, select the application controller project.
2. Select **File** and then **New**.
3. From the **Client Tier** node in the **New Gallery**, select **MAF AMX Page**, and then click **OK**.

4. Complete the **Create MAF AMX Page** dialog, shown in Figure 2-9, 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**.

Figure 2-9 Create MAF AMX Page Dialog



5. Build the MAF AMX page. See Creating MAF AMX Pages.

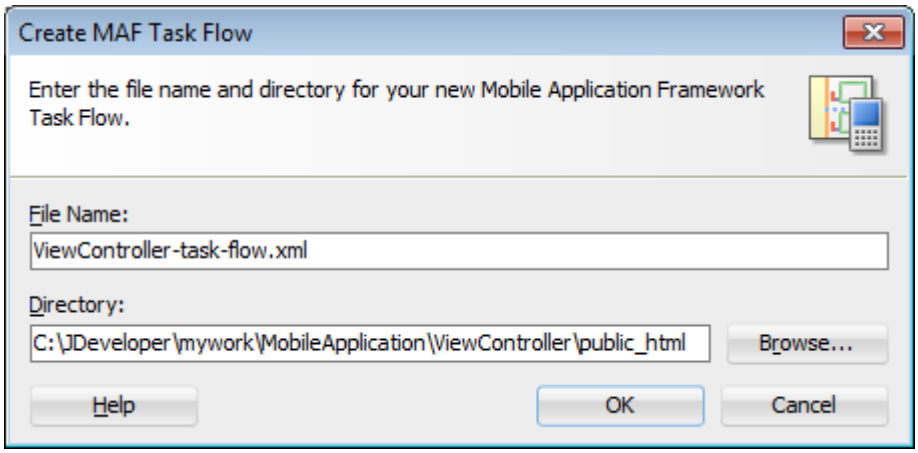
2.6.2 How to Create MAF Task Flows

The content for an application feature can be delivered as a MAF task flow. Use the procedure to access and use the Create MAF Task Flow window to define a task flow for an application feature.

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

1. In the **Applications** window, select the view controller project.
2. Select **File** and then **New**.
3. From the **Client Tier** node in the **New Gallery** select **MAF Task Flow** and then click **OK**.
4. Complete the **Create MAF Task Flow** 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 view controller project. Click **OK**.

Figure 2-10 Creating a MAF Task Flow in a View Controller Project



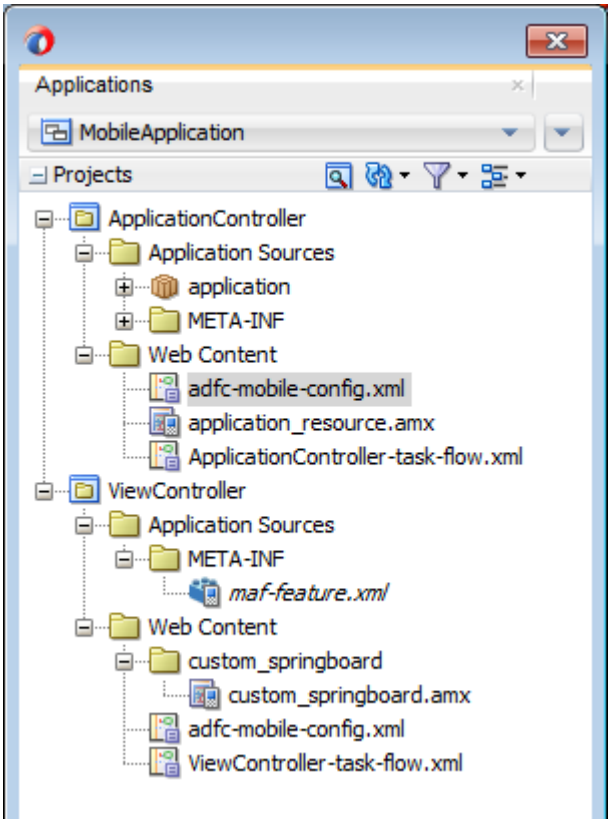
5. Build the task flow. See also Creating Task Flows.

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

When you create MAF AMX pages and task flows, JDeveloper adds them to the web content of the application controller and the view controller projects. JDeveloper generates the `adfc-mobileconfig.xml` file which you can use to create or update a task flow.

JDeveloper places the MAF AMX pages and task flows in the **Web Content** node of the view controller project, as shown by `custom_springboard.amx` and `ViewController-task-flow.xml` (the default name for a task flow created within this project) in Figure 2-11. These artifacts are referenced in the `maf-feature.xml` file. To manage the unbounded task flows, JDeveloper 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 2-11 MAF AMX Pages and Task Flows within Application Controller and View Controller Projects



JDeveloper places the MAF AMX page and task flow as application resources to the MAF application in the **Web Content** node of the application controller project. As illustrated in Figure 2-11, 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).

3

Configuring the Content of a MAF Application

This chapter describes how you configure the `maf-application.xml` and `maf-features.xml` files to define information such as the application name and application features to include for your MAF application.

This chapter includes the following sections:

- Introduction to Configuring MAF Application Display Information
- Setting Display Properties for a MAF Application
- Changing the Launch Screen for Your MAF Application on iOS
- 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 overview editor of the `maf-application.xml` file.

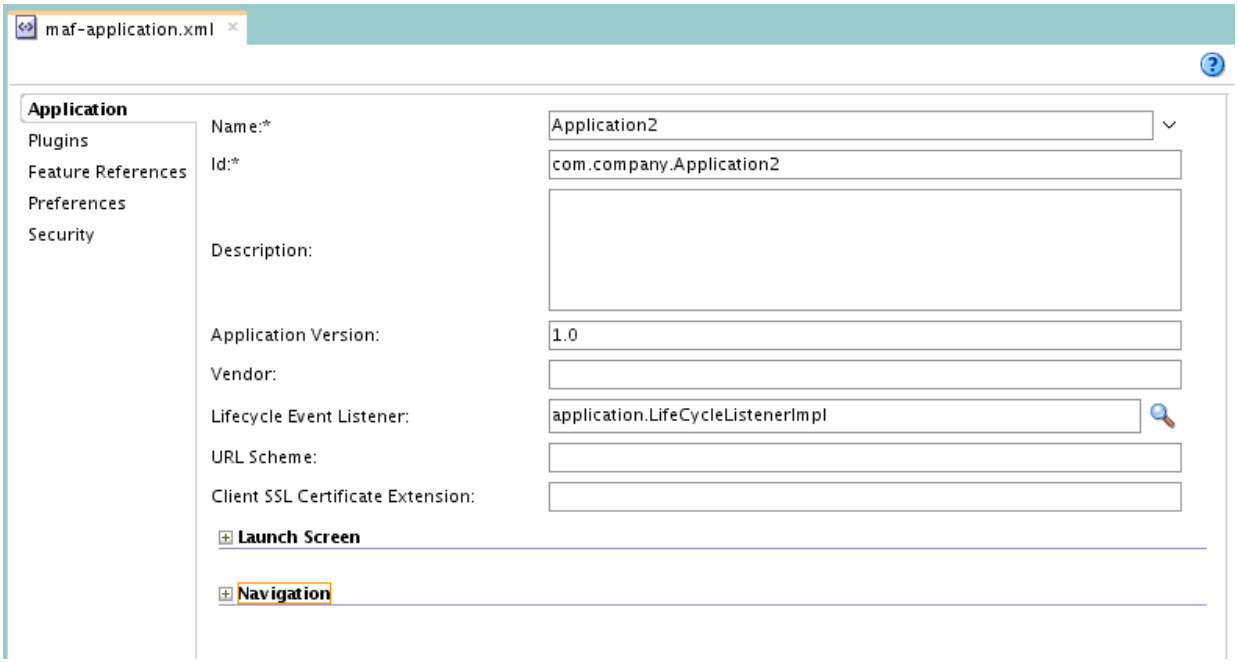
Examples of the type of information you enter for the application include the display name, a description of your application, and the application's version number. You can enter similar information for individual application features that you include in your MAF application or distribute for use in 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

You can set the display properties for a MAF application using the Application page.

Figure 3-1 shows the **Application** page of the `maf-application.xml` file's overview editor where you set the display name and application ID of your MAF application.

Figure 3-1 Setting the Basic Information for the MAF Application



To set the basic information for a MAF application:

1. Choose the **Application** page.
2. In the **Applications** window, expand the Application Resources panel.
3. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
4. Double-click the **maf-application.xml** file and in the overview editor that appears, click the **Application** navigation tab.
5. Enter a display name for the application in the **Name** field.

   You can select a value from a resource bundle if you intend to localize your application.

   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.

   ---

6. Enter a unique ID in the **Id** field.

   To avoid naming conflicts, Android and iOS use reverse package names, such as *com.company.application*. JDeveloper 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 Deploying a MAF Application to the Android Platform and How to Create an iOS Deployment Profile.

Note:

To make sure 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*) fails to deploy. An ID that begins with letters, such as hello925090 (*com.company.hello925090*) deploys successfully.

---

7. In the **Description** field, enter text that describes the application.
8. Enter the version in the **Version** field.
9. Enter the name of the vendor who originated this application in the **Vendor** field.
10. 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 information about Lifecycle Listeners, see Using 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 Application page of the `maf-application.xml` file's overview editor. The HTML page you create is saved in the `ApplicationController/public_html` 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 `ApplicationController/public_html` 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/
```

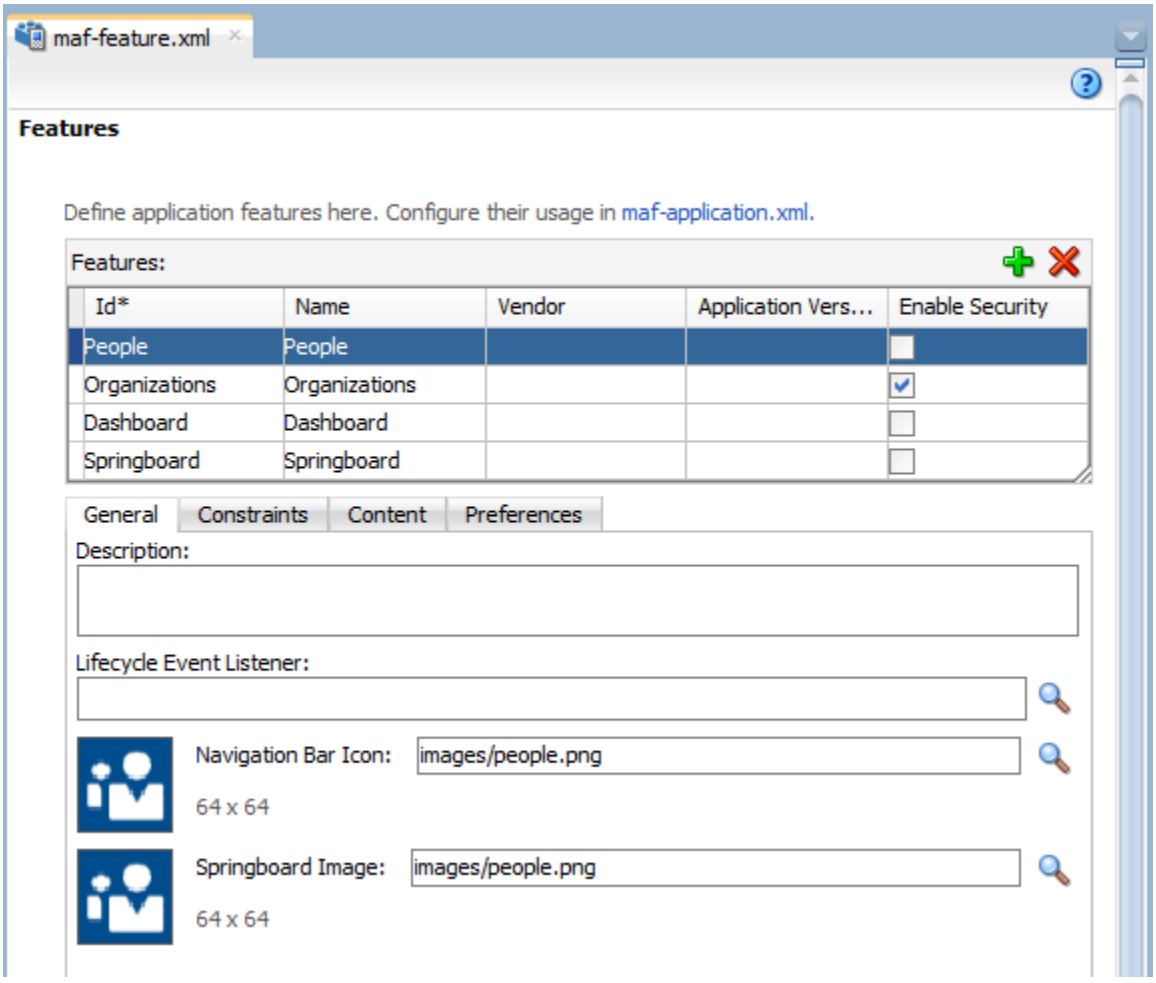
3.4 Setting Display Properties for an Application Feature

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 overview editor for the `maf-feature.xml` file enables you to define the child elements of `<adfmf:features>` 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 3-2 shows the **General** tab of the overview editor for the People application feature in 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 3-2 General Tab for Application Feature in maf-feature.xml File



Before you begin:

If an application feature uses custom images for the navigation bar and springboard rather than the default ones provided by MAF, you must create these images to the

specifications described by the Android Developers website (`http://developer.android.com/design/style/iconography.html`) and in the "Custom Icon and Image Creation Guidelines" chapter in *iOS Human Interface Guidelines*, which is available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

You place these images in the view controller project's `public_html` directory. See also What You May Need to Know About Selecting External Resources .

In addition, you must open the `maf-feature.xml` file and select the General tab.

To set the basic information for the application feature:

1. Choose the **General** tab.
2. Click the **Add** icon in the **Features** section.
3. Complete the **Create MAF Feature** dialog and click **OK**.

   To complete the **Create MAF Feature** dialog:

   - Enter a display name for the application feature in the **Feature Name** field.
   - Enter a unique identifier for the application feature in the **Feature ID** field.
   - If needed, change the location for the application feature to any directory within the `public_html` directory (the default parent directory). Enter this location in the **Directory** field.
   - Select the **Add a corresponding feature reference to maf-application.xml** checkbox to include the newly defined application feature in the MAF application.
4. (Optional) In the **General** tab of the overview editor, enter the originator of the application feature in the **Vendor** field.
5. (Optional) Enter the version number of the application feature in the **Version** field.
6. (Optional) Enter a brief description of the application's purpose in the **Description** field.
7. (Optional) Enter the fully qualified class name (including the package, such as `oracle.adfmf.feature`) using the Class and Package Browser in the **Lifecycle Event Listener** field to enable runtime calls for start, stop, hibernate, and return to hibernate events.
8. (Optional) In the **Navigation Bar Icon** and **Springboard Image** fields, browse to, and select, images from the project to use as the icon in the navigation bar and also an image used for the display icon in the springboard. You can also drag and drop the image files from the **Applications** window into the file location field.

4

Configuring the Application Navigation

This chapter describes how to configure application navigation in a MAF application.

This chapter includes the following sections:

- Introduction to the Display Behavior of MAF Applications
- Configuring Application Navigation
- What Happens When You Configure the Navigation Options
- What Happens When You Set the Animation for the Springboard
- What You May Need to Know About Custom Springboard Application Features with HTML Content
- What You May Need to Know About Custom Springboard Application Features with MAF AMX Content
- What You May Need to Know About the Runtime Springboard Behavior
- Restarting an Application Feature in a MAF Application
- Navigating a MAF Application Using Android's Back Button
- Creating a Sliding Window in a MAF Application
- Using Custom URL Schemes in MAF Applications

4.1 Introduction to the Display Behavior of MAF Applications

You can configure the MAF application to control the display behavior of the springboard and the navigation bar.

You can configure in the following ways:

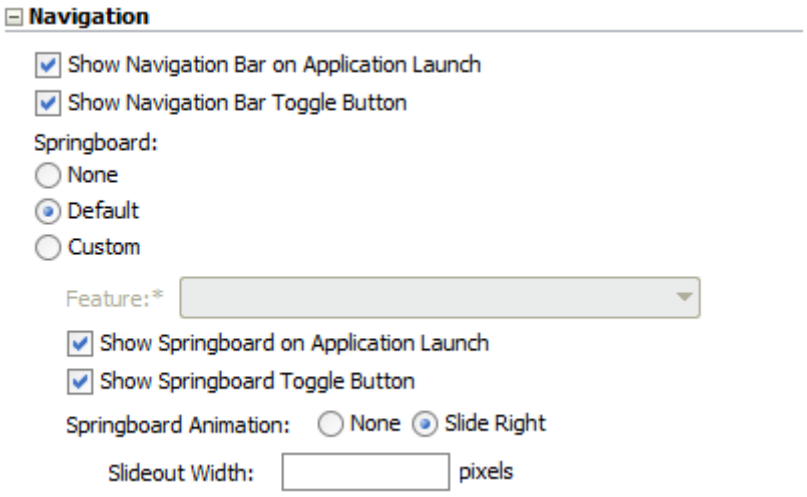
- Hide or show the springboard and navigation bar to enable the optimal usage of the mobile device's interface. These options override the default display behavior for the navigation bar, which is shown by default unless specified by the application feature.
- Enable the springboard to slide from the 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

You can configure the navigation bar of an application using the Navigation options that appear in the Applications page of the `maf-application.xml` file's overview editor.

The **Navigation** options of the **Applications** page, shown in Figure 4-1, 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

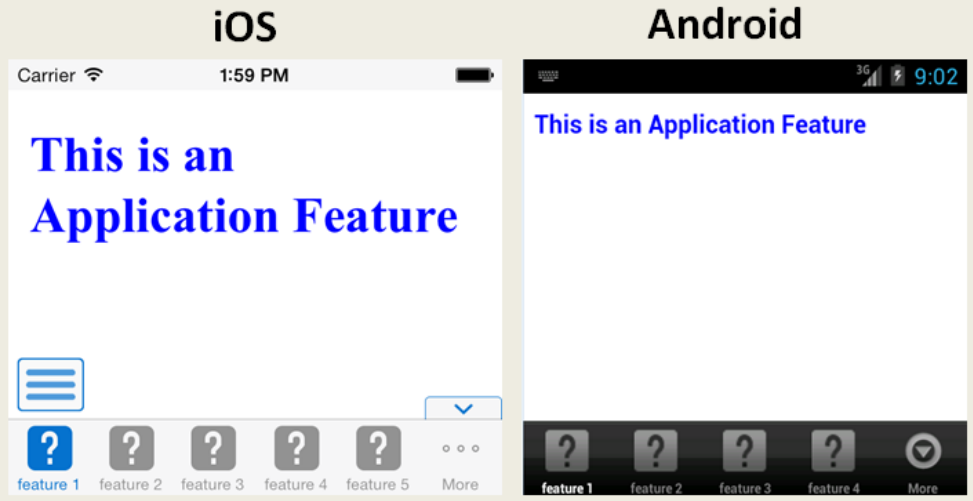
The default behavior for a MAF application is to show the navigation bar on application launch.

You can change this default behavior in the Application page of the `maf-application.xml` file's overview editor.

To set the display behavior for the navigation bar:

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

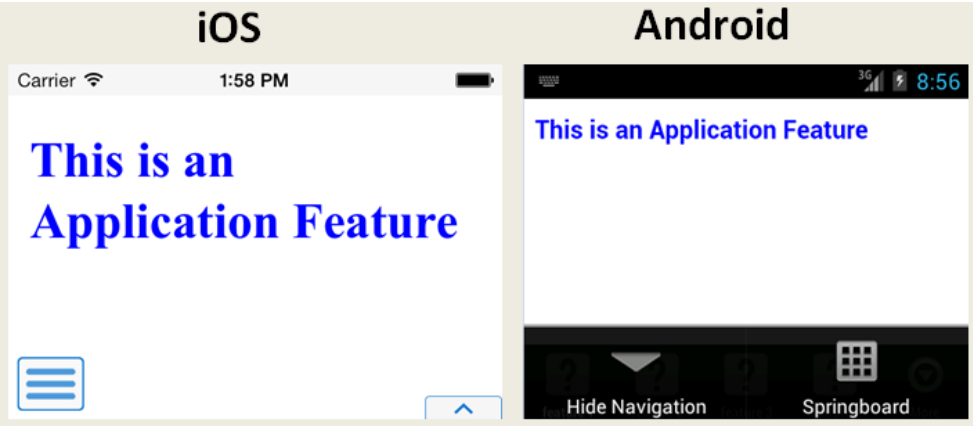
Figure 4-2 The Navigation Bar, Shown By Default



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 Navigation Bar 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 **Application** page of the `maf-application.xml` file's overview editor.

To set the display behavior for the springboard:

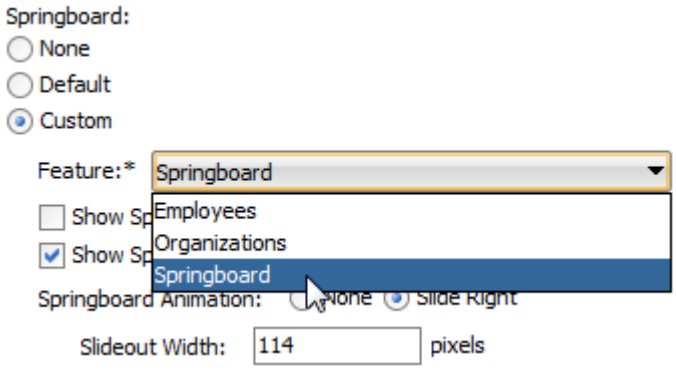
1. Select the type of springboard (if any):

- **None**—Select this option if the springboard should not be displayed in the application.
- **Default**—Select to display the default springboard provided by MAF. The default springboard is implemented as a **MAF AMX** page. For more information, see What You May Need to Know About Custom Springboard Application Features with MAF AMX Content.
- **Custom**—Select to use a customized springboard. You may, for example, create a custom springboard that arranges the embedded application features in a grid layout pattern, or includes a search function, or data, such as a list of common tasks (*My Reports*, or *My Leads*, for example). This application, which can be implemented either as an HTML page or as a **MAF AMX** page, is declared as an application feature in the `maf-feature.xml` file (which is located within a view controller project). For more information, see Setting Display Properties for an Application Feature. For information on enabling navigation within a customized springboard written in HTML, see Local HTML and Application Container APIs.
  - **Feature**—Select the application feature used as a springboard, as shown in Figure 4-4.

Note:

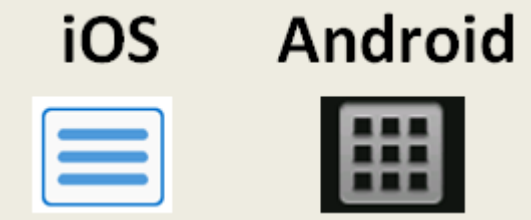
MAF's design time prompts you to set the **Show on Navigation Bar** and **Show on Springboard** options to `false` when you designate an application feature as a custom springboard. This makes sure that the page behaves as a custom springboard rather than as an application feature that users launch from a navigation bar or from a springboard.

Figure 4-4 Selecting an Application Feature as a Custom Springboard



2. Select **Show Springboard on Application Launch** to enable the MAF application to display the springboard to the end user after the MAF application has been launched. (This option is only available for the **Default** or **Custom** options.)
3. Select **Show Springboard Toggle Button** to enable the display of the springboard button, shown in Figure 4-5, that displays within an application feature. Figure 4-2 shows this button within the context of an application feature. This option is only available for the **Default** or **Custom** options.

Figure 4-5 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 **Application** page of the `maf-application.xml` file's overview editor.

To set the slideout behavior for the springboard:

1. Select **Springboard Animation** and then choose **Slide Right**. The springboard occupies an area determined by the number of pixels (or the percent) entered for the **Slideout Width** option. If you select **None**, 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.

   Note:

   The slideout option is only applicable when you select either the **Custom** or **Default** springboard options.

   ---

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 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. Click the **Feature References** page of the `maf-application.xml` overview editor.

2. Use the up- and down-arrows shown to arrange the display order of the feature references, or use the drop-down list in rows of the **Feature Id** column to reorder the feature references. 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.

Note:

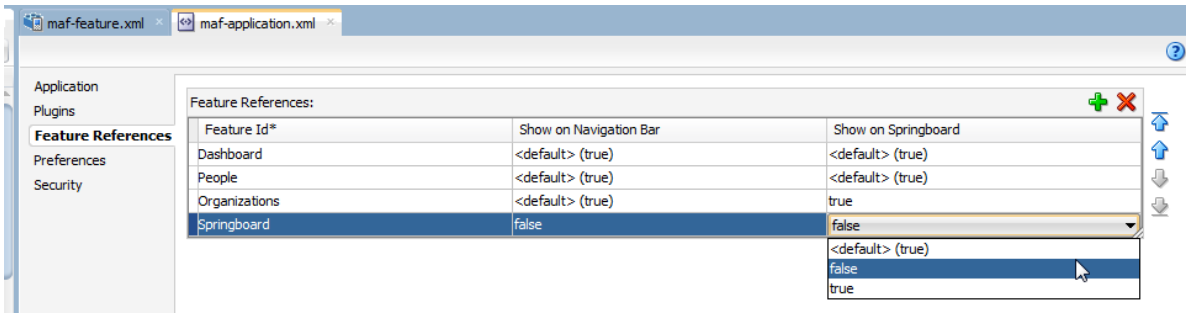
The top-most Id in the **Feature References** table is the first application feature to display within the MAF application. See, for example, the Dashboard application feature in the WorkBetter sample application.

3. Set the springboard and navigation bar display behavior for the application feature by selecting `true` or `false` from the drop-down lists in the rows of the **Show on Navigation Bar** and **Show on Springboard** columns. Figure 4-6 shows selecting these options to prevent an application feature from displaying in the navigation bar.

   Tip:

   Set these options to `false` if the application uses a custom springboard or if the application feature displays as a sliding window.

Figure 4-6 Changing the Navigation Options



The springboard and the navigation bar display by default (that is, these attributes are set to `true`). If both the navigation bar and springboard attributes are set to `false`, 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 set to `false` 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 `maf-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>
```

Example 4-1 illustrates how the `enabled` attribute is set to `true` when you select **Default**.

Note:

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 Example 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

You can set the animation for the springboard, where the springboard is set to slide out and occupy a specified area of the display (213 pixels)

Example 4-3 shows the navigation block of the `maf-application.xml` file.

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, Example 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

You can customize springboard application features to include in a customized login page.

The default HTML springboard page provided by MAF uses the following technologies:

- CSS—Defines the colors and layout.
- JavaScript—The `<script>` tag embedded within the springboard page contains references to the methods described in 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`
- 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 controller project which also contains the `maf-feature.xml` file.

The default springboard (`adfmf.default.springboard.jar`, located in *`jdev_install`*`\jdeveloper\jdev\extensions\oracle.maf\lib`) is a MAF AMX page that is bundled in a Feature Archive (FAR) JAR file and deployed with other FARs that are included in the MAF application. This JAR file includes all of the artifacts associated with a springboard, such as the `DataBindings.cpx` and `PageDef.xml` files. This file is only available after you select **Default** as the springboard option in the `maf-application.xml` file. Selecting this option also adds this FAR to the application classpath. For information about FAR, see Deploying Feature Archive Files (FARs).

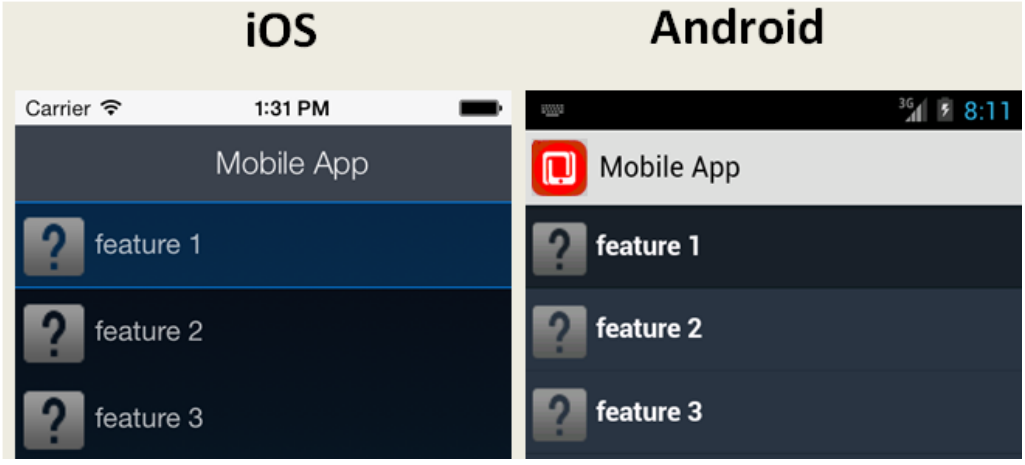
The default springboard (`springboard.amx`, illustrated in the following example) is implemented as a MAF AMX application feature.

```
<?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}"
                    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>
```

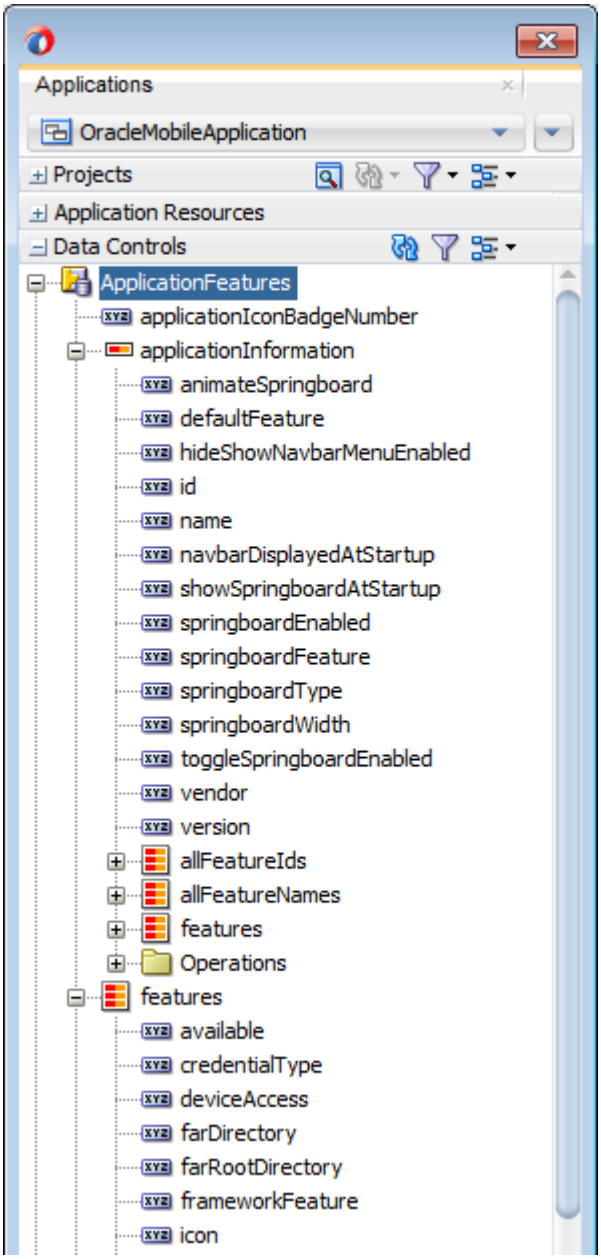
As shown in Figure 4-7, a MAF AMX file defines the springboard using a List View whose List Items are the MAF application's embedded application features. These application features, once deployed, are displayed by their names and associated icons. The `gotoFeature` method of the `AdfmfContainerUtilities` API provides the page's navigation functions. For a description of using this method to display a specific application feature, see gotoFeature. See also How to Use List View and List Item Components.

Figure 4-7 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 4-8, 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 information about this application (and other samples that ship with MAF), see MAF Sample Applications.

Figure 4-8 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 information about data controls, see Using Bindings and Creating Data Controls in MAF AMX . For 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 Springboard 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 Restarting an Application Feature in a MAF Application

MAF provides an AMX action and APIs that you can use to restart an application feature that has a content type of MAF AMX Page or MAF Task Flow (MAF AMX application feature).

Restarting a MAF AMX application feature returns end users to the start of an application feature. In the case of an application feature with task flows, this restarts the task flow.

You can restart an application feature in one of two ways:

- Expose a UI command component, for example, a button that invokes the start action (`amxCommandButton action="__start"`).
- Write code that invokes one of the methods exposed by the `AdfmfContainerUtilities` class. For example, `AdfmfContainerUtilities.queueNavigationRestart("feature1", true);` restarts the `feature1` application feature and navigates the end user to the start of the application feature.

Use an application feature restart as an alternative to a feature reset (`AdfmfContainerUtilities.resetFeature`) as `resetFeature` takes more time

than a restart. An application feature restart, for example, does not log out authenticated end users.

MAF also provides APIs that allow you to execute code before and after the application feature restarts by writing a Java class that implements the following interface: `oracle.maf.api.feature.FeatureLifeCycleListener`. For information about lifecycle listeners, see Using Lifecycle Listeners in MAF Applications .

4.8.1 How to Restart an Application Feature

Configure a command component to invoke the `__start` action or a managed bean method that invokes `AdfmfContainerUtilities.queueNavigationRestart`.

4.8.2 What Happens When You Restart an Application Feature

MAF clears the following state information when you restart an application feature:

- Task flow call stack
- pageFlowScope instances
- Data control contexts and contained data controls and providers
- Application feature's current viewScope
- Application feature's current binding container
- Application feature's current navigation history used for back-button navigation
- History from the feature's web view

The JavaScript cache is cleared in the JavaScript layer. A restart does not reload the JavaScript or Cordova plugins in the web view. So, no cost is incurred from reloading. If you store state in static field of a class, then no classes will get reloaded. In the case of a feature reset, classes get reloaded, but not in the case of a feature restart.

4.9 Navigating a MAF Application Using Android's Back Button

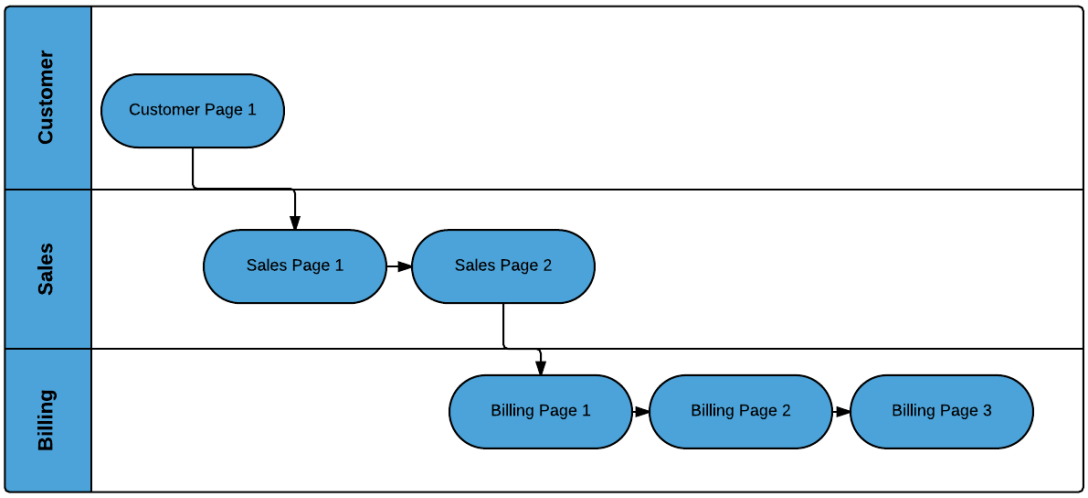
End users can navigate backwards on MAF applications using the Android system's Back button.

Figure 4-9 shows the Android system's Back button that appears on the Android navigation bar or on the Android device itself. Figure 4-9 shows the Android 4.x and Android 5.x versions of the navigation bar where this button appears.

Figure 4-9 Android's Back Button

Figure 4-10 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 4-10 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 information about JavaScript, see *JSDoc Reference for Oracle Mobile Application Framework*.

For information about using the `systemActionBehavior` component, see 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

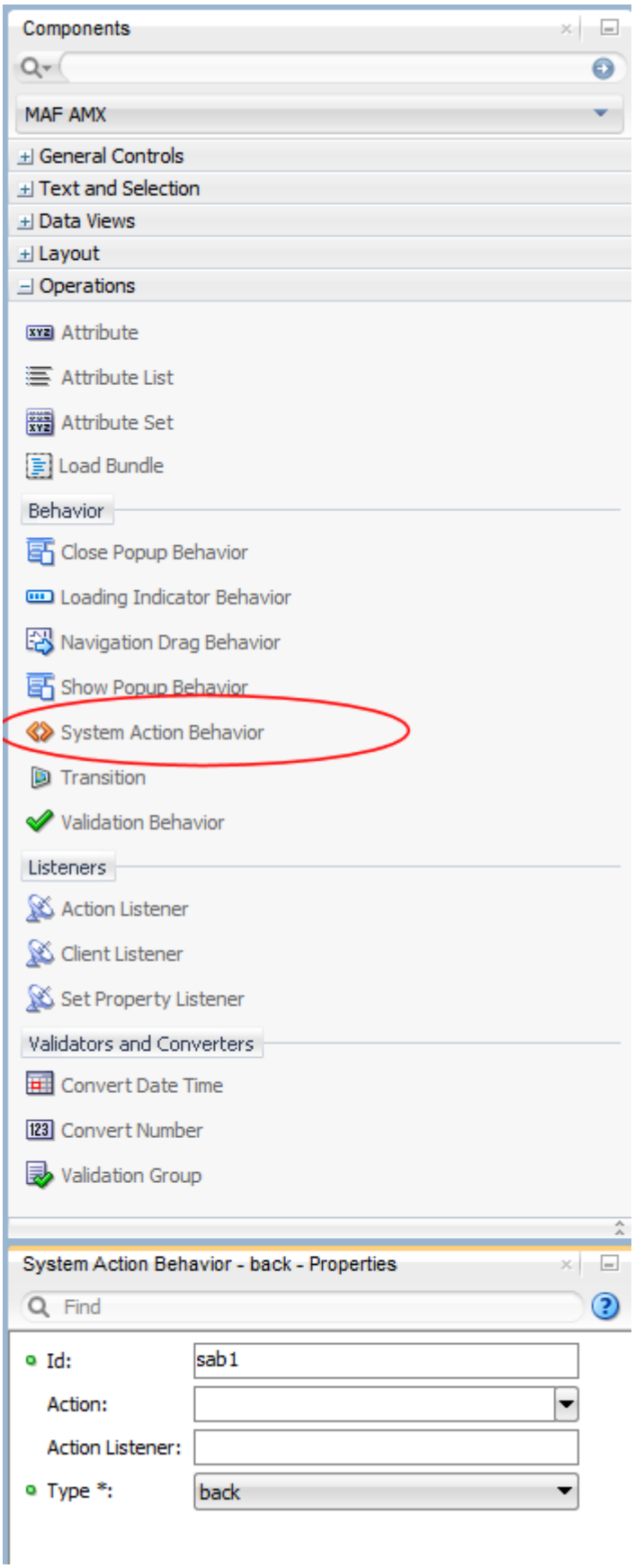
Figure 4-9, an end user navigates from the Billing application feature to Sales application feature and finally Customer application feature before hibernating the MAF application. You can implement this legacy behavior in MAF applications created using this release of MAF by configuring a parameter 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 *Installing Oracle Mobile Application Framework*. Implementing this legacy behavior causes the MAF application to ignore any usage of the `systemActionBehavior` component and the `registerSystemActionOverride` JavaScript method discussed here.

4.9.1 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 JDeveloper, the System Action Behavior is located under **Operations** in the Components window, as Figure 4-11 shows.

Figure 4-11 System Action Behavior in the Components Window



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.

```
public class MyBean {

   public void onBackButton() {
      // do processing
   }

   public String getNavAction() {
      return "";
   }
}
```

In the preceding example, the `getNavAction` method could return the `"__back"` String to enable the back navigation. In this case, the action would be resolved when the MAF AMX page is loaded; it would not be called every time the system back button on the Android-powered device is pressed.

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()
{
```

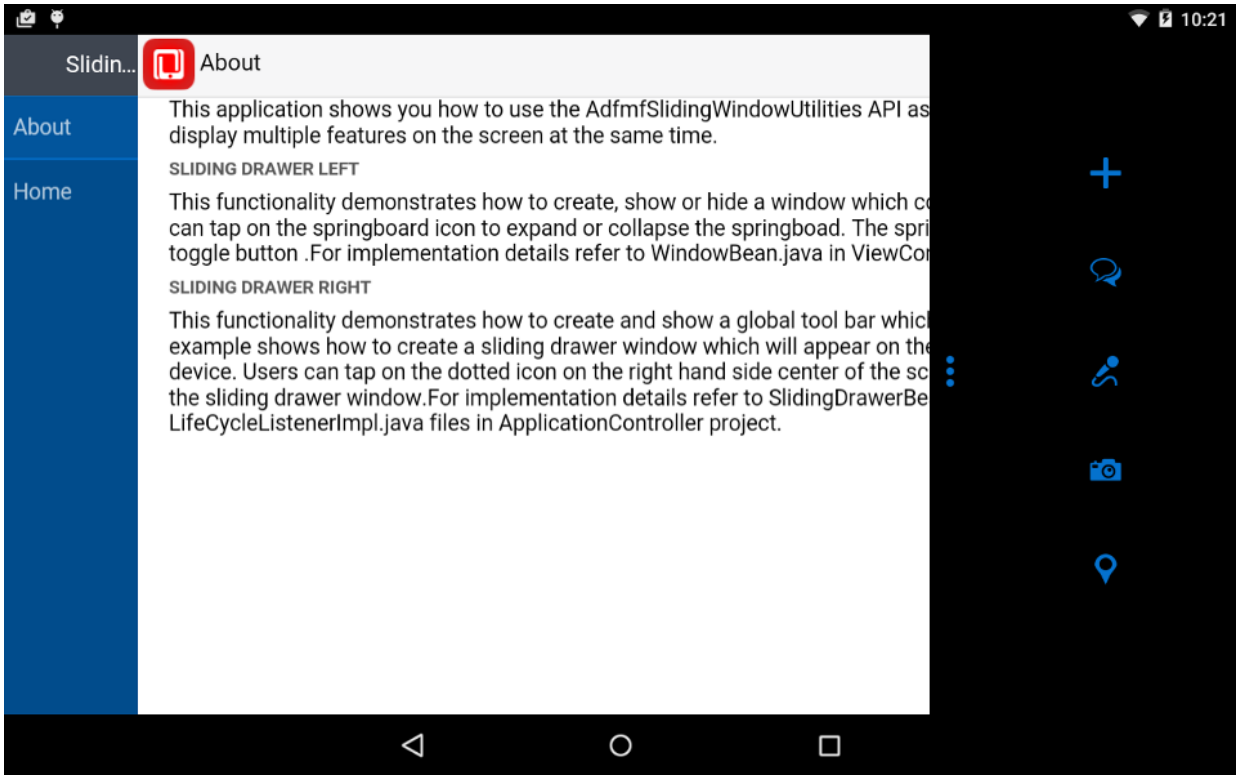
```
   adf.mf.api.amx.doNavigation("__back");
};
adf.mf.api.registerSystemActionOverride("back", handleSystemBack);
```

4.10 Creating a Sliding Window in a 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 4-12 shows the SlidingDrawer application feature from the `SlidingWindow` sample application, described in MAF Sample Applications. This application feature appears on the right of an application screen while overlaying other application features.

Figure 4-12 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 4-12 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 MAF Sample Applications.

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

4.11 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 **Overview** editor of the `maf-application.xml` file 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 application controller 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 information about the `Event` class, see *Java API Reference for Oracle Mobile Application Framework*.

3. Register an application lifecycle event listener (ALCL) class.

For information about lifecycle listerner, see 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.xml` file's overview 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 4-13.

Figure 4-13 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 introduces the content types that you can use in the applications features of your MAF application and describes how to create each supported content type in an application feature.

This chapter includes the following sections:

- Introduction to Content Types for an Application Feature
- Defining the Application Feature Content as Remote URL or Local HTML
- Defining the Application Feature Content as a MAF AMX Page or Task Flow
- Configuring the Web View of Application Features with AMX Content on iOS
- Selecting External Resources for Use in Application Features

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.

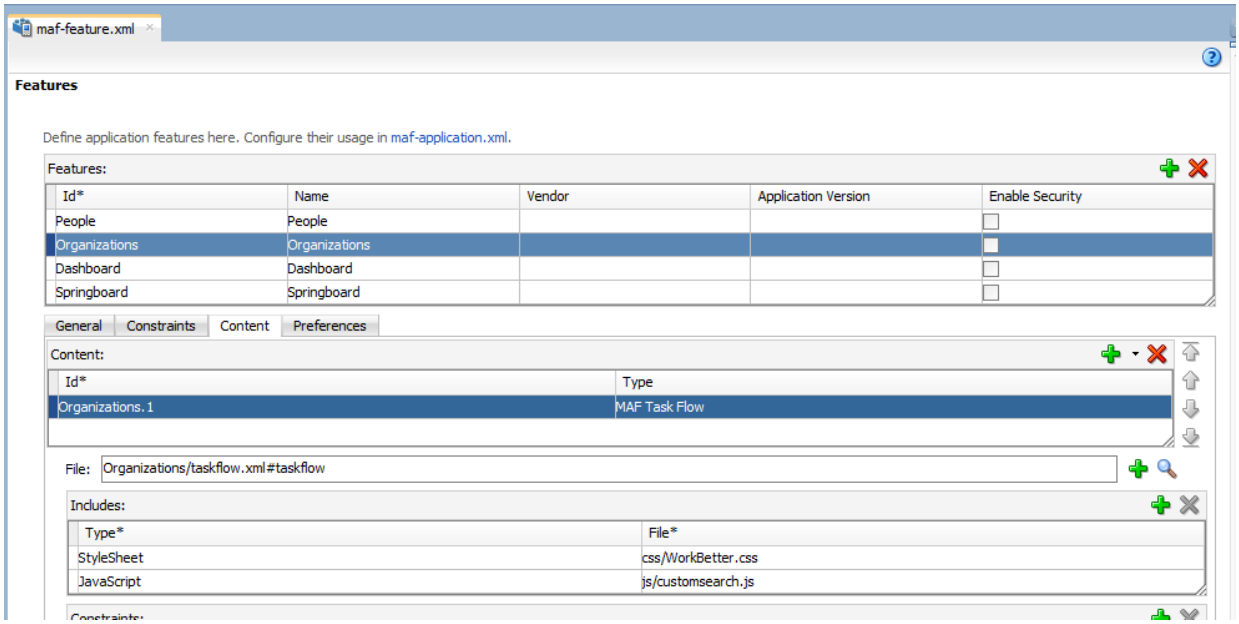
An application feature can also derive its content from remotely hosted pages that contain content appropriate to a mobile context. These web pages might be a JavaServer page authored in Apache Trinidad for smartphones, or be comprised of ADF Faces components for applications that run on tablet devices. 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 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 overview editor, shown in Figure 5-1, provides you with drop-down 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.

Each content type has its own set of parameters. As shown in Figure 5-1, for example, you specify the location of the MAF AMX page or task flow for the application features that you implement as MAF AMX content. In addition, you can optionally select a CSS file to give the application feature a look and feel that is distinct from other application features (or the MAF application itself), or select a JavaScript file that controls the actions of the MAF AMX components.

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 fields within the Content 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.

The **Content** tab of the overview editor, shown in Figure 5-1, provides you with drop-down lists and fields for defining the target content-related elements and attributes shown in Example 5-1.

Before you begin:

Each content type has its own prerequisites, as follows:

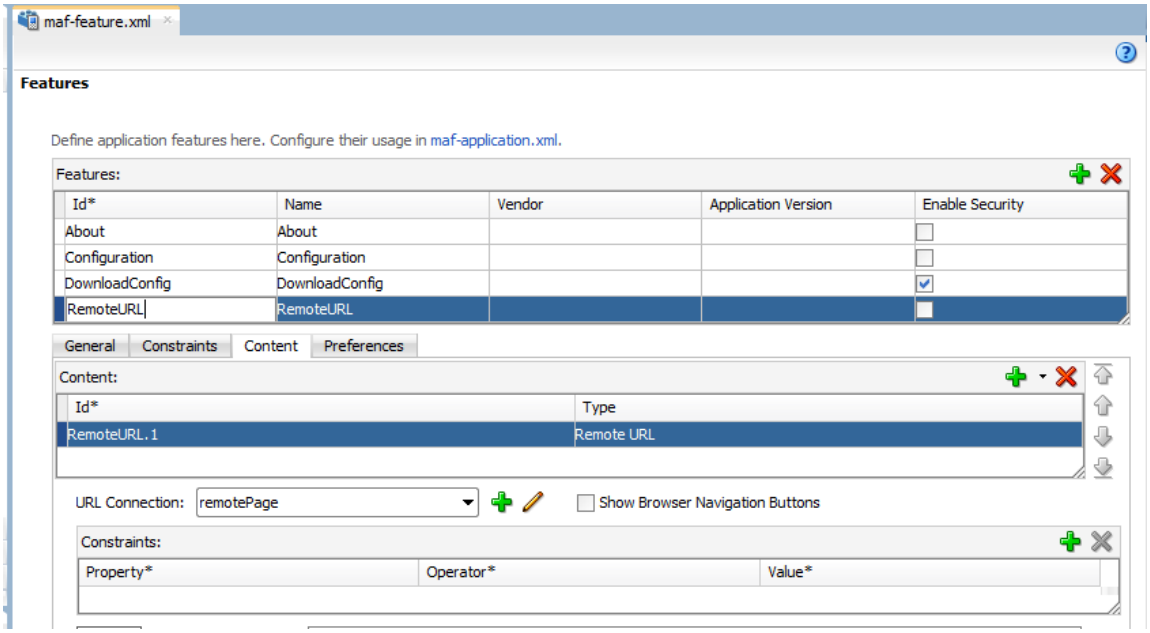
- **Remote URL**—A reference to a web application. You can enhance an existing web application for mobile usage and extend device services. Remote content can complement both MAF AMX and local HTML content by providing a local data cache and a full set of server-side data and functionality. The remote URL implementation requires a valid web address. See 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. See Local HTML and Application Container APIs.

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

1. Select an application feature listed in the **Features** table in the `maf-feature.xml` file.
2. Click **Content**.
3. Click **Add** to create a new row in the **Content** table.
4. Select one of the following content types to correspond with the generated ID:
   - **Remote URL**
   - **Local HTML**
5. Define the content-specific parameters:
   - For remote URL content, select the connection, as shown in Figure 5-2, that represents address of the web pages on the server (and the location of the launch page).

Figure 5-2 Selecting the Connection for the Hosted Application

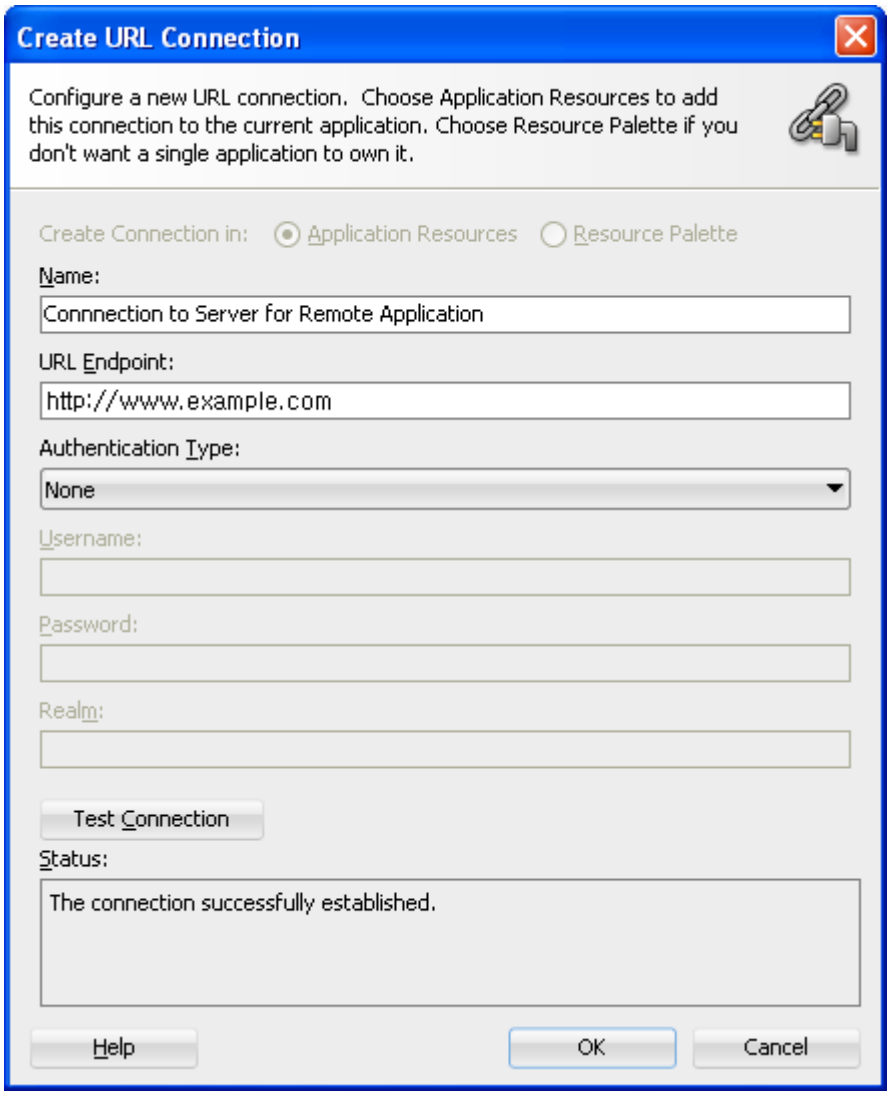


You can create this connection by first clicking **Add** and then completing the **Create URL Connection** dialog, shown in Figure 5-3. For more information on this dialog, see the online help for Oracle JDeveloper. This connection is stored in the `connections.xml` file.

Note:

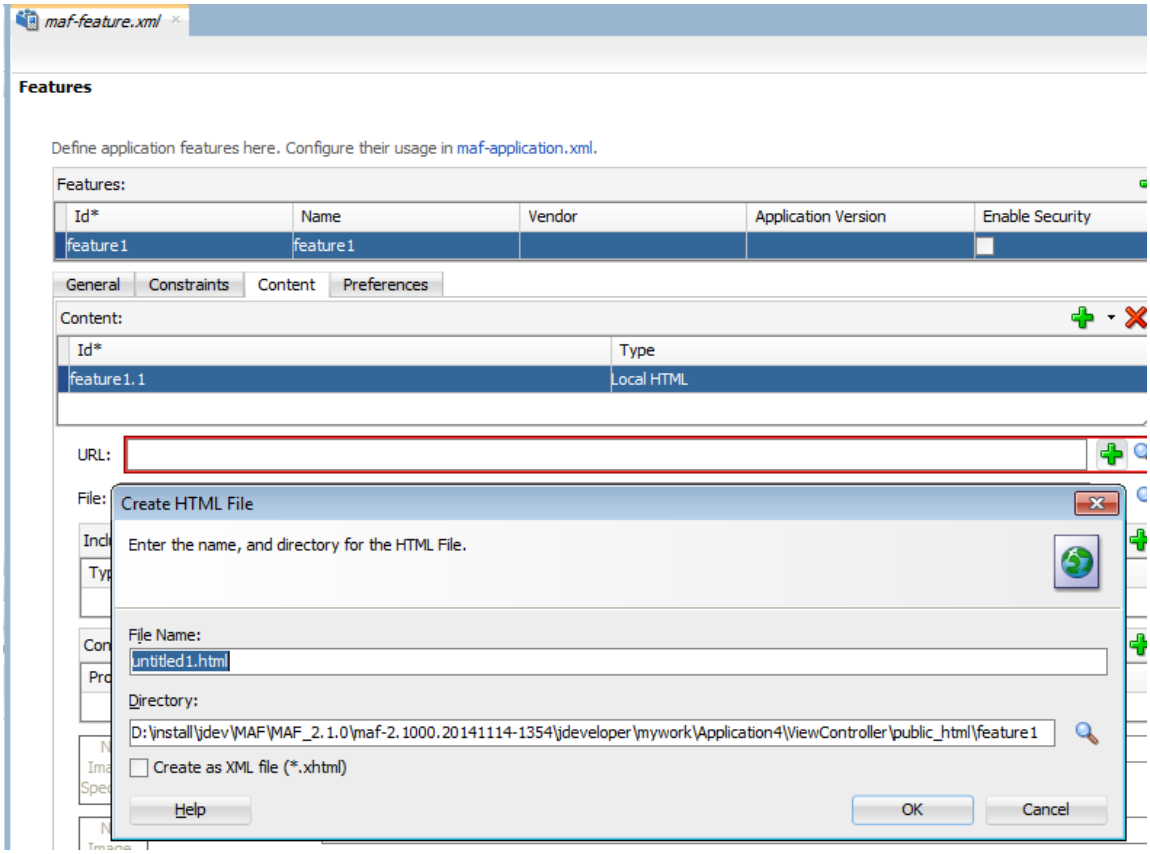
This connection can only be created as an application resource.

Figure 5-3 Creating a URL Connection



- For local HTML content, enter the location of the local bundle or create the HTML page by clicking **Add** in the URL field, completing the dialog as shown in Figure 5-4, and then building the page using JDeveloper's HTML editor. Because this is an application feature, this page is stored within the Web Content folder of the view controller project.

Figure 5-4 Creating the Local HTML Page as the Content for an Application Feature



6. If needed, do the following:

   - Enter constraints that describe the conditions under which this content is available to users. See 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 fields within the Content 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.

The **Content** tab of the **Overview** editor, shown in Figure 5-1, provides you with drop-down lists and fields for defining the target content-related elements and attributes shown in Example 5-1.

Before you begin:

Each content type has its own prerequisites, as follows:

- **MAF AMX**—The default content type for application features. See 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 by first highlighting the view controller project in the **Applications** window and then by choosing **New**.

Note:

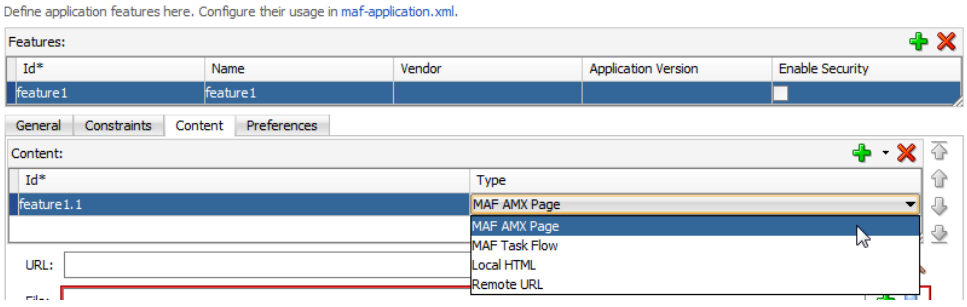
When manually editing references to task flows, MAF AMX pages, CSS and JavaScript files in the `maf-feature.xml` file, keep in mind that file systems used on devices may enforce case-sensitivity and may not allow special characters. To ensure that these files can be referenced, check the mobile device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

- **MAF 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). See Creating Task Flows.

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

1. Select the application feature.
2. Click **Content**.
3. If needed, click **Add** to create a row in the **Content** table and choose **MAF AMX Page** or **MAF Task Flow** from the drop-down list in the **Type** column, as shown in the following figure.

Figure 5-5 Selecting MAF AMX Page or MAF Task Flow as the Content Type



4. In the **File** field, choose the appropriate option:
   - If you have already created a MAF AMX page or task flow, click the **Browse** icon and choose the location of the page or task.
   - If you want to create a new MAF AMX page, click the **Add** icon to invoke a dialog where you can create a new MAF AMX page or task flow.
5. If needed, do the following:

- Enter the JavaScript files by clicking **Add** in the **Includes** table, choose **JavaScript**, and then browse to the location of the file.
- Override the default style sheet designated in `maf-config.xml` by first clicking **Add** and then by choosing **Stylesheet**. Browse to the location of the file.
- Enter the constraints, as described in Setting Constraints on Application Features .
- Select navigation bar and springboard images.

Note:

The images, style sheet, and JavaScript files must reside within the `public_html` folder to enable deployment. See What You May Need to Know About Selecting External Resources .

5.4 Configuring the Web View of Application Features with AMX Content on iOS

MAF applications use WKWebView by default to render AMX content when you deploy to iOS 9 devices.

WKWebView is a newer iOS web view that offers improved performance compared to UIWebView. Application features that use local HTML or remote URL content types use the UIWebView by default as this web view supports the `/~maf.device~/` virtual path to access JavaScript APIs. You can configure application features with local HTML and remote URL content types to use WKWebView if you do not need the `/~maf.device~/` virtual path. You can also configure application features with AMX content to use the older UIWebView, if desired. Configure the `iOSWebView` property in the `maf-features.xml`, as demonstrated by the following examples to make these configuration changes.

```
<adfmf:feature id="WKWebViewExample" name="WKWebViewExample">
      <adfmf:constraints>
          <adfmf:constraint property="device.os" operator="contains" value="iOS"
id="c6"/>
      </adfmf:constraints>
    <adfmf:content id="WKWebViewExample.1">
      <adfmf:amx file="WKWebViewExample/home.amx"/>
    </adfmf:content>
    <adfmf:properties id="wkp1">
      <!-- To use WKWebView, set to modern  -->
      <adfmf:property id="wkp1-1" name="iOSWebView" value="modern" />

      <!-- To use UIWebView, set to legacy    -->
      <!-- name="iOSWebView" value="legacy"   -->


    </adfmf:properties>
  </adfmf:feature>
```

When the `iOSWebView` property is missing or is set to `default` then WKWebView is used for AMX content and UIWebView is used for local HTML and remote URL content types.

WKWebView is only used on iOS 9. UIWebView will always be used on iOS 8.

5.5 Selecting External Resources for Use in Application Features

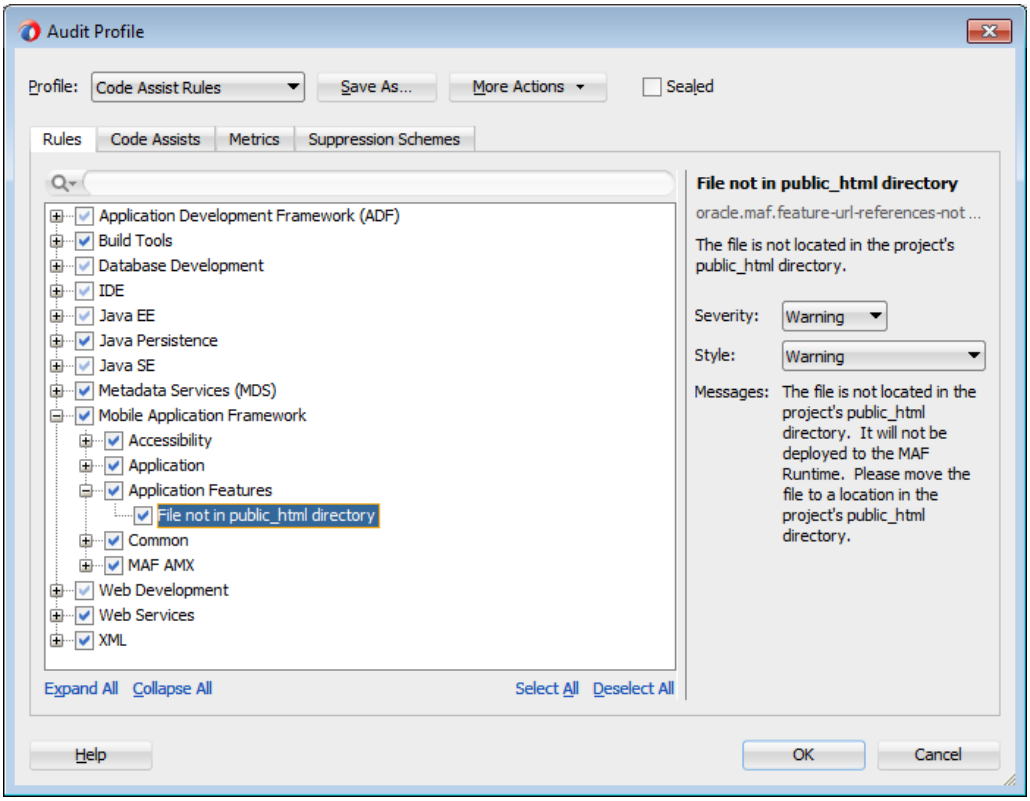
To enable deployment, all resources referenced by the attributes must be located within the `public_html` directory of the view controller project.

All the resources referenced by the following attributes:

- The `icon` and `image` attributes for `<adfmf:feature>` (for example, `<adfmf:feature id="PROD" name="Products" icon="feature_icon.png" image="springboard.png">`).
- The `icon` and `image` attributes for `<adfmf:content>` (for example, `<adfmf:content id="PROD" icon="feature_icon.png" image="springboard_image.png">`).
- The `file` attribute for `<adfmf:amx>` (for example, `<adfmf:amx file="PRODUCT/home.amx" />`).
- The `url` attribute for `<adfmf: localHTML>` (for example, `<adfmf:localHTML url="oracle.hello/index.html"/>`).
- 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"/>`).

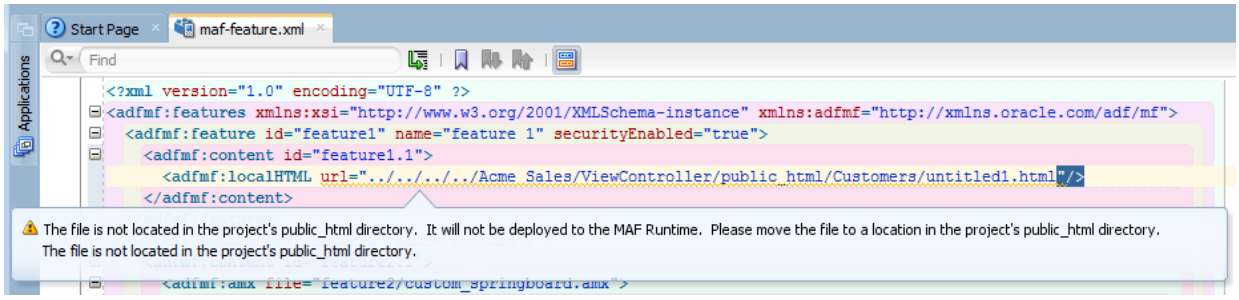
MAF does not support resources referenced from another location, meaning that you cannot, for example, enter a value outside of the `public_html` directory using `../` as a prefix. To safeguard against referencing resources outside of `public_html`, MAF includes an audit rule called *File not in public_html directory*. You can access the MAF audit profiles, shown in Figure 5-6, from the Audit Profiles node in Preferences by choosing **Tools** > **Preferences** > **Audit** > **Profiles**.

Figure 5-6 MAF Audit Profiles



When this profile is selected, JDeveloper issues a warning if you change the location of a resource. As shown in Figure 5-7, JDeveloper displays such a warning when the default values are overridden. For information about auditing, see the Auditing and Monitoring Java Projects chapter in *Developing Applications with Oracle JDeveloper*.

Figure 5-7 The External Resource Warning



6

Creating the Client Data Model in a MAF Application

This chapter describes how to create data and service objects in the client data model of your MAF application by retrieving resources from REST services.

This chapter includes the following sections:

- Introduction to the Client Data Model in a MAF Application
- Overview of Creating a Client Data Model in a MAF Application
- Connecting to a REST Service to Create the Client Data Model
- Discovering Candidate Data Objects for the Client Data Model
- Selecting and Persisting Data Objects for the Client Data Model
- Specifying Parent-Child Relationships for Data Objects
- Defining CRUD REST Resources
- Specifying CRUD REST Resource Details
- Setting Runtime Options for the Client Data Model
- Generating the Client Data Model
- Editing the Client Data Model in a MAF Application
- Accessing the SQLite Database Using the MAF Client Data Model DBPersistenceManager
- Defining a Custom Resource
- Executing Custom Logic After CRUD REST Calls
- Getting Programmatic Access to Service Objects
- Understanding Usage of the Primary Key
- Using Filtered Entity Lists
- Creating a User Interface from a MAF Client Data Model
- Synchronizing Offline Transactions from a MAF Application
- Understanding the Client Data Model's Support for Data Change Events
- Forcing Offline Mode in a MAF Application
- Using a Visual Indicator for Running Background Tasks

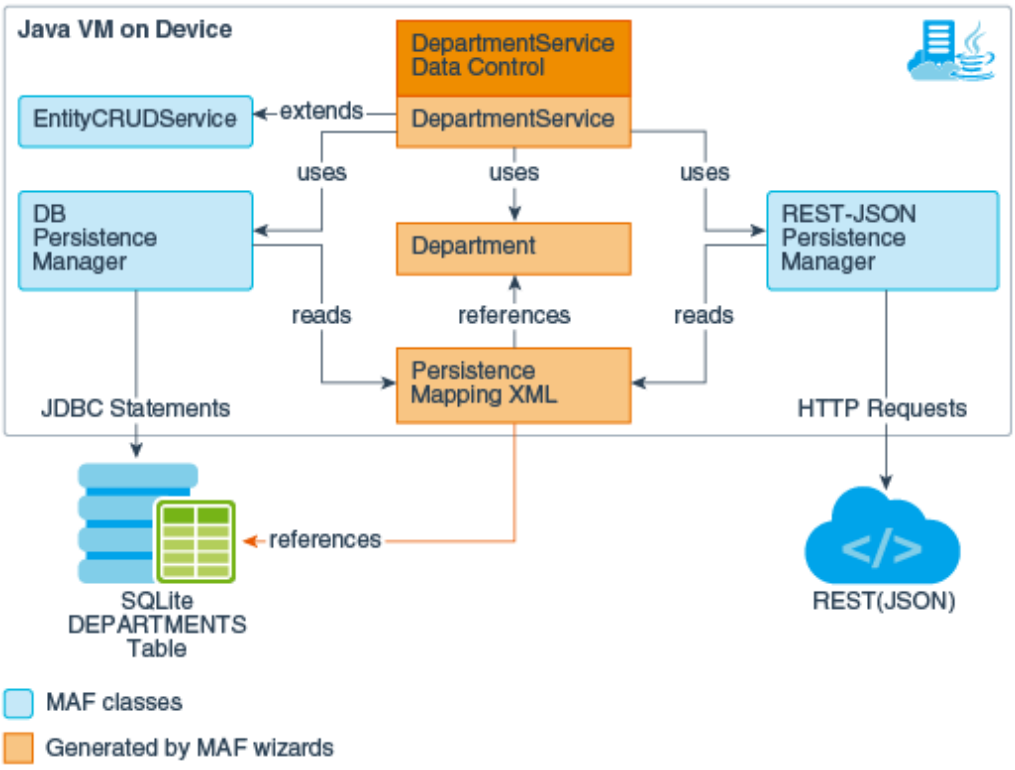
6.1 Introduction to the Client Data Model in a MAF Application

MAF provides design-time support to connect your MAF application to REST services from where you can expose data objects. You can then create a client data model based upon these data objects. MAF also helps you determine what data to persist on the application in offline mode.

The client data model MAF application contains Java classes and associated files to represent the data model of a MAF application. MAF uses a SQLite database to store data for offline usage, and two types of Java class: data objects (also known as entity objects) and service objects (also known as entity CRUD service objects) to interact with the data. Data objects hold the data that you retrieve from the REST service(s) that your MAF application connects to. The MAF application stores and retrieves data objects in a SQLite database on the device. Service objects perform create, read, update, and delete (CRUD) actions plus other custom actions that operate on the data objects. You typically use service objects to create data controls from where you can drag and drop the associated methods and entity collections onto AMX pages to build the UI layer. The MAF client data model uses the `persistence-mapping.xml` file to store the object-relation mapping information that identifies how database tables and columns map to data objects and attributes of data objects plus how data objects and attributes of data objects map to attributes in the REST response payloads.

Figure 6-1 illustrates the runtime architecture of the MAF client data model by reference to a specific implementation in a MAF application that reads and writes department information from a REST service.

Figure 6-1 MAF Client Data Model Runtime Architecture



In Figure 6-1, the service object (`DepartmentService`) provides the CRUD actions plus other custom actions that operate on the `Department` data object.

The `Department` data object has getter and setter methods for the department attributes (name, ID, and so on) that map to the corresponding attributes in the REST service request and response payloads. The `DEPARTMENTS` table in the SQLite database has columns that map to the same data object attributes. The `persistence-mapping.xml` file stores the information that maps the relationship between the attributes in the various locations (database table columns, Java class attributes and REST payload).

MAF provides a number of wizards in JDeveloper that you use to create the client data model in your MAF application. See Overview of Creating a Client Data Model in a MAF Application. Once you have created the client data model, you can generate data controls from the service objects in the client data model. MAF provides a further wizard (the MAF User Interface Generator) that generates a user interface in your MAF application based on the data controls and data model that the MAF client data model generates. See Creating a User Interface from a MAF Client Data Model.

> **Tip:** See Consuming and persisting REST/JSON services with Oracle MAF and the Client Data Model (CDM) for a tutorial that guides you step-by-step through the creation of a MAF application that uses the client data model to access a REST service.

6.2 Overview of Creating a Client Data Model in a MAF Application

Create a client data model from REST services by discovering data objects, configuring their attributes, mapping relationships for them, identifying the REST resources to use for CRUD actions, and generating the artefacts to include in the model.

Using this wizard, you can connect to a generic REST service or to REST services hosted on Oracle Mobile Cloud Service (MCS). Once you connect, MAF displays additional dialogs where you perform tasks to identify and retrieve the data you want to use in your application. These tasks are:

1. Discover the data objects that are candidates for use in your MAF application. MAF supports the discovery of data objects from the following data object resources:
    a. Sample REST resource URLs
    b. Sample resource payloads
    c. RAML files
    d. ADF Business Components REST services
2. Having discovered the candidate data objects for use in your MAF application, MAF presents you with a dialog where you select the data objects that you want to use. MAF also provides an option to create new data objects.
3. Additional dialogs let you inspect and modify data object attributes. Tasks you can perform include edit the attribute name, the name that appears in the REST service payload, the Java type and the database column type for each attribute in addition to choosing not to persist sensitive data on the device. You also select a key attribute. It is important that the key attribute you select be unique.
4. Specifying parent-child relationships for data objects.
5. Define the REST resources and associated HTTP methods to use for CRUD actions plus specify resource details such as the query and path parameters.

6. Once you have identified the REST resources to use for CRUD actions, MAF presents you with dialog where you set the runtime behavior of your MAF application by, for example, enabling offline transactions, enabling remote read and write in the background, or showing web service errors.

7. MAF finally presents a dialog where you specify the location in your application of the artefacts that MAF generates for the client data model. Use this dialog to determine the project location and package name(s) of Java classes that MAF generates.

Once you complete the MAF Client Data Model From REST Web Service wizard, MAF generates a client data model for your MAF application. You can invoke the wizard again to edit the generated client data model by, for example, identifying additional data objects in the REST service that you want to include in the data model. If you want to edit a client data model without connecting to the REST service to retrieve additional data objects, invoke the Edit Persistence Mapping wizard. This latter wizard shares dialogs with the MAF Client Data Model From REST Web Service wizard, but does not require a connection to REST services.

You can also generate data controls from the service objects in the client data model. Once you generate data controls, you can drag and drop data and operations from the Data Controls panel to AMX pages to create the UI of your MAF application. Creating data controls is a prerequisite if you want to use the MAF UI Generator to generate a prototype UI for your MAF application. See Creating a User Interface from a MAF Client Data Model.

Note: If you connect to REST services hosted on MCS, you should not use the MCS Data Offline & Synchronization API. The MAF client data model enables your MAF application to work offline and synchronize data when the MAF application returns online.

6.3 Connecting to a REST Service to Create the Client Data Model

Connect to a REST service to identify the data object resources that you want to retrieve for use in the client data model of a MAF application.

You connect to the REST service (whether a generic service, or a service hosted on MCS) by invoking MAF Client Data Model From REST Web Service wizard. See:

- How to Connect to the REST Service to Retrieve Data Objects
- If connecting to MCS, see What You May Need to Know About the MCS Anonymous Access Key

This task is one in a series of tasks to generate a client data model in your MAF application. See Overview of Creating a Client Data Model in a MAF Application.

6.3.1 How to Connect to the REST Service to Retrieve Data Objects

Use the MAF Client Data Model From REST Web Service wizard to connect to the REST resources and identify candidate data objects that you select for inclusion in the client data model.

To create a MAF application data model:

1. Create a MAF application. See How to Create a MAF Application.

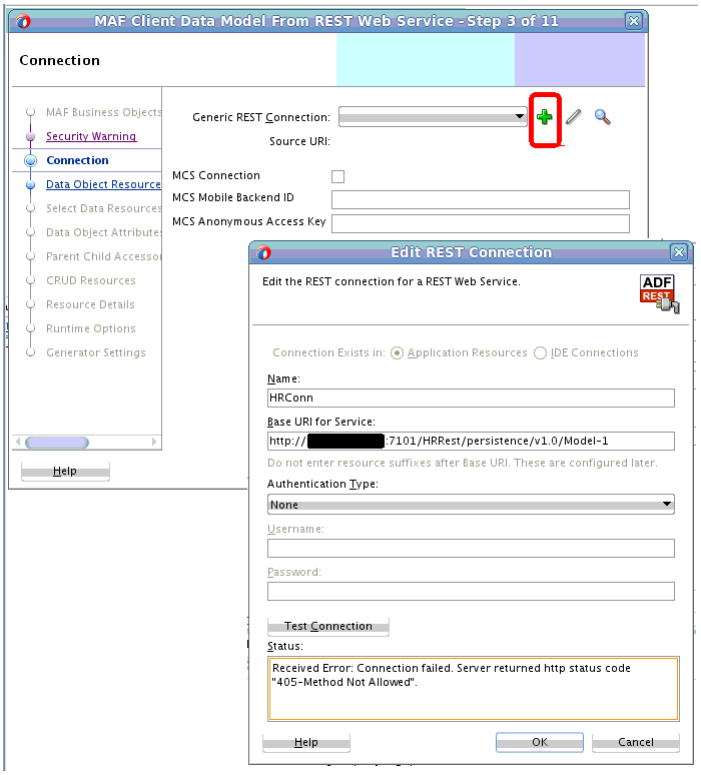
2. In the main menu, select **File** and then **From Gallery**.

3. In the New Gallery, in the Mobile Application Framework node of the Business Tier category, double-click **MAF Client Data Model From REST Web Service**.

4. On the **Welcome** page, click **Next**.

5. On the **Connection** page, shown in Figure 6-2, click the green plus icon beside the **REST Service Connection** drop-down field to invoke a dialog where you define a connection name the URL endpoint for the REST service that you want to connect to.

Figure 6-2 REST Connection Dialog



6. Enter a name for the connection and specify the URL endpoint for the REST service in the **Base URI for Service** input field.

   For your convenience, specify the part of the URL endpoint that is the same for all REST resources that you want your MAF application to consume. This reduces the amount of typing that you have to do on subsequent pages of the wizard. One exception is when you specify the URL endpoint to an instance of MCS. In this latter case, the URL endpoint should end in `/mobile`. This allows the MAF application to use the connection for both custom API calls and MCS platform API calls.

> **Note:** Clicking the **Test Connection** button does not work here as you specify an incomplete URL for the URL endpoint, as illustrated in Figure 6-2. Make sure that the URL endpoint does not end with a forward slash (the wizard checks this before it allows you to click **Next**).

7. If the REST resources you access are secured, enter the authentication information in the respective fields.

   See Accessing Secure Web Services.

8. Click **OK** to return to the Connection page and select the appropriate option:

   - Click **Next** if the REST service that you connect to is not hosted on MCS.
   - Select the **MCS Connection** checkbox if the REST service that you connect to is hosted on MCS. For information about completing the **MCS Mobile Backend ID** and **MCS Anonymous Access Key** input fields. See What You May Need to Know About the MCS Anonymous Access Key.

Once you connect to the REST service, you can discover the data objects in the REST service to retrieve and use in the client data model of the MAF application. See Discovering Candidate Data Objects for the Client Data Model.

6.3.2 What You May Need to Know About the MCS Anonymous Access Key

MAF uses the MCS anonymous access key value to create the authorization header when the MAF application accesses MCS before the application has been authenticated with MCS. This is useful if, for example, you want to send a `startSession` MCS analytics event to MCS before your user logs in.

The access key value that you use does not have to be the anonymous key. You can use the authorization key of an MCS user defined in the user realm of your MCS mobile backend. Do this if you want to, for example, access MCS storage collections or other resources that are not accessible to anonymous users.

You do not need to prefix the access key with `Basic`. MAF adds or removes the `Basic` prefix as needed. If your MAF application needs to support dynamic MCS connections, you can specify an EL expression in the MCS Mobile Backend ID and MCS Anonymous Access Key fields. After you authenticate against MCS, MAF automatically injects the authorization header into every REST call based on the user's login credentials. That is, MAF ignores the MCS anonymous access key value in the input field once the MAF application has been authenticated.

6.4 Discovering Candidate Data Objects for the Client Data Model

After a MAF application is connected to the REST service, you can select the data objects to be used in the client data model of the application.

The MAF Client Data Model From REST Web Service wizard provides the following options to discover data objects to use in your MAF application:

- **REST resource URLs**: Use this option to invoke the REST service to return candidate data.

  See How to Discover Data Objects Using a REST Resource URL.

- **Sample resource payloads**: Use this option to enter a sample JSON or XML payload. This option is useful if the first item returned by invoking a live REST service has missing attributes or child data. It is also useful if you do not currently have access to the REST service or if your application will not retrieve data, but will post data.

  See How to Discover Data Objects Using a Sample Payload.

- **RAML file**: A RAML file describes the REST service your application is going to access. MCS automatically creates a RAML file when you define the endpoints for an API in MCS.

  See How to Discover Data Objects Using a RAML File.

- **ADF BC REST Describe**: Use this option to access REST services published by ADF Business Components.

  See How to Discover Data Objects Using ADF BC REST Describe.

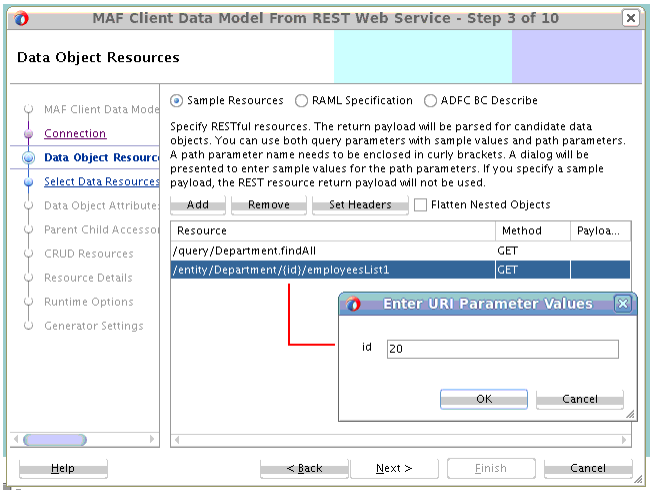
This task is one in a series of tasks to generate a client data model in your MAF application. See Overview of Creating a Client Data Model in a MAF Application.

6.4.1 How to Discover Data Objects Using a REST Resource URL

Specify one or more REST resources including the HTTP method (typically GET) to invoke the REST service and return candidate data objects based on the structure of the response payload.

Figure 6-3 shows the Data Object Resources dialog in the wizard where you specify the REST resources to query.

Figure 6-3 Querying a REST Resource for Data Object Resources



Use the **Add** and **Remove** buttons to manage the list of REST resources that you enter in the Resource list, as shown in Figure 6-3. Use the **Set Headers** button to invoke a dialog where you enter header key-value pairs if specific HTTP headers are required to invoke the REST resource. For example, if you want to make sure that the payload returned by the REST service is in JSON format rather than XML, click **Set Headers** and enter `Content-Type` as the key and `application/json` as the value in the dialog that appears.

For information about the usage of the **Flatten Nested Objects** checkbox, see What You May Need to Know About the Flatten Nested Data Objects Option.

Note the following before you click **Next** to query the REST service and return candidate data objects based on the structure of the response payload:

- MAF inspects the first item in the collection of data that returns from the REST service. If, for example, the first employee in a collection of employees is not a salesperson and, as a result, does not have an associated commission attribute, then the candidate data object for employee will not include a commission

attribute. Likewise, if the response payload returned from the REST service is a master-detail structure of departments and employees, make sure that the first department returned contains employees. If it does not, employee will not be identified as a child data object. You can add data objects and attributes on later pages in the wizard. However, it is best to avoid this work if possible.

- When you enter path parameters enclosed in curly brackets, the wizard presents a dialog (Enter URI Parameter Values) to enter sample values for the path parameter(s) when you click **Next**. In Figure 6-3, we specified {`id`} as the path parameter in the `employeesList` resource to reference the current department ID so the dialog appears. Make sure to enter a sample value that returns data. If, for example, the department with the ID value of `20` does not contain employees, no employee data object will be identified.

- The resources you specify in this page only identify candidate data objects to create and use in your application. You use later pages to specify the exact resources to use for the CRUD actions. The values you enter here will only be used as the default value for the Find All Resource input field for each data object in the CRUD Resources page later in the wizard. A situation where you use different resources to discover data objects and the Find All Resource is where the Find All Resource value returns a subset of the available attributes. For example, a Find All Resource for employees that returns the attributes you show in a list view (employeeId, firstName and lastName). When an end user clicks an employee in the list view to navigate to a detail screen, the MAF application loads the additional employee attributes by invoking a canonical employee REST resource that returns all attributes for the selected employee. This optimizes performance when you have large data sets where each data object has a lot of attributes. In such a scenario, specify the canonical REST resource in the Data Object Resources wizard page to identify all employee attributes.

6.4.2 How to Discover Data Objects Using a Sample Payload

Specify a sample payload to create candidate data objects and attributes.

This option is useful when the:

- First item of the data collection returned by invoking a REST service misses some attributes and/or child data

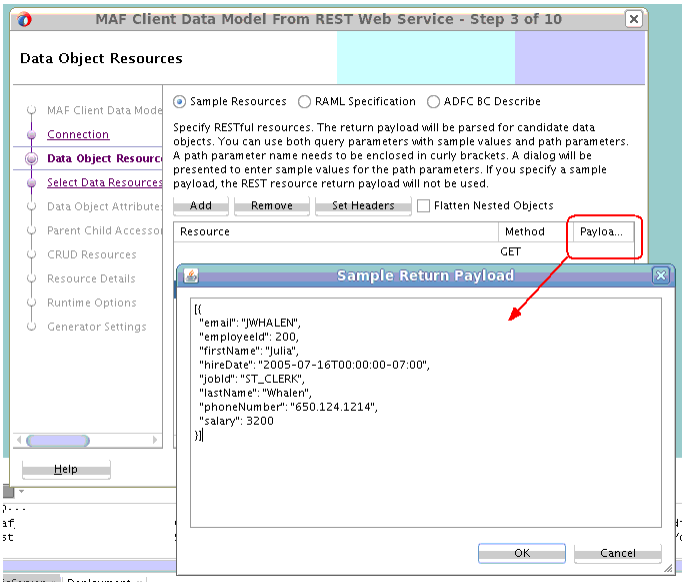
- REST resource is not available yet because the back-end development team is still working on it

- Your application is not retrieving data but only creating (posting) new data. In other words, you do not have a GET resource that returns a response payload to parse. In this case, you specify a sample payload that contains all the attributes that you want to send in your REST call when creating new data. The full signature of the REST call to create new data can be specified later on in the wizard.

When you specify a sample payload, any REST Resource you enter is not invoked. Instead, it derives a default name for your data object and the default value for Find All Resource value in the CRUD Resources page later in the wizard.

Figure 6-4 shows the Sample Return Payload dialog where you enter the sample payload. Invoke this dialog by clicking in the Payload column.

Figure 6-4 Discovering Candidate Data Objects Using a Sample Payload



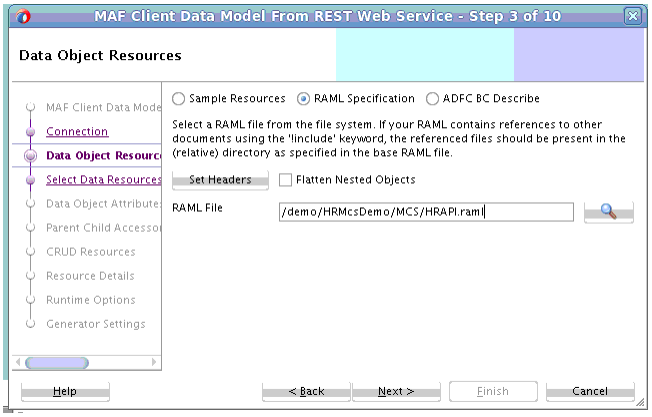
Click **OK** to return to the Data Object Resources dialog and **Next** to proceed to the next page in the wizard.

6.4.3 How to Discover Data Objects Using a RAML File

Specify a RESTful API Modeling Language (RAML) file to create candidate data objects and attributes based on the description in the RAML file.

MAF suggests data objects, parent-child relationships and CRUD resources based on the content in the RAML file. That is, subsequent pages in the wizard present default values based on the content of the RAML file.

Figure 6-5 Discovering Candidate Data Objects Using a RAML File



Use the **Set Headers** button to invoke a dialog where you enter header key-value pairs if specific HTTP headers are required to invoke the REST resource. For example, if you want to make sure that the payload returned by the REST service is in JSON format rather than XML, click **Set Headers** and enter `Content-Type` as the key and `application/json` as the value in the dialog that appears.

For information about the usage of the **Flatten Nested Objects** checkbox, see What You May Need to Know About the Flatten Nested Data Objects Option.

6.4.4 What You May Need to Know About the Flatten Nested Data Objects Option

Select the **Flatten Nested Data Objects** checkbox when you have a JSON payload like the following where, by default, manager will be created as its own data object. Selecting the Flatten Nested Data Objects checkbox includes the two manager attributes (name and email) within the department data object.

```
[
{
"departmentId" : 10,
"departmentName" :"Administration",
"locationId" : 1700,
"manager": {"name" :"Steven"
            "email"  : "steven@oracle.com"
           }
}
, ...
]
```

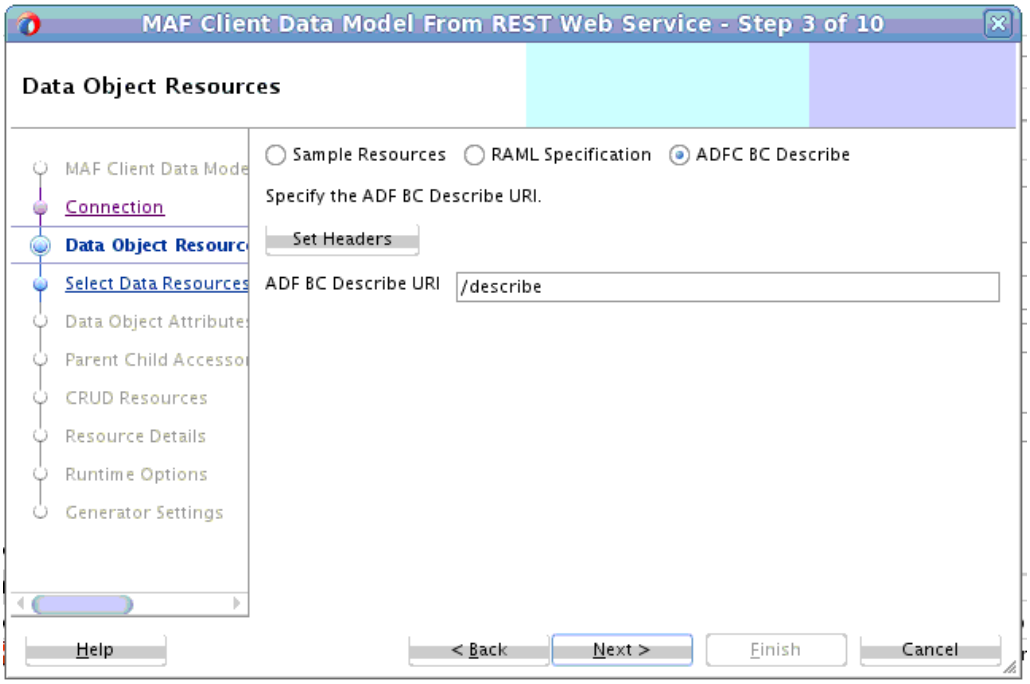
6.4.5 How to Discover Data Objects Using ADF BC REST Describe

Specify a URI to invoke REST services that are published by ADF Business Components.

Oracle ADF Business Components support publication of REST APIs that work on top of the application module that exposes a set of view objects. A `/describe` resource is automatically included when you create this type of REST API. See Creating RESTful Web Services with Application Modules in *Developing Fusion Web Applications with Oracle Application Development Framework*.

A MAF application request to the URI that includes the `/describe` resource returns metadata describing the REST resources. The wizard inspects the metadata and populates the data objects and REST resources pages later in this wizard based on this inspection.

Figure 6-6 Discovering Candidate Data Objects Using ADF Business Components Describe



6.5 Selecting and Persisting Data Objects for the Client Data Model

Select the data objects that you want to use in the client data model of your MAF application from the list of candidate data objects that MAF identifies. Also select those data objects that you want to persist so that the MAF application can access instances of the data objects when offline.

The Select Data Objects and Data Object Attributes pages in the MAF Client Data Model From REST Web Service wizard help you to perform the following tasks:

- Select and persist data objects. See How to Select and Persist Data Objects.
- Create new data objects. See How to Create New Data Objects.
- Modify data object attributes. See How to Modify Data Object Attributes.

This task is one in a series of tasks to generate a client data model in your MAF application. See Overview of Creating a Client Data Model in a MAF Application.

6.5.1 How to Select and Persist Data Objects

Select the data objects that you want to include and clear the checkboxes for unwanted data objects. Modify the values for the data objects in the Class Name column so that MAF generates valid Java class names when it creates the client data model.

Select the **Persist** checkbox for each data object that you want to persist the data for so that the MAF application can access it in offline mode. This instructs MAF to create a SQLite database table for the data object that is populated at runtime based on the REST resources you specify later to retrieve the data.

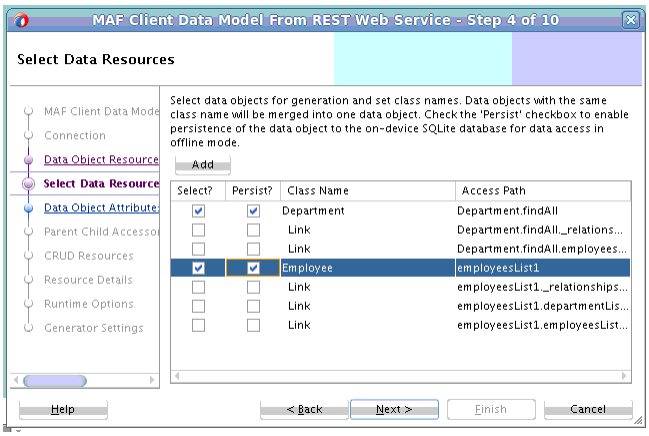
> **Note:** If you do not want to persist data objects for offline access because, for example, the data is volatile, you can still use the MAF client data model for offline transactions. The transactions are stored in a separate `DATA_SYNCH_ACTIONS` table and synchronization is independent of the data object tables.

The accuracy of the list of candidate data objects that the Select Data Objects page, shown in Figure 6-7, presents to you depends on the method used to discover the candidate data objects. A list of suggested data objects from a RAML file is, for example, typically more accurate than the list returned by a sample REST resource like the following:

```
[
  {
    "departmentId": 10,
    "departmentName": "Administration",
    "locationId": 1700,
    "_relationships": [
      {
        "_link": {
          "href": "http://localhost:7101/HRRest/persistence/v1.0/Model-1/entity/
Department/10/employees1",
          "rel": "employees1"
        }
      },
      {
        "_link": {
          "href": "http://localhost:7101/HRRest/persistence/v1.0/Model-1/entity/
Department/10/employeesList1",
          "rel": "employeesList1"
        }
      }
    ],
    "employeesList1": [
      {
        "_link": {
          "href": "http://localhost:7101/HRRest/persistence/v1.0/Model-1/entity/
Employee/200",
          "method": "GET",
          "rel": "self"
        }
      }
    ]
  },
```

The sample REST resource displayed above shows candidate data objects for link information that is included in the payload, as shown in Figure 6-7. As it is unlikely that you want to include information about links in the data model, clear the checkbox in the Select column for these candidate data objects. Also edit the value for Class Name so that a valid Java class name appears.

Figure 6-7 Candidate Data Objects Retrieved from REST Resource



6.5.2 How to Create New Data Objects

Create data objects that only live on the mobile device and are not populated through the REST web services that your MAF application connects to.

Click the **Add** button in the Select Data Objects page to add a new data object to the data model. Edit the default generated name (`NewDataObject`) to a unique valid Java class.

MAF creates a database table in the SQLite database for each data object that you add. You can populate these database tables with data using the CRUD operations from the service object that MAF generates for the data object when you complete the wizard.

6.5.3 How to Modify Data Object Attributes

Select or modify attributes of the data objects that you have selected for inclusion in the client data model of the MAF application.

You can set or change the key, required, attribute name, name in payload, Java type and database column type of each attribute as desired. For an attribute carrying sensitive data you can choose to not persist it, which causes the attribute value to be null when the application runs in offline mode.

It is important that the key attribute(s) be unique. The persistence runtime uses a data object cache based on the selected key attribute. If you have multiple data object instances with the same key, they will all be written to the same database table row. Because of this, only one instance (with the attributes of the last instance processed) appear in the UI of the MAF application. In other words, all instances are merged into one because MAF thinks it is the same instance. For a child data object, the reference attribute(s) to the parent might be part of the key. If your payload does not contain such reference attribute(s) you can create them in the Parent Child Accessors page of the wizard. Then return to this page (Data Object Attributes) to select the parent-populated attributes as the key attribute.

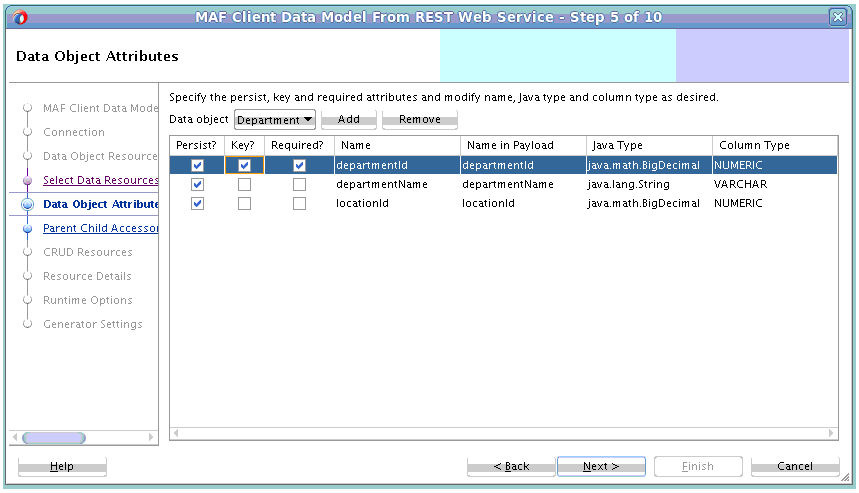
When using sample resources to identify the candidate data objects, you usually need to modify the Java type for the attributes as they typically show up as `java.lang.String`. Some attributes may show up as `java.math.BigDecimal` when the payload value is not enclosed in double quotes. You can change the Java types using the class picker, and the database column type automatically updates based on the Java type you select. Typical type changes include changing numeric attributes from `String` or `BigDecimal` to `Long` or `Integer`. Date attributes should be set to `java.util.Date` and a date format should be specified for use in the JSON

payload (for example, `yyyy-MM-dd'T'HH:mm:ssZ`). You specify the date format in the Payload Date Format field of the CRUD Resources page that you access later in the wizard.

If you used a RAML file or ADF BC REST Describe to discover the data objects, the default attribute type is usually correct and does not need modification.

> **Note:** Select each data object from the Data object drop-down list to make the necessary modifications to the object's attributes.

Figure 6-8 Modify Data Object Attributes

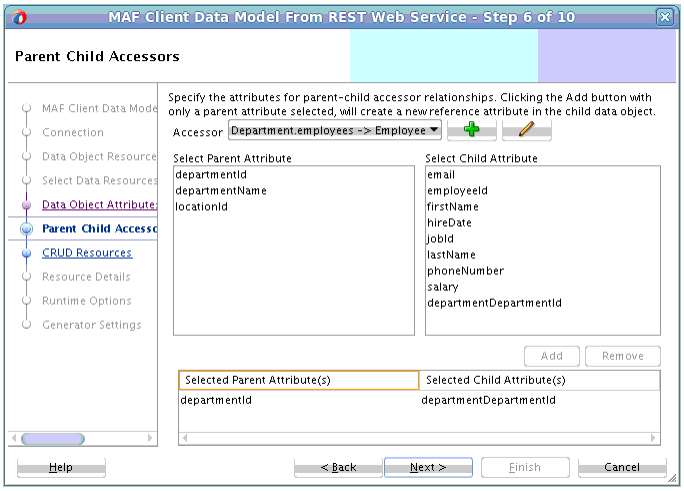


6.6 Specifying Parent-Child Relationships for Data Objects

Specify parent-child relationships for data objects where you want the UI of your MAF application to render master-detail screens.

Figure 6-9 shows the Parent Child Accessors page where you define such a relationship. Use this dialog when you want to set up a relationship such as that which exists between a department and a department's employees.

Figure 6-9 Parent-Child Accessors Page



When you specify REST resources that return parent and child data objects in the same payload, and you selected both data objects, the **Accessor** drop-down list is populated with the parent-child relationship. This also happens if you use a RAML file or ADF BC REST Describe to access REST resources. However, if you retrieve the child data object in a separate REST call and the child object is not included in the payload of the parent data object, the Accessor drop-down list will be empty. If this scenario occurs, you define the parent-child relationship manually in the Parent-Child Accessors page, as described in How to Specify a Parent-Child Relationship for Data Objects.

You need to specify attribute mappings for each parent-child relationship regardless of whether MAF automatically completed the values in the Parent Child Accessors page for you or you manually defined the parent-child relationship. You do this so that MAF restores the one-to-many or one-to-one relationship when the application runs in offline mode. In database terms, you set up a foreign key relationship between the underlying parent and child data object tables. Figure 6-9 shows an example where `departmentID` and `departmentDepartmentId` have been specified by adding both attributes to the list in the lower part of the Parent Child Accessors page using the **Add** button.

If the child data object contains a foreign key attribute that is in the response payload, you select the parent key attribute in the Select Parent Attribute list and the child data object foreign key attribute in the Select Child Attribute list, then click **Add** to add the attribute mapping. However, this foreign key attribute might not be present in the child data object because the payload did not include it. This typically happens because the child data object is nested within the parent data object payload. In this scenario, there is no need to include the foreign key attribute as the value is inferred by the hierarchical nature of the payload. MAF still needs an attribute mapping for offline mode, so you can select the parent attribute and click **Add** button without selecting a matching child attribute. Figure 6-9 above shows such a use case where `departmentID` has been selected in the Select Parent Attribute list and the **Add** button was clicked. This added the `departmentDepartmentId` attribute to the employee data object (and a column in the underlying database table. Although this attribute is not included in the REST payload when querying the employee details for a department, the runtime code populates this attribute (and underlying column value) based on the department instance for which the employee details are retrieved.

> **Tip:** The name of this parent-populated attribute is the name of the data object suffixed with the selected parent attribute name. You can change these attribute names to get clean attribute names. In the example above, do this by clicking the **Back** button, select the department data object and rename the `departmentDepartmentId` attribute to `departmentId`.

This task is one in a series of tasks to generate a client data model in your MAF application. For more information, see Overview of Creating a Client Data Model in a MAF Application.

6.6.1 How to Specify a Parent-Child Relationship for Data Objects

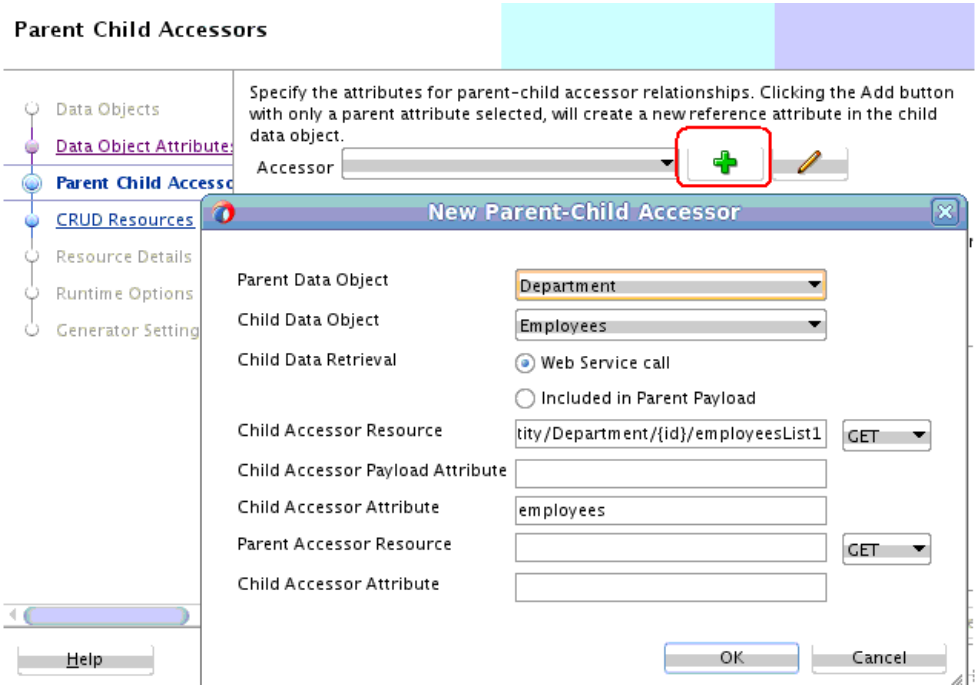
Access the Parent Child Accessors page from the MAF Client Data Model from REST Web Service wizard.

To specify a parent-child relationship for data objects:

1. In the Parent-Child Accessor page, click the Plus icon shown in Figure 6-10, and complete the fields in the dialog that appears.

Figure 6-10, for example, shows a dialog where a parent-child accessor to retrieve all employees in a department has been defined.

Figure 6-10 Parent-Child Accessor Page



Note: MAF populates the Child Accessor Resource field with the resource you used to identify a candidate data object in the Data Object Resources page. In our example, this was `/entity/Department/{id}/employeesList1`.The value in the Child Accessor Attribute field has been set to `employees`. On completion of the wizard, a `getEmployees` method is generated in the `department` data object, and an employees collection appears under the department node in the data control that you create later.

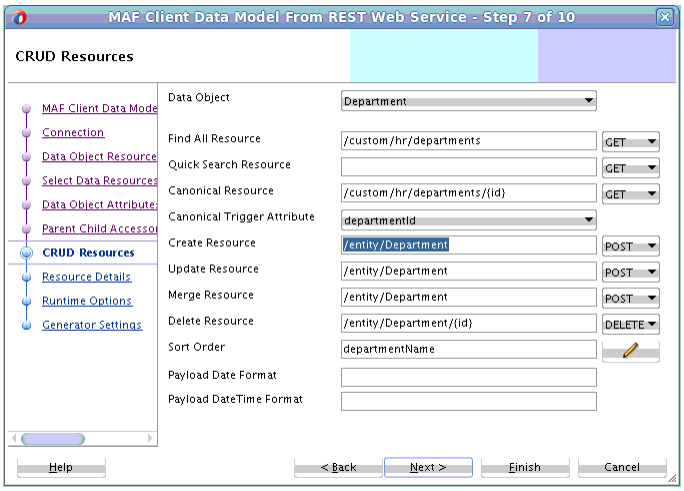
6.7 Defining CRUD REST Resources

Define CRUD actions to enable the MAF application end user to edit the data objects that the MAF application retrieves from the REST services it connects to.

This task is one in a series of tasks to generate a client data model in your MAF application. For more information, see Overview of Creating a Client Data Model in a MAF Application.

MAF provides the CRUD Resources page where you determine what actions to perform on the data objects that you retrieve to use in the client data model of the MAF application. Figure 6-11 shows the page where a number of operations, such as create, update and delete have been specified for a Departments data object.

Figure 6-11 Defining the CRUD REST Resources for the Client Data Model



MAF populates values in the CRUD Resources page depending on values that you entered previously plus the method that you used to discover the candidate data objects. If, for example, you used a RAML file or ADF BC REST Describe, MAF populates the page based on best-practice conventions for REST as follows:

- A POST resource creates a resource
- A PUT or PATCH resource updates or merge a resource
- A DELETE resource to delete a resource

If you used sample resource URLs to discover your data objects, the **Find All Resource** defaults to the sample resource. While the defaults are usually accurate you might need to change the values of the resource and/or HTTP method if your REST services do not follow best practice.

Select each data object from the **Data Object** drop-down list and enter values in the input fields as follows:

- **Quick Search Resource**: Useful for large data sets. For a large data set you typically do not want to execute a Find All Resource that returns all instances as this may cause your application to run out of memory. In such a situation, define a quick search facility in the user interface that returns only the instances that match the search criterion. With a smaller data set, use the Find All Resource to return all instances at once. Execute a quick search filter directly against the on-device SQLite database. Performance is faster as no web service is invoked.
- **Canonical Resource**: Specify a value for Canonical Resource when you have a data object with many attributes, and you only want to bring in all the attributes once a specific data object instance has been selected from a list.
- **Canonical Trigger Attribute**: If you specify the Canonical Trigger Attribute, MAF generates code in the data object class that automatically invokes the canonical REST resource when the value of the attribute is retrieved through the getter method.

  For example, if a `Department` data object's Find All Resource returns a list of department IDs and names to display on a list page and you select a `department` to go to the detail page which shows all department attributes, then we can set the Canonical Trigger Attribute to `locationId`, the `getLocationId` method

invokes when you navigate to the detail page. MAF has generated code inside the `getLocationId` method to automatically invoke the canonical REST resource.

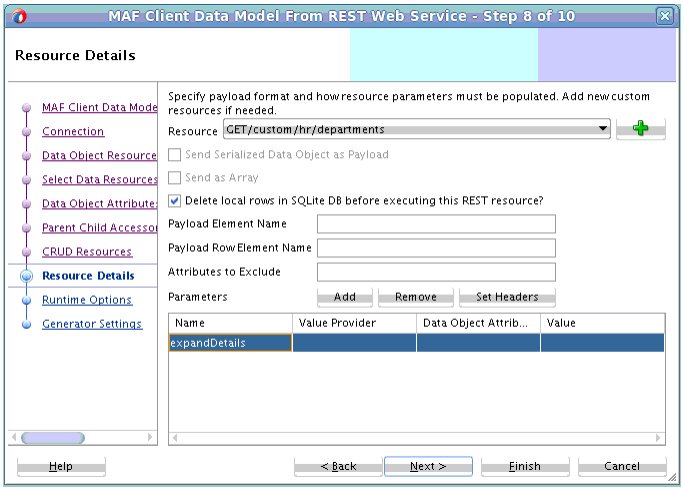
- CRUD resources (**Create Resource**, **Update Resource**, and so on): The service object, generated for each data object that has at least one REST resource specified on this page, includes a `save[DataObjectName]` method. When you invoke the `save[DataObjectName]` method by, for example, dragging and dropping it from the Data Controls panel, MAF decides based on the data object state which resource to call: `Create`, `Update`, or `Merge`. MAF makes its decision as follows:
  - If the data object state is new, MAF calls `Create` resource. If the `Create` resource is not specified, MAF calls the `Merge` resource. If neither resource is specified, the MAF application will not make a REST call.
  - If the data object state is not new, MAF calls the `Merge` resource. If `Merge` resource is not specified, MAF calls the `Update` resource. If neither resource is specified, the MAF application will not make a REST call. The data object state is automatically set to `New` when you create a new data object through the data control `Create` operation. If you programmatically create a new data object instance using a subclass of `oracle.maf.api.cdm.persistence.model.Entity`, call `setIsNewEntity(true)`.
- **Sort Order**: Define a comma-separated list of attribute names. You can suffix the attribute name with `desc` to get a descending sort for the attribute. Note that the sort order specified here is the default order in the user interface. The persistence runtime code makes it easy to add UI controls to change the sort order at runtime.
- **Payload Date Format**: Specify the Java date pattern to convert date attribute string values in the payload to a `java.util.Date` instance.

6.8 Specifying CRUD REST Resource Details

Specify resource details, including query and path parameters and their values for the various REST resources that you defined in your MAF application.

MAF provides the CRUD Resources Details page where you specify resource details. The wizard page populates the fields with default values. It is good practice to check whether these defaults make sense in your specific application. Figure 6-12 shows the page. You select additional resources from the Resource drop-down to review or customize. You can also add a custom resource by clicking the Plus icon beside the Resource drop-down menu. For more information, see Defining a Custom Resource.

Figure 6-12 Specifying CRUD REST Resource Details



To understand the options on this page, distinguish between GET (read) and non-GET (write) resources. Some fields only apply to one type of resource, or have a different meaning depending on the type of resource.

The Resource Details page helps you to perform the following tasks:

- Specify GET (Read) Resource Details

  For more information, see How to Specify GET (Read) Resource Details.

- Specify non-GET (Write) Resource Details

  For more information, see How to Specify Non-GET (Write) Resource Details.

- Add Custom Resources

  For more information, see How to Add Custom Resources.

- Specify Query and Path Parameters

  For more information, see How to Specify Query and Path Parameters.

- Add HTTP Header Parameters

  For more information, see How to Add HTTP Header Parameters.

This task is one in a series of tasks to generate a client data model in your MAF application. For more information, see Overview of Creating a Client Data Model in a MAF Application.

6.8.1 How to Specify GET (Read) Resource Details

Configure the following menu options to specify GET resource details.

- **Delete Local Rows in SQLite DB before executing this REST resource**

  If you select the **Delete Local Rows** checkbox and the GET resource is used as Find All Resource, then all rows in the table created for the corresponding data object are deleted after the REST call is made and before the REST response is processed. This is useful to ensure that any obsolete rows that are no longer included in the GET response payload do not remain visible in the application just because that data was downloaded before. If you select the **Delete Local Rows** checkbox and the GET resource was defined in Parent-Child Accessor dialog to

retrieve child data objects, then all those child rows will be removed just before the REST call is executed.

- **Payload List Element Name**

  The **Payload List Element Name** instructs MAF how to find the actual array of JSON objects representing the resource data object in the response payload. For example, if the response payload looks like the following:

  ```
  {"departmentList":[{"departmentId":10, "departmentName": "Accounting",
  "locationId": 1400, "managerId": 103}, {"departmentId":20, "departmentName":

                  "Marketing", "locationId": 1400, "managerId": 102} ]}
  ```

  Then you should set the Payload List Element Name to `departmentList`. Note that `departmentList` does not have to be an attribute of the top-level JSON object, MAF recursively traverses the response object tree until it finds an object with this attribute name. You can leave this field blank if the response payload directly starts with the array you need.

- **Payload Row Element Name**

  You can use the Payload Row Element Name field to instruct MAF how to find the correct JSON object within a specific array instance. For example, with the following response payload:

  ```
  {"departmentList":["department":{"departmentId":10, "departmentName":
  "Accounting", "locationId": 1400, "managerId": 103}}, "department":
  {"departmentId":20,

          "departmentName": "Marketing", "locationId": 1400, "managerId": 102}) ]}
  ```

  Set the Payload Row Element Name to `department`. If you have to traverse a deeper nested object tree to get to the actual JSON object that MAF should parse, use the dot notation in this field. For example, `departmentTopObject.department`

6.8.2 How to Specify Non-GET (Write) Resource Details

Configure the following menu options to specify non-GET resource details.

- **Send Serialized Data Object as Payload**

  The **Send Serialized Data Object as Payload** checkbox should usually be selected when making a PUT, POST or PATCH request. It might be checked with DELETE request as well, depending on how the delete is implemented at the server side. By selecting this checkbox, MAF creates a request payload with key value pairs for each data object attribute that has the **Name in Payload** property set in the Data Objects Attributes page, described in How to Modify Data Object Attributes:

  ```
  {"departmentId":10, "departmentName": "Accounting", "locationId": 1400,
  "managerId": 103}
  ```

- **Send as Array**

  Select the **Send as Array** checkbox to enclose the JSON object that holds the key-value pairs with brackets, as shown in the following example. This checkbox is disabled until you select the **Send Serialized Data Object as Payload** checkbox.

  ```
   [{"departmentId":10, "departmentName": "Accounting", "locationId": 1400,
  "managerId": 103}]
  ```

- **Payload Row Element Name**

  Use to instruct MAF to nest the actual JSON object with key-value pairs inside another object. For example, if the request payload should look like this:

  ```
  {"department":{"departmentId":10, "departmentName": "Accounting", "locationId":
  1400, "managerId": 103}}
  ```

  Then you should set the **Payload Row Element Name** to `department`.

  If your request payload requires payload attribute names different from the GET resource, you cannot specify these write-specific payload attribute names here. You need to create your own remote persistence manager that subclasses the `RestJSONPersistenceManager` and overrides the `getPayloadKeyValuePairs` method. You can register your custom remote persistence manager on the next page of the wizard, or register it later, see Editing the Client Data Model in a MAF Application.

- **Attributes to Exclude**

  Specify a comma-delimited list of attribute names that should not be serialized into the JSON object used as request payload. For example, if you specify `departmentId`, the following payload exclude it from serialization into the JSON object:

  ```
  {"departmentName": "Accounting", "locationId": 1400, "managerId": 103}
  ```

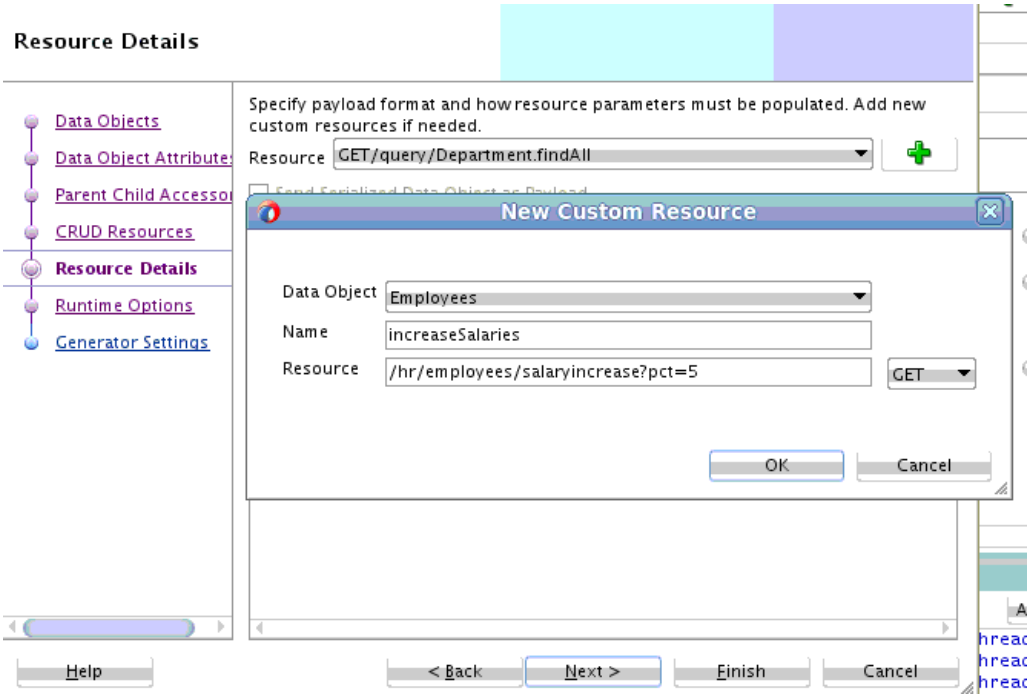
  You could achieve the same by nullifying the value for **Name in Payload** field in the Data Object Attributes page, but then `departmentId` would also be ignored in GET requests which is typically not what you want. Note that you should use the data object attribute names in this field, not the payload names of these attributes.

6.8.3 How to Add Custom Resources

A custom resource is a REST call you make that does not map to a find, create, update or delete action that you specified on the CRUD Resources page.

You can add a custom resource in the Resource Details page. Click the green plus icon, as shown in Figure 6-13, to invoke the New Custom Resource dialog where you can define the custom resource:

Figure 6-13 Adding a Custom Resource



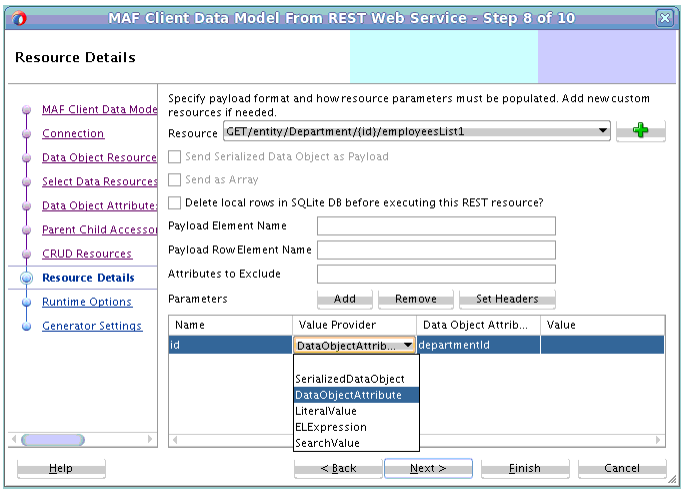
In the service class that MAF generates for the data object, a method will be added with the name you specified in the Name field. Calling this method invokes the REST resource. Custom methods are also included in the data synchronization mechanism of the MAF client data model, so, if you call the custom method in offline mode, it will be registered as a pending synchronization action and the REST call will be executed when the MAF application is online again.

6.8.4 How to Specify Query and Path Parameters

Path parameters enclosed in curly brackets that you specified in the REST resources of the CRUD Resources page or the Parent-Child Accessors dialog appear by default in the Parameters list.

Use the **Add** button to add additional query parameters.

Figure 6-14 Specifying Query and Path Parameters



When a MAF application executes the REST resource, it automatically populates the resource query and path parameter values based on the Value Provider you choose:

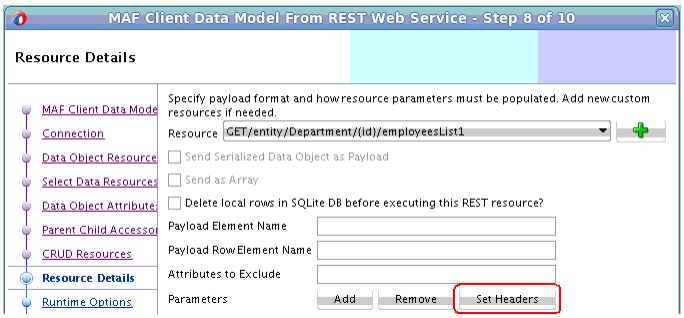
- **DataObjectAttribute**: Populates the parameter with the value of a data object attribute. When using this value provider, you need to choose a value from the Data Object Attribute drop-down list. The data object whose attributes are displayed in the drop-down list are determined by the usage of the resource. For example, the above resource returns the employees within a department, so it is assumed you want to select an attribute from the department data object to set the context for the employees list.
- **SerializedDataObject**: If the data that must be persisted should be sent through a query parameter rather than in the request body, you choose this setting. With this value provider, the other columns should remain empty.
- **LiteralValue**: Populates the parameter with a literal value specified in the Value column.
- **ELExpression**: Populates the parameter with a value obtained by evaluating an EL Expression. You specify the EL Expression in the Value column. You can use EL Expressions with any scope (applicationScope, pageFlowScope, viewScope, deviceScope, preferenceScope) but it is your own responsibility that the expression is valid in the execution context of the REST service call. Remember that when making transactions in offline mode, the REST service will be invoked later and the EL expression context will then be determined by the task flow and page that triggers the data synchronization.
- **SearchValue**: Populates the parameter with the value of the quick search value entered in the user interface. You will typically use this value provider only with the Quick Search Resource that you define in the CRUD Resources page described in Defining CRUD REST Resources.

6.8.5 How to Add HTTP Header Parameters

The HTTP headers that you specified in Discover Objects Resources page are applied by default to all resources that you have specified.

If you want to remove a header and/or add another header specific for one resource, click the **Set Headers** button to invoke a dialog where you enter or modify header key-value pairs if HTTP header parameter changes are required.

Figure 6-15 Setting HTTP Header Parameters



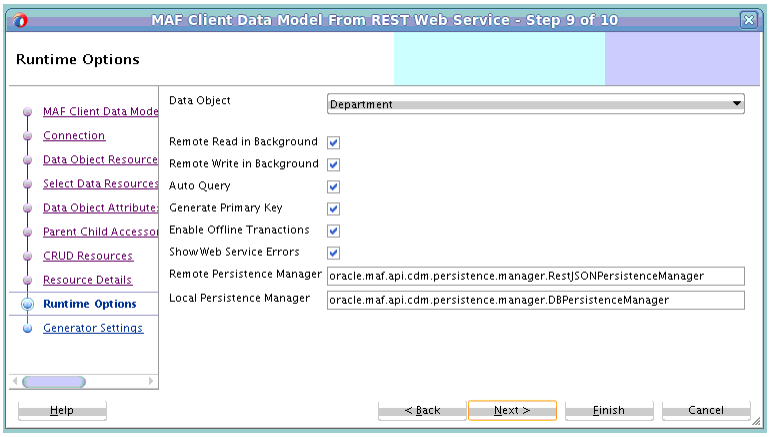
6.9 Setting Runtime Options for the Client Data Model

Set runtime options for the MAF application that you generate on completing this wizard.

This task is one in a series of tasks to generate a client data model in your MAF application. For more information, see Overview of Creating a Client Data Model in a MAF Application.

Figure 6-16 shows the Runtime Options page where you determine how the generated MAF application behaves at runtime.

Figure 6-16 Setting Runtime Options for a MAF Application



Configure the fields shown in Figure 6-16 as follows:

- **Remote Read in Background**: If selected, the GET requests (typically the Find All, Find and Canonical resources) specified for the data object execute in a background thread. It is a good idea to select this checkbox as it enhances the end user's perception of the MAF application's performance. MAF first queries the SQLite database and show these results immediately on the screen (assuming you persisted the data object, as described in How to Select and Persist Data Objects). REST calls will be made in the background without blocking the user interface. Once the MAF application receives the REST response, it automatically updates the user interface.

- **Remote Write in Background**: If selected, non-GET requests (typically the Create, Update, and Merge resources) specified for this data object execute in a background thread. You will usually leave this checkbox selected as it enhances the end user's perception of the MAF application's performance. After triggering a REST call through, for example, a Save button, the end user can continue to use the application. The user interface is not blocked for the duration of the REST call. Again, if the REST response includes some attributes with server-updated values, the MAF application automatically refreshes the user interface.

- **Auto Query**: If selected, MAF automatically queries the on-device database for all rows and/or call resource specified by Find All Resource when the service object class for the data object is initialized. This is convenient when building your AMX pages using the bean data control that you create for the service object class. The bean data control exposes a collection element that you can drag and drop onto an AMX page to, for example, create a list view or form view. When auto-query is selected, this collection element returns all data objects. It initially returns what is present in the SQLite database and refreshes with the remote collection once execution of the Find All Resource completes. If you clear this checkbox, you will need to execute a finder method in your task flow before navigating to the AMX page. Otherwise the AMX page will show no data.

- **Generate Primary Key**: If selected, the MAF application automatically generates a primary key when a new data object is inserted into SQLite database when the primary key attribute value is still null. This feature only works when the primary key attribute is a numeric attribute. MAF queries the SQLite database for the current maximum value and increments this value with 1.
- **Enable Offline Transactions**: If selected, write REST calls can be invoked while the MAF application is offline. MAF registers the transaction (create, update, remove or a custom action) as a pending data synchronization action. Once the MAF application comes online, MAF synchronizes these actions and executes the associated REST calls. Note that when the MAF application is online, and the REST call fails because the server is not available or the server throws some error when invoking the REST resource, the transaction is also registered as a pending synchronization action. MAF retries execution the next time the MAF application makes a REST call. If this checkbox is cleared, an error message appears to the end user when the MAF application is offline or when the REST call fails.
- **Show Web Service Errors**: If selected, any REST Call failure shows an error message popup in the user interface. This is a useful setting during development so you can see errors. We recommend you clear this checkbox when publishing your MAF application in production. You typically do not want to show technical details about REST call failures to end users.
- **Remote Persistence Manager**: Register your own remote persistence manager. This is useful if you want to extend the default behavior of the MAF remote persistence manager.
- **Local Persistence Manager**: Register your own local persistence manager. This is useful if you want to extend the default behavior of the MAF local persistence manager.

6.10 Generating the Client Data Model

Specify the project and package names for the data objects that MAF generates for the client data model.

This task is one in a series of tasks to generate a client data model in your MAF application. For more information, see Overview of Creating a Client Data Model in a MAF Application.

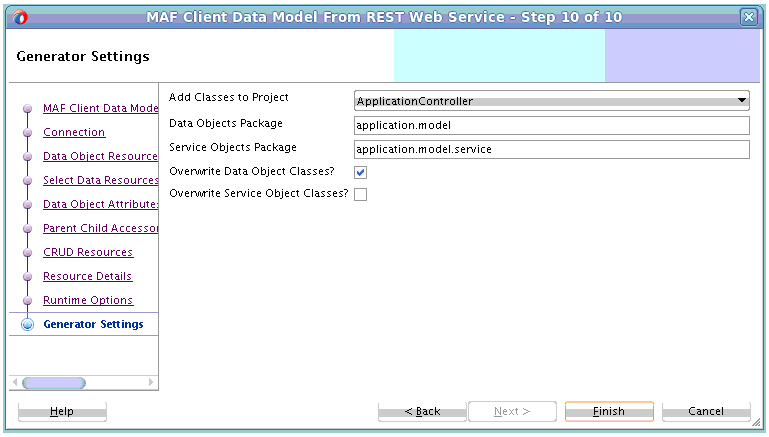
Figure 6-17 shows the Generator Settings page where you determine the location and package names for the objects in the client data model.

Configure the fields in this page as follows:

- **Add Classes to Project**: Select the project within your application where you generate the Java classes. MAF recommends using the default ApplicationController project. This allows you to use the classes in application-scoped managed beans as well as in the application lifecycle methods. It is also avoids class loader issues when using the same data objects in multiple application features. Each MAF application feature has its own class loader, but all share the parent class loader of the ApplicationController project.
- **Data Objects Package**: Specify the Java package name used by the Java classes that are generated for each data object.
- **Service Objects Package**: The Java package used by the Java classes that are generated for each service object. Such a class is generated when a data object has at least one REST resource associated.

- **Overwrite Data Object Classes**: If selected, MAF overwrites existing data object classes. If you added custom code to these classes after a previous generation, you will lose this code.
- **Overwrite Service Object Classes**: If selected, MAF overwrites existing service object classes. If you added custom code to these classes after the previous generation, you will lose this code. This checkbox is clear by default because the service object class typically does not need to change when payload structures change. Also, this class is the most likely to contain custom code that you added after generation.

Figure 6-17 Generating the Client Data Model



6.11 Editing the Client Data Model in a MAF Application

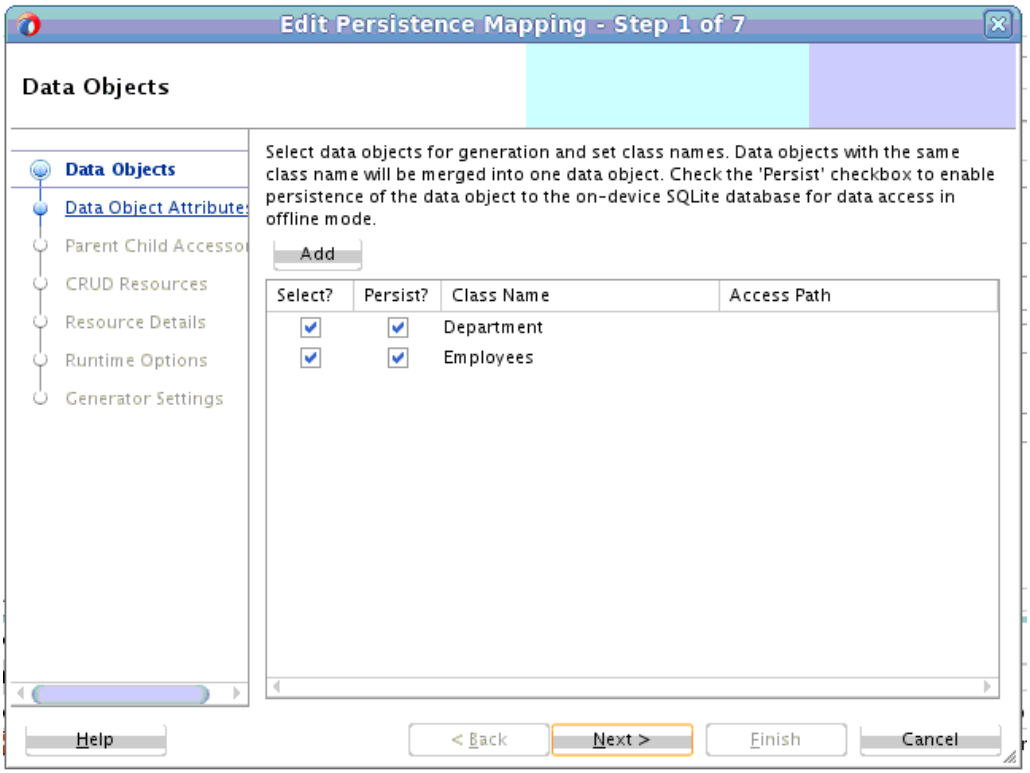
Describes how to invoke wizard(s) where you can modify a previously-generated client data model in a MAF application.

MAF supports iterative development of the client data model in a MAF application. You can extend and refine the client data model after its initial creation. Use one of the following two options:

- Re-run the Client Data Model from REST Service wizard to discover new data objects or edit existing data objects. You can re-enter this wizard to discover new data objects. If you do not want to modify existing data objects, clear the associated **Select** checkboxes so the existing data objects do not show up in subsequent wizard pages and no Java classes will be regenerated for these data objects, regardless of the settings on the last generator settings wizard page.
- Run the Edit Persistence Mappings wizard shown in Figure 6-18. Invoke this wizard from the **Edit Persistence Mappings** context menu entry that appears when you right-click the project that contains the client data model.

  The Edit Persistence Mappings wizard displays the same pages as the Client Data Model from REST Service wizard except that it does display pages to connect to REST services or discover new data objects. As a result, you typically use this wizard to edit existing data objects although you can use it create data objects that you store in the SQLite database which do not require REST calls to read or write data from a REST service.

Figure 6-18 Editing the Client Data Model



6.12 Accessing the SQLite Database Using the MAF Client Data Model DBPersistenceManager

The MAF client data model provides an API (`oracle.maf.api.cdm.persistence.manager.DBPersistenceManager`) to access the on-device SQLite database within your MAF application.

The MAF client data model delegates all interaction with the SQLite database to `DBPersistenceManager`. Use `DBPersistenceManager` as an alternative to writing the low-level JDBC statements described in How to Connect to the Database. For more information about the role of `DBPersistenceManager` in the client data model, see Introduction to the Client Data Model in a MAF Application. For information about the `DBPersistenceManager` and the methods it exposes, see the *Java API Reference for Oracle Mobile Application Framework*.

The following examples demonstrate how you get an instance of `DBPersistenceManager`, perform some search operations and other operation on data objects (insert, update, and remove). Apart from using the constructor as shown in the following example, you can also get an instance of the `DBPersistenceManager` by calling `getLocalPersistenceManager()` from an instance of a service object. This returns an instance of `DBPersistenceManager` or a subclass if you registered a custom local persistence manager in the Runtime Options page of the client data model wizard, described in Setting Runtime Options for the Client Data Model.

```
/* Use the default constructor to get an instance of DBPersistenceManager.*/
DBPersistenceManager pm = new DBPersistenceManager();

/* findAll takes a class or class name as an argument and returns a list of all data
objects.*/
```

```
List<Department> departments = pm.findAll(Department.class);


/* The overloaded find method returns a filtered list of data objects.
 *
 * There are various ways to specify your filter conditions, as demonstrated by the
following examples.
 *
 * A quick search method that returns all employees where at least one of the String
 * attributes (firstName, lastName, emailAddress, and so on) starts with "king"
(case insensitive).
 * This translates to use of the SQL LIKE operator where the search value is
suffixed with %. */
List<Employee> emps = pm.find(Employee.class, "king");


/* A search method which allows you to specify the search attributes. It returns all
employees where at least
 * one of the attributes passed in as list in the third argument starts with "king"
(case insensitive). */
List<String> searchAttrs = new ArrayList<String>();
searchAttrs.add("firstName");
searchAttrs.add("lastName");
List<Employee> emps = pm.find(Employee.class, "king", searchAttrs);


/* This is a search method which allows you to specify separate values for each
 * attribute you want to search on, it will return all employees where the firstName
 * equals "Steven" and the lastName equals "King". The query is case sensitive.
 */
Map<String, String> searchAttrs = new HashMap<String, String>();
searchAttrs.put("firstName", "Steven");
searchAttrs.put("lastName", "King");
List<Employee> emps = pm.find(Employee.class, searchAttrs);


/* The findByKey method returns one data object based on its primary key.
 * It first checks the data object cache. If the department with id 10 does not
exist in the cache, it queries
 * the database. If you do not want to check the cache, add a third boolean argument
checkEntityCache, as demonstrated
 * by the second example
 */
Department dep = (Department) pm.findByKey(Department.class, new Object[] { 10 });
Department dep = (Department) pm.findByKey(Department.class, new Object[] { 10 },
false);


/* Insert, updates and remove a row for a data object instance.
 * MAF automatically commits the change if the second argument is true. If false, you
 * need to call pm.commit() later.
 * MAF generates the primary key attribute for you when calling insertEntity if you
selected
 * the Generate Primary Key checkbox in the Runtime Options page of the MAF client
data model wizard.
 */
pm.insertEntity(department, true);
pm.updateEntity(department, true);
pm.removeEntity(department, true);
```

Using Custom SQL Statements

You can specify custom SQL statements if the above data object-based APIs do not provide an option to execute the SQL statement you need. We distinguish between SQL SELECT and SQL DML statements.

Use code as demonstrated in the following example if the result of your SELECT statement needs to be converted to a data object or a data object list.

```
DBPersistenceManager pm = new DBPersistenceManager();
ClassMappingDescriptor descriptor = ClassMappingDescriptor.getInstance(Dealer.class);
StringBuffer sql = pm.getSqlSelectFromPart(descriptor);
sql.append(" WHERE SALES_ACCOUNT_ID not in (SELECT DEALER_ID FROM
PRIORITY_ASSIGNMENT WHERE PRIORITY_ID=" +
     priority.getId() + ")");
sql = pm.constructOrderByClause(sql, descriptor);
ResultSet set = pm.executeSqlSelect(sql.toString(), null);
List<Dealer> dealerList = pm.createEntitiesFromResultSet(set,
descriptor.getAttributeMappingsDirect());
```

Since the result must be converted to data object list, the SELECT clause should include all the columns, and the FROM clause should use the table name that corresponds to the data object. By using the `getSqlSelectFromPart` convenience method, the SELECT and FROM clause will be automatically created for you using the data object class descriptor from the `persistence-mapping.xml` file. You can then append your custom WHERE clause to the SQL statement. You can also append your custom ORDER BY clause, or, if you just want to use the default order by as registered with the data object class descriptor you can use the convenience method `constructOrderByClause` as in the example above. Once you have constructed your SQL SELECT statement, you can execute it using the `executeSqlSelect` method which returns a JDBC `RowSet` object. You then convert the JDBC `RowSet` to a data object list using the `createEntitiesFromResultSet` method.

For all SQL SELECT results that cannot be converted to a data object list, you can define the whole SQL statement, and also process the JDBC `RowSet`. The following is an example of an aggregate query:

```
DBPersistenceManager pm = new DBPersistenceManager();
String sql = "SELECT AVG(SALARY) FROM EMPLOYEE";
ResultSet set = pm.executeSqlSelect(sql, null);
try {
  set.first();
  int averageSalary = set.getInt(1);
}
catch (SQLException e) {
  sLog.severe("Error executing SQL statement: "+e.getLocalizedMessage());
}
```

For single-row insert, update or delete you will typically use the data object-based API described above. However, if you want to insert, update or delete multiple rows at once, you can use the `executeSqlDml` method. Here is an example where we increase the salary of all clerks by 10%.

```
DBPersistenceManager pm = new DBPersistenceManager();
String sql = "UPDATE EMPLOYEE SET SALARY = SALARY * 1.1 WHERE JOB_ID='CLERK'";
pm.executeSqlDml(sql, null, true);
```

If the third `doCommit` argument is `true`, the batch update will be automatically committed. If you set it to `false`, you need to call `pm.commit()` later.

6.13 Defining a Custom Resource

Describes how to add custom REST resources in your MAF application's client data model that do not map directly to the standard CRUD resources supported by the MAF client data model.

The Resource Details wizard page, described in Specifying CRUD REST Resource Details, allows you to add custom resources. When you do this, MAF generates an additional method with the same name as your custom resource into your service object with the following signature:

```
public void doSomething(Department department) {
  invokeCustomMethod(department, "doSomething");
}
```

The advantage of this approach is that it is fast and easy to implement. If your MAF application is offline or the REST call fails due to a server error, the custom resource action will be registered as a pending data synchronization action. MAF makes the call later on when the MAF application returns to online mode. In other words, it is handled in the same ways as standard CRUD transactions in offline mode, as described in Synchronizing Offline Transactions from a MAF Application.

The disadvantage of this approach is that it limits you in how you supply query and path parameters, and how you provide the request payload (this can only be the serialized data object). The value of query and path parameters can be defined declaratively using the options described in Specifying CRUD REST Resource Details. If you need complete flexibility in how you construct your URI path with query and path parameters, it is better to go for the programmatic approach and code your REST call in Java.

You can invoke any REST resource using the `invokeRestService` method on the remote persistence manager, which is either the `RestJSONPersistenceManager`, or if you connect to Oracle Mobile Cloud Service (MCS), the `MCSPersistenceManager`. For more information about these classes and methods, see *Java API Reference for Oracle Mobile Application Framework*.

You could also use the `RESTServiceAdapter`, described in Creating a Rest Service Adapter to Access Web Services. However, the `invokeRestService` method has the following advantages:

- One line of code makes the REST call
- It handles all responses with HTML status code in `200-299` range successfully. No exception is thrown for status codes `201-299`, as is the case with the `RESTServiceAdapter`.
- If you have enabled web service logging, you can easily view the REST call details.
- If connecting to MCS, you do not need to specify the `Oracle-Mobile-Backend-Id` and (anonymous) `Authorization` header parameters.

The `invokeRestService` method has the following signature:

```
public String invokeRestService(String connectionName, String requestType, String
requestUri,
                                                String payload, Map<String, String>
headerParamMap, int retryLimit, boolean secured)
```

Note: The last argument (`secured`) is not used. It included for backward compatibility.

You typically add a method to your service object in which you invoke the `invokeRestService` method. You can then make the REST call from the user interface by dragging and dropping the method onto your AMX page. Here is a sample method that makes such a REST call:

```
public void invokeSomeRestResource(String pathParamValue,String queryParamValue) {

  if (isOnline()) {
      RestJSONPersistenceManager rpm = new RestJSONPersistenceManager();
      String uri = "/someResourcePath/"+pathParamValue+"?
someQueryParam="+queryParamValue;
      String result = rpm.invokeRestService("MyRESTConn", "GET", uri, null, null, 0,
false);
    // do something with the result
  }
}
```

If you want to execute the REST call in the background, the code looks as follows:

```
public void invokeSomeRestResource(String pathParamValue,String queryParamValue) {
  TaskExecutor.getInstance().execute(true, () -> {
    if (isOnline())
    {
      RestJSONPersistenceManager rpm = new RestJSONPersistenceManager();
      String uri = "/someResourcePath/"+pathParamValue+"?
someQueryParam="+queryParamValue;
      String result = rpm.invokeRestService("MyRESTConn", "GET", uri, null, null, 0,
false);
      // do something with the result
    }
  });
}
```

If the response should be converted to a list of entities and stored in SQLite database, like the standard Find All resource, use the `handleReadResponse` method on the remote persistence manager do all this for you. This is the signature of this method:

```
public <E extends Entity> List<E> handleReadResponse(String jsonResponse, Class
entityClass,
                                                     String collectionElementName,
String rowElementName,
                                                     List<BindParamInfo>
parentBindParamInfos, boolean deleteAllRows)
```

The `collectionElementName` and `rowElementName` arguments map to the **Payload List Element Name** and **Payload Row Element Name** that we specify for standard GET resources. See Specifying CRUD REST Resource Details for an explanation on how to set the values of these arguments based on the structure of the response payload. If the response returns an array as top-level object, you need to specify root as the value for `collectionElementName`.

You can leave the `parentBindParamInfos` argument `null`, and the `deleteAllRows` argument is obsolete (It is included for backwards compatibility). If you want to delete all local rows prior to processing the response payload, then you need to add the code to do so yourself.

The following code sample illustrates how the above example can be extended to process the response payload into a list of entities:

```
public void invokeSomeRestResource(String pathParamValue,String queryParamValue) {
  if (isOnline()) {
      RestJSONPersistenceManager rpm = new RestJSONPersistenceManager();
      String uri = "/someResourcePath/"+pathParamValue+"?
someQueryParam="+queryParamValue;
      String result = rpm.invokeRestService("MyRESTConn", "GET", uri, null, null, 0,
false);
      List<Employee> emps = rpm.handleReadResponse(result, Employee.class, "root",
null, null, false);
      setEntityList(emps);
  }
}
```

6.14 Executing Custom Logic After CRUD REST Calls

Describes the methods that MAF generates in service classes to execute CRUD operations against the local and/or remote persistence manager(s). MAF generates these methods from the MAF Client Data Model From REST Web Service wizard.

A department service class, for example, will have methods like `findAllDepartment`, `saveDepartment` and `removeDepartment`. These methods indirectly make the corresponding REST calls if you configured these calls in the MAF Client Data Model From REST Web Service wizard. So, if you want to execute custom logic based on a REST response, you might be inclined to add your logic at the end of these methods. However, this will not work if you execute these REST calls in the background, which might be the case when you select the **Remote Read in Background** and/or **Remote Write in Background** checkboxes in the Runtime Options page. In this scenario, your custom logic would already be executed while the REST call is still in progress in a separate background thread.

You need to follow a different approach to make sure your custom logic executes after the REST call completes in the background. If you need to add custom logic after a read REST call (`GET`), you can override one of the following methods in your service class:

- `executeRemoteFindAll`
- `executeRemoteFindAllInParent`
- `executeGetCanonical`

Add your custom logic after the call to `super`, as demonstrated in the following example that overrides `executeRemoteFindAll`:

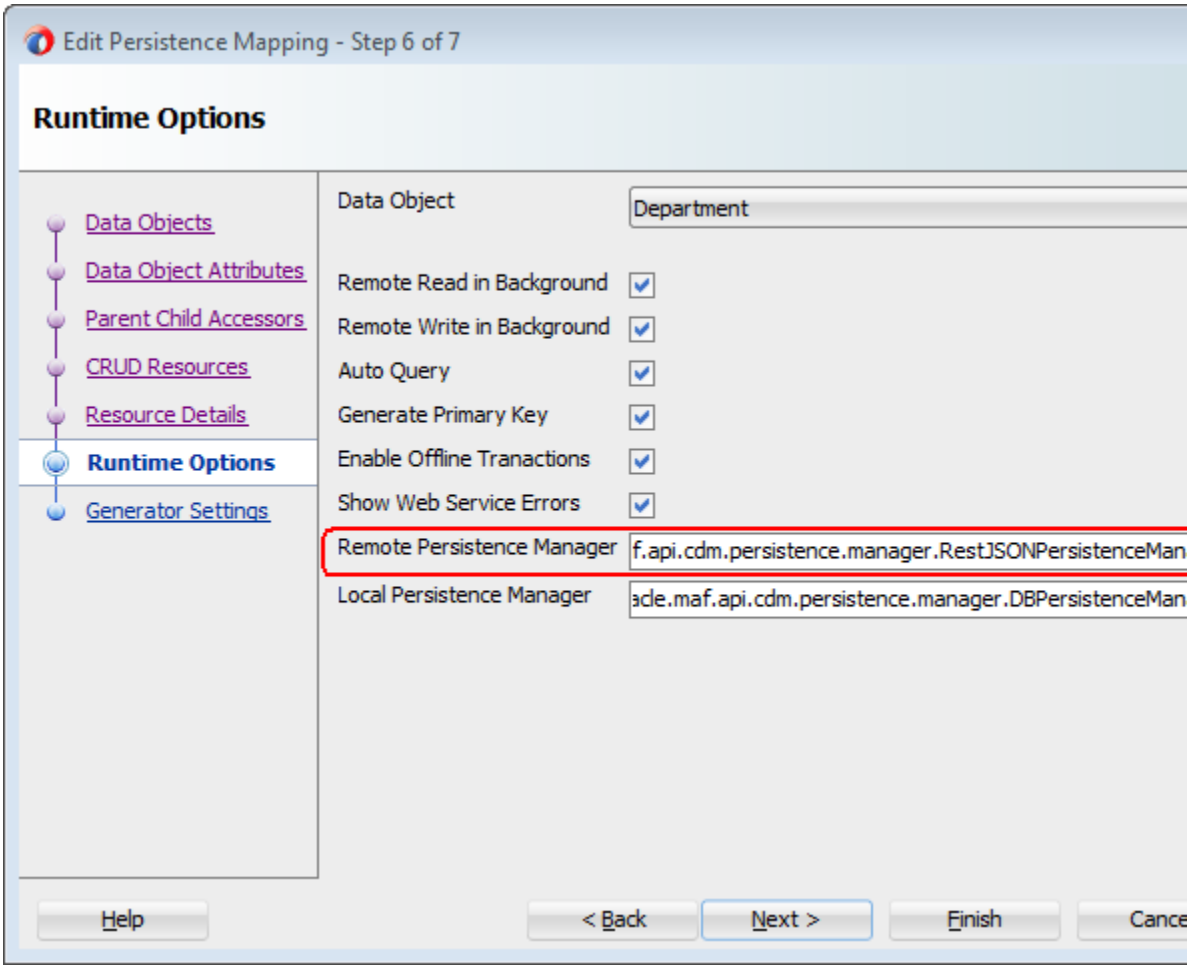
```
@Override
protected List<Department> executeRemoteFindAll()
{
  // call super to get the new list of departments from REST call
  List<Department> result =  super.executeRemoteFindAll();
  // do some custom logic here
  ...

  // return department list
  return result;
}
```

If you need to execute custom logic after a write REST call (`POST`, `PUT`, `PATCH`, `DELETE`), create a subclass of the remote persistence manager, and override one of the following methods:

- `insertEntity`
- `updateEntity`
- `mergeEntity`
- `removeEntity`

Find the remote persistence manager class that you need to subclass by using the Edit Persistence Mapping wizard. Navigate to the Runtime Options page shown in Figure 6-19, select the correct data object, and copy the class name from the **Remote Persistence Manager** field. Create a new Java class that subclasses from this class and override one or more of the methods listed above. To ensure that the MAF application runtime uses your custom remote persistence manager, enter the fully qualified class name of your subclass in the **Remote Persistence Manager** field shown in Figure 6-19. For information about invoking the Edit Persistence Mapping dialog to access the Runtime Options page, see Editing the Client Data Model in a MAF Application.

Figure 6-19 Remote Persistence Manager



You need to subclass the remote persistence manager for write methods because of the offline transactions supported by MAF. In offline mode, REST calls will not be

executed but will instead be registered as pending synchronization action(s). When online again, the MAF application executes the REST call directly using your remote persistence manager subclass, bypassing the service class.

If you need to execute custom logic based on the raw response from the REST call, before MAF has done any processing, override the following method in the remote persistence manager:

```
public String invokeRestService(String connectionName, String requestType, String
requestUri,
                                String payload, Map111String,String222
headerParamMap, int retryLimit, boolean secured)
```

This method handles all REST calls in MAF and returns the response of the REST call. After calling `super`, you can also call `getLastResponseStatus()` and `getLastResponseHeaders()` to get more information about the response.

6.15 Getting Programmatic Access to Service Objects

Describes how to get programmatic access to an instance of a service object.

You typically create a data control for your service objects to create the user interface of your MAF application using drag and drop from the Data Controls panel. If you want to access a service object from a managed bean or a lifecycle listener method, create an instance of your service object as follows:

```
DepartmentService service = new DepartmentService();
```

Where `DepartmentService` extends from `oracle.maf.impl.cdm.persistence.service.EntityCRUDService`.

This triggers a REST call if you have selected **Auto Query** in the Runtime Options page, as described in Setting Runtime Options for the Client Data Model, and you specified a value for **Find All Resource** , as described in Defining CRUD REST Resources. If you do not want to perform this automatic query for the instance you create programmatically, use the following constructor that takes `autoQuery` as a boolean argument:

```
DepartmentService service = new DepartmentService(false);
```

If you do want to perform an automatic query, write code as follows:

```
DepartmentService service = new DepartmentService();
List<Department> deps = service.getDepartment();
// do something with the departments
```

This will not work reliably if the REST call executes in the background which happens when the **Remote Read in Background** checkbox is selected in the Runtime Options page. In that case, the `getDepartment()` method call only returns the departments stored in the local database (if any), as the REST call executes in a background thread. So, the safest way is to use the following constructor:

```
public DepartmentService(boolean doRemoteReadInBackground, boolean
doRemoteWriteInBackground)
{
  super(false);
  setDoRemoteReadInBackground(doRemoteReadInBackground);
  setDoRemoteWriteInBackground(doRemoteWriteInBackground);
}
```

This constructor calls `super` with the `autoQuery` argument set to `false`, and will set the `remoteReadInBackground` and `remoteWriteInBackground` properties as specified in your constructor call. In other words, using this constructor you can get an instance that ignores the selections made in the Runtime Options page (and stored in the `persistence-mapping.xml` file).

```
DepartmentService service = new DepartmentService(false,false);
// get latest department list from server by making synchronous REST call
service.findAllDepartmentRemote;
// get a handle in the department list
List<Department> deps = service.getDepartment();
// do something with the department list
```

In managed bean code, you might need access to the service object instance used by the data control. Do this using the following convenience method:

```
DepartmentService service = (DepartmentService)
EntityUtils.getEntityCRUDService(Department.class);
```

This method looks up a data control instance by the name of your service object class. Be aware though that if you use this method without the data control being instantiated yet (that is, used on an AMX page), it creates a new instance using the default constructor which might trigger an unwanted REST call as explained previously. A data control instance lives in the context of an application feature. If you have used the same data control in the AMX pages of two different application features, you will have two instances of the underlying service object. The application feature context in which you execute the `getEntityCRUDService` method determines which instance returns.

Move as much logic as possible into your service class to reduce the need to get a handle on the data control instance in your managed bean code. If you evaluate lots of value binding and method binding expressions, and subsequently execute lots of these method bindings, you might want to rethink your coding strategy.

Finally, if you only need access to the local SQLite database in your custom Java code and do not require any REST calls to be made, you do not have to create a service object instance. Instead, use an instance of `oracle.maf.api.cdm.persistence.manager.DBPersistenceManager` to query or manipulate data objects. For more information, see the *Java API Reference for Oracle Mobile Application Framework*.

6.16 Understanding Usage of the Primary Key

Every data object must have a primary key to ensure that rows in the SQLite database can be uniquely identified. MAF also uses the primary key to minimize data object creation in the data object cache and to prevent the creation of multiple instances of the data object.

Use the following method to retrieve a specific data object instance from the cache:

```
EntityCache.getInstance().findByUID(Class entityClass, Object[] key)
```

Use the following method from the `DBPersistenceManager` class to retrieve a data object from the cache or from the database if the data object has not been cached:

```
findByKey(Class entityClass, Object[] key)
```

The primary key can be a composite key consisting of multiple attributes/columns. This is why the `key` argument in the above methods takes an object array. If the primary key is a single numeric attribute, MAF automatically generates a primary key

if you select the **Generate Primary Key** checkbox in the Runtime Options page described in Setting Runtime Options for the Client Data Model. MAF queries the SQLite database for the current maximum value and increments this value by `1` when the `DBPersistenceManager`'s `insertEntity` method is called, either by the framework, or by your custom code.

You can also generate a primary key yourself, by using the following method:

```
EntityUtils.generatePrimaryKey(Entity entity, int increment)
```

> **Note:** The latter method only works if you selected the **Generate Primary Key** checkbox for the data object as it checks this flag in the `persistence-mapping.xml` file.

Understanding MAF Management of Server-Derived Primary Key Values

The primary key value that you create or generate for new data object instances can be regarded as a temporary primary key. Obviously, this primary key is not guaranteed to be unique at the server side. Typically, when you invoke a REST service that inserts the data object to the remote server (the **Create Resource** you specified in the wizard), the remote server generates a truly unique primary key.

If the REST call that inserts the data object returns the data object with the new server-derived values in the response, then the MAF client data model automatically updates the temporary primary key with the server-derived primary key. The MAF client data model deletes the database row with the temporary primary key and inserts a new row with the server-derived key. It also updates the database object cache.

Make sure that the Create Resource endpoint returns the full data object in the response to benefit from this MAF client data model feature.

SQLite Auto-Increment Functionality

Avoid using SQLite's auto-increment functionality because MAF uses the primary key to identify instances in the data object cache and to update existing rows in the SQLite database when the MAF application fetches the up-to-date data from the server. For this reason, it is better to specify the primary key using one or more data object attributes that are included in the payload coming from the remote server rather than SQLite's auto-increment functionality.

It is fine to use SQLite's auto-increment functionality for data objects where no existing data objects instances are retrieved from the server. That is, where no GET resource called to load data into the SQLite database.

For more information about SQLite's auto-increment functionality, see SQLite's documentation.

6.17 Using Filtered Entity Lists

Describes the methods that the MAF client data model generates to facilitate the creation of filtered lists in the UI of your MAF application.

MAF generates a getter method in the data object's CRUD service class that returns a collection of the data object instances. An employee data object's service class (`EmployeeService`) has, for example, the following method:

```
public List<Employee> getEmployee() {
  return getEntityList();
}
```

The `EntityCRUDService` superclass stores the returned list in a private member variable (`private EntityList<E> entityList`). This list contains all data object instances retrieved from the local SQLite database and/or from a call to the Find All REST resource if you selected the **Auto Query** checkbox in the Runtime Options page, as described in Setting Runtime Options for the Client Data Model.

To filter this list based on a quick search field in the user interface, use the `find[entityName]` method that the data object CRUD service class also generates. For example, the following method is generated for an employee data object.

```
public void findEmployee(String searchValue) {
  super.find(searchValue);
}
```

Drag and drop the method from the Data Controls panel onto your AMX page to render a search field and a command button to invoke the `find[entityName]` method. When the end user taps the command button to invoke the `find[entityName]` method, the `find[entityName]` method delegates to the `find` method in the `DBPersistenceManager` class. For more information about the `DBPersistenceManager` class, see Accessing the SQLite Database Using the MAF Client Data Model DBPersistenceManager and *Java API Reference for Oracle Mobile Application Framework*. The result returned by the `find` method is stored as the new content of the `entityList` variable. A data change event is send to the user interface to refresh the list correctly after the end user clicks the command button.

You can also write a custom method to filter entity lists. The following example filters a list of employees to only clerks.

```
public void filterClerks()
{
  DBPersistenceManager pm = new DBPersistenceManager();
  Map<String,String> searchAttrs = new HashMap<String,String>();
  searchAttrs.put("jobId","CLERK");
  List<Employee> clerks = pm.find(Employee.class,searchAttrs);
  setEntityList(clerks);
}
```

The call to `setEntityList` updates the `entityList` variable and sends the required data change events to the user interface once `setEntityList` is invoked.

> **Note:** It does not matter whether this method is invoked through some UI action or through Java code in a background thread.

Assume, for example, that the user interface of your application needs to show multiple filtered lists at the same time. You want to show, for example, a list of employees and a list of clerks. In this scenario, the `filterClerks` method cannot update the `entityList` member variable because `entityList` already shows all employees in the user interface. The solution is to add a getter method to return the list of clerks. This adds a clerks collection attribute to the `EmployeeService` data control that you can drag and drop onto AMX pages to show the list of clerks. The following code example illustrates such a getter method:

```
public List<Employee> getClerks()
{
  DBPersistenceManager pm = new DBPersistenceManager();
  Map<String,String> searchAttrs = new HashMap<String,String>();
  searchAttrs.put("jobId","CLERK");
  List<Employee> clerks = pm.find(Employee.class,searchAttrs);
```

```
  return clerks;
}

@Override
protected void setEntityList(List<Employee> entityList)
{
  super.setEntityList(entityList);
  // we also need to refresh the clerks list
  getPropertyChangeSupport().firePropertyChange("clerks", null, getClerks());
  getProviderChangeSupport().fireProviderRefresh("clerks");
}
```

The previous example overrides the `setEntityList` method to handle the situation where we retrieve the latest list of employees in the background by calling the Find All REST resource. When a MAF application executes a Find All REST call in the background, it calls `setEntityList` when the response is returned and the database has been updated with the latest set of data object instances included in the response payload. In our example, the user interface shows both a list of all employees and a list of clerks. We want both lists updated when the REST call completes. Overriding the `setEntityList` method ensures that the `getClerks` method is executed again once the latest set of employees has been retrieved from the REST service and ensures that data change events are sent to refresh the clerks list.

6.18 Creating a User Interface from a MAF Client Data Model

MAF applications where you have generated a client data model can avail of a number of features provided by MAF that facilitate your development work.

Once you create the client data model, you typically create a data control for each entity service class that the client data model generated. These data controls include a large number of methods that allow you to build your MAF AMX pages by dragging and dropping operations and collections from the Data Control palette in JDeveloper. For more information, see How to Create Data Controls from the Client Data Model.

If you created your client data model as described in Overview of Creating a Client Data Model in a MAF Application, you can make use of the following features that MAF provides to make you productive as you build and test the user interface layer of your MAF application:

- Use the MAF User Interface Generator wizard to test your client data model and prototype the user interface of your MAF application. This wizard generates a complete MAF application feature that includes a task flow, AMX pages, and data bindings. To use this wizard, you create a data control and then generate the application feature from the data control. For more information, see How to Use the MAF User Interface Generator.
- View pending data synchronization actions using reusable data synchronization application feature included with MAF. Alternatively, create your own AMX pages using the data synchronization service data control.
- Force your MAF application to behave as if it is offline while it is actually connected to the internet.
- Add a visual indicator to show end users that the MAF application is performing work the background. You typically use this visual indicator to inform end users that the MAF application is reading or writing data to or from a remote server.

- Inspect the REST calls your MAF application makes by using the web service calls application feature included with MAF

6.18.1 How to Create Data Controls from the Client Data Model

Describes how to create data controls from the service objects created during the generation of the MAF application's client data model.

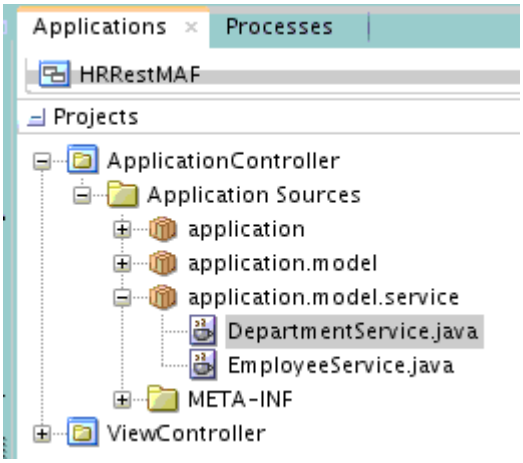
Once you complete the creation of the client data model, as described in Overview of Creating a Client Data Model in a MAF Application, you typically create data controls from the service objects generated during the latter task. These data controls facilitate the creation of the user interface of your application. You can drag the collections, attributes, or operations from the Data Controls panel in JDeveloper and drop them on the AMX page where MAF prompts you to choose the appropriate AMX components to render data or UI controls in the AMX page. Alternatively, you can generate an application feature from the data control using the MAF User Interface Generator wizard, as described in How to Use the MAF User Interface Generator.

To create data controls from the client data model:

1. In the Applications window, navigate to the service object that the client data model wizard generated.

   The location of the service objects depends on the options you chose in the client data model wizard. Figure 6-20, for example, shows the DepartmentService and EmployeeService service objects in the ApplicationController project of a MAF application.

   Figure 6-20 Client Data Model Service Object

2. In the Applications window, right-click the service object (for example, `DepartmentService.java`) and choose **Create Data Control** from the context menu that appears.

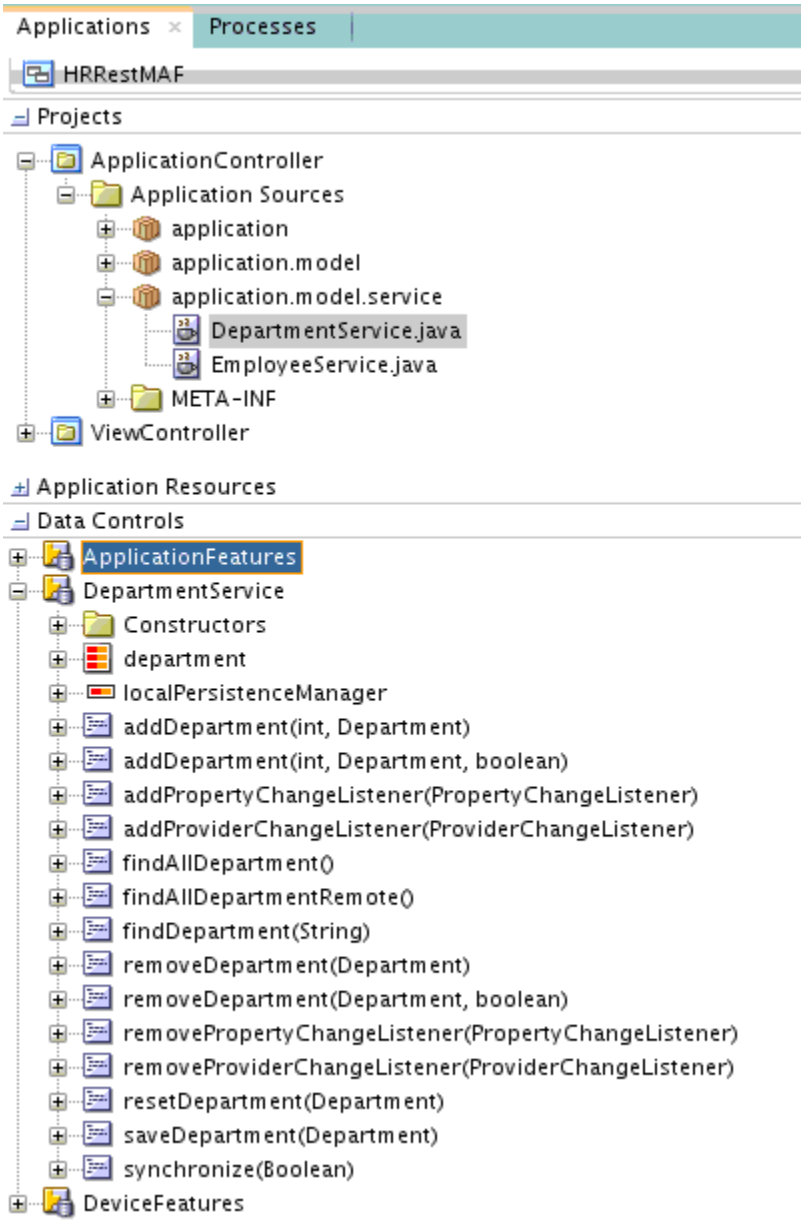
3. Complete the fields in the Create Bean Data Control wizard that appears and click **Finish**.

6.18.2 What Happens When You Create a Data Control from the Client Data Model

MAF generates a data control from the service object with a range of elements and operations that you can drag and drop to AMX pages.

This data control can also be used by the MAF User Interface Generator wizard to create an application feature, as described in How to Use the MAF User Interface Generator.

Figure 6-21 Data Control Created from a Client Data Model Service Object



6.18.3 How to Use the MAF User Interface Generator

Use the MAF User Interface Generator wizard to create a MAF application feature with a task flow, AMX pages and data bindings if you generated a client data model in your MAF application.

This wizard helps test your client data model and creates a prototype user interface for your MAF application. Before you use the wizard, create a client data model as described in Overview of Creating a Client Data Model in a MAF Application. Generate data controls from the service objects that are created when you create a client data model. Make sure that the data control for which you want to generate a

MAF application feature appears in the Data Control window of JDeveloper. You may need to click the refresh icon.

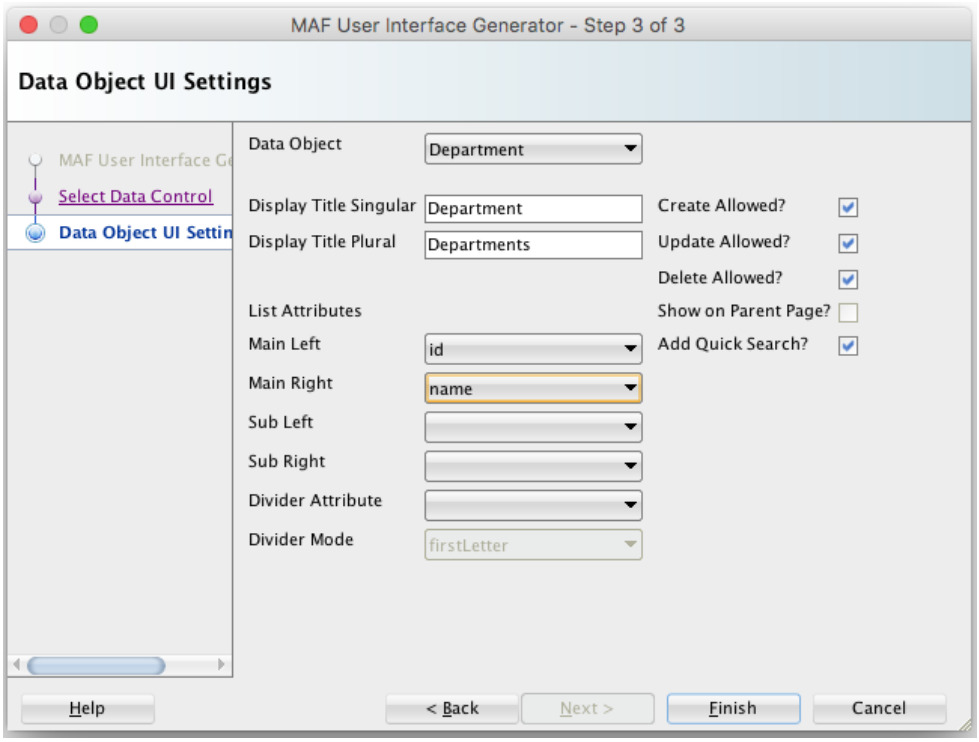
The wizard generates a list view page and a form page for every data object that the data control exposes. You can choose the data object attributes to expose in the list view page. The form page displays all data object attributes.

To use the MAF User Interface Generator wizard:

1. In the main menu, choose **File** and then **From Gallery**.
2. In the New Gallery, in the Mobile Application Framework node of the Client Tier category, double-click **MAF User Interface Generator**.
3. Click **Next** in the welcome page and in the **Select Data Control** page, select the data control for which you want to generate a MAF application feature.
4. You can select the **Enable Feature Security?** checkbox in the Select Data Control page if you have already configured security for your MAF application, as described in Securing MAF Applications . You can perform this task at a later time.
5. Click **Next** to navigate to the **Data Object UI Settings** page, shown in Figure 6-22, and configure values as follows:

    - **Data Object**: Choose the data object from the drop-down list and configure the display title of this object plus the operations that the user interface exposes for end users to perform (create, update, or delete the object, and so on). Select the checkboxes for **Create Allowed**, **Update Allowed** and **Deleted Allowed** if you have defined CRUD REST resources as described in Defining CRUD REST Resources.
    - **Show on Parent Page**: Select for child data objects where you want to include the list view of the child data object inside the form page of the parent data object. Clear the checkbox to display on a separate page. Use of this checkbox assumes that you have configured a parent-child relationship, as described in Specifying Parent-Child Relationships for Data Objects.
    - **List Attributes**: Specify the data object attributes to display in the list view page. You can also choose a list divider.

Figure 6-22 Configuring the User Interface for Pages Generated by MAF User Interface Generator



6. Click **Finish** to generate the MAF application feature includes the user interface.

6.18.4 What Happens When You Generate a User Interface

MAF makes the following changes to your MAF application when you complete the MAF User Interface Generator wizard:

- Adds the newly-created MAF application feature to the `maf-feature.xml` file.
- Generates a task flow in a sub-directory of the `Web Content` directory. The name of the sub-directory derives from the name of the data object (for example, `Department`).
- Generates AMX pages for every view activity in the generated task flow and generates AMX page definitions for the generated AMX pages.
- Adds `GoToFeature` and `Connectivity` managed beans to the `adfc-mobile-config.xml` file. The generated AMX pages use these managed beans.
- Adds references to the newly-created MAF application feature plus two other application features that MAF includes when you generate the client data model. These are the data synchronization application feature and the web services call application feature. For more information about using these application features, see Synchronizing Offline Transactions from a MAF Application and Inspecting Web Service Calls in a MAF Application.

6.19 Synchronizing Offline Transactions from a MAF Application

MAF applications allow end users to perform transactions when the application is in offline mode. MAF performs a data synchronization action for each transaction when the application is next in online mode.

A transaction in this context is a method invocation on a service object that results in a REST call to create, modify or delete data. If you invoke a service object's method, such as, saveDepartment(Department), MAF checks if there is a corresponding REST resource to call. If there is no corresponding REST resource, MAF invokes the method against the on-device SQLite database if applicable. If there is a corresponding REST resource to call, MAF creates a data synchronization action. The data synchronization action holds the type of resource to execute (Insert, Update, Remove, or a custom action) and all data of the data object instance which the transaction requires.

If the MAF application is in online mode, MAF starts the synchronization action and invokes the corresponding REST resource. If the MAF application is in offline mode and you enabled offline transactions for the data object, MAF registers the data synchronization action and stores it as a pending synchronization action. MAF synchronizes the action when the MAF application is next in online mode. MAF preserves the exact sequence in which transactions are committed when synchronizing.

You enable offline transactions for a data object by selecting the **Enable Offline Transactions** checkbox, as described in Setting Runtime Options for the Client Data Model. If you disable offline transactions for a data object, MAF throws a "`Device is offline`" exception if an end user attempts to perform an offline transaction on the data object. Prevent this exception by disabling the UI controls when the MAF application is in offline mode so that end users cannot create a transaction that throws the exception.

MAF stores data synchronization actions in the `PENDING_SYNCH_ACTIONS` table of the SQLite database. The data object's SQLite database table does not store information related to data synchronization actions. Assume, for example, an end user deletes a department when the MAF application is in offline mode. This action removes the corresponding row from the `DEPARTMENTS` table in the SQLite database and a corresponding data synchronization action is added to the `PENDING_SYNCH_ACTIONS` table. When the MAF application is next in online mode, the REST action associated with the data synchronization action of deleting the department is performed. This ensures that the local SQLite database tables reflect the latest transactions performed by end users regardless of whether the associated REST calls have been performed or not.

When a MAF application returns to online mode from offline mode, MAF waits for the application to invoke a REST call. When this event occurs, MAF synchronizes the pending data synchronization actions before processing the REST call. This synchronization makes sure that the subsequent REST call(s) and responses to the MAF application do not use obsolete data. You can explicitly invoke this automatic synchronization when the MAF application returns to online mode, even if there are no REST calls to trigger the automatic synchronization by invoking the synchronize(boolean) method from any service object in your MAF application. The `oracle.maf.impl.cdm.persistence.service.EntityCRUDService` class, from which all service object classes extend provides the `synchronize(boolean)` method. Invoking this method once from any service object class performs an automatic synchronization for all pending data synchronization actions in the MAF application. The boolean argument for synchronize determines whether synchronization happens in the background (`true`) or in the foreground (`false`).

Although not recommended, you can disable the default behavior of MAF applications to synchronize pending synchronization actions before invoking a REST call. For more information, see What You May Need to Know About Disabling Automatic Synchronization.

MAF applications cannot detect if data synchronization conflicts occur when a MAF application returns to online mode and synchronizes data. Assume, for example, that an end user of your MAF application updates a department when the MAF application is in offline mode. Elsewhere, another user of a web application that accesses the same data set modifies the same department information. MAF cannot detect this latter change. When the MAF application returns online, MAF attempts to synchronize the changes in the `PENDING_SYNCH_ACTIONS` table. To resolve and work around the issue just described, you need to identify and resolve data synchronization conflicts at the location where all applications (mobile, web, and so on) access the data set that your MAF application accesses.

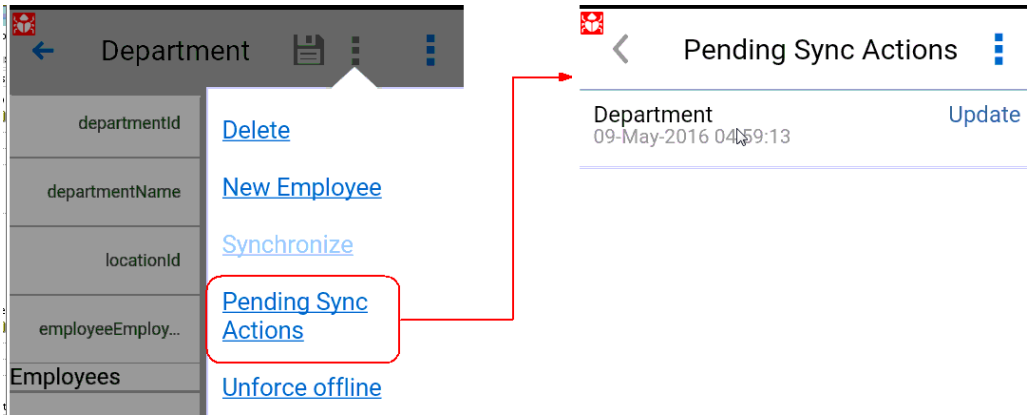
When MAF tries to synchronize pending synchronization actions by calling the corresponding REST resource, the REST call may return an error response for an action because, for example, the server is down. If this happens, MAF keeps the data synchronization action in the `PENDING_SYNCH_ACTIONS` table. It also updates the action with the timestamp of the synchronization attempt and the synchronization error. MAF continues processing the remaining data synchronization actions in the table despite the failure of one or more actions. MAF retries these pending data synchronization actions the next time it performs synchronization. You can expose these pending synchronization actions to end users so they can view and make a decision on what do with actions that remain to be synchronized. For more information, see How to View Pending Synchronization Actions. You can also write custom logic to execute in your MAF application on completion of a synchronization action. You can, for example, write code that executes in response to failure to synchronize. For more information, see How to Add Custom Logic to Handle Failed Synchronization Actions.

6.19.1 How to View Pending Synchronization Actions

Add the `DataSyncFeature.jar` feature archive to your MAF application to display an application feature where end users can view and remove pending synchronization actions.

Once added, your MAF application includes an application feature that includes a menu to view pending synchronization actions. End users can tap each pending synchronization action to view more detail and make a decision to remove the action or leave it for MAF to re-attempt to synchronize it. Figure 6-23 shows a composite image of the menu entry and the Pending Sync Actions screen in a MAF application where this feature archive has been added.

Figure 6-23 Viewing Pending Sync Actions

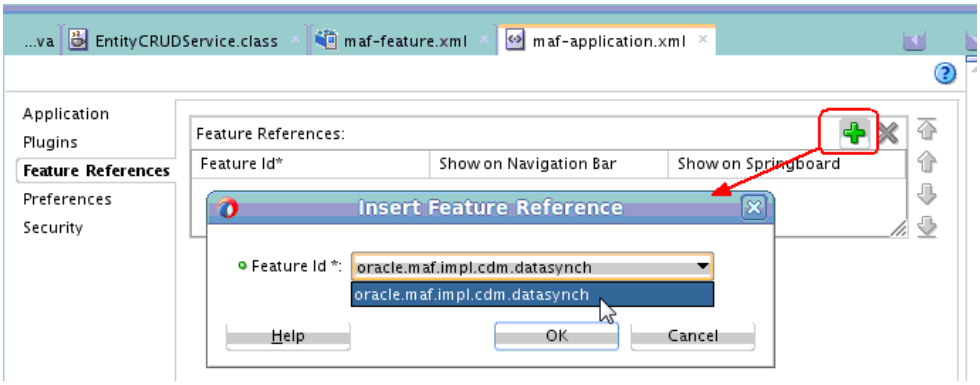


Note: The MAF User Interface Generator wizard automatically adds this FAR to the MAF application on completion. For more information, see Creating a User Interface from a MAF Client Data Model.

To add the `DataSyncFeature.jar` FAR:

1. In the main menu, choose **Application** and then **Application Properties**.
2. In the Application Properties dialog, navigate to the Libraries and Classpath page and click **Add JAR/Directory**.
3. In the Add Archive or Directory dialog that appears, navigate to the `jdeveloper/jdev/extensions/oracle.maf/FARs/CDM` directory in your JDeveloper installation and select the `DataSynchFeature.jar` file so that it appears under the Classpath entries for your MAF application.
4. Click **OK** to close the dialogs.
5. In the Feature References page of the `maf-application.xml` file's overview editor, add the data synchronization feature to your application by selecting it from the Insert Feature Reference dialog, as shown in Figure 6-24.

Figure 6-24 Adding FAR to View Pending Synchronization Actions



6.19.2 How to Add Custom Logic to Handle Failed Synchronization Actions

Write custom code that executes in your MAF application once data synchronization completes if you want to handle failed synchronization actions programmatically.

Perform the following steps to write Java code in and register it in your MAF application:

1. Create a Java class that extends from `oracle.maf.impl.cdm.persistence.service.DataSynchManager`.

2. Override the `dataSynchFinished` method to add custom logic after the data synchronization action(s) complete.

```
protected void dataSynchFinished(java.util.List<DataSynchAction>
succeededDataSynchActions,
                                          java.util.List<DataSynchAction>
failedDataSynchActions)
```

   The `dataSynchFinished` method has two arguments: a list of successful synchronization actions and a list of failed synchronization actions. You can use this method to, for example, warn end users that one or more transactions failed and will be re-tried later, or you can inform them that all pending data synchronization actions have been processed successfully.

   Add the Java code to implement your synchronization policy to this method. The following simple example informs the user about the number of successful and failed synchronization actions:

```
package application.model.service;

import java.util.List;
import oracle.adfmf.framework.exception.AdfException;
import oracle.maf.impl.cdm.util.MessageUtils;
import oracle.maf.api.cdm.persistence.service.DataSynchAction;
import oracle.maf.impl.cdm.persistence.service.DataSynchManager;

public class MyDataSynchManager extends DataSynchManager {

  public DataSynchManager() {
    super();
  }

  @Override
  protected void dataSynchFinished(List<DataSynchAction>
succeededDataSynchActions,
                                   List<DataSynchAction>
failedDataSynchActions)  {
    int ok = succeededDataSynchActions.size();
    int fails = failedDataSynchActions.size();
    int total = ok + fails;
    MessageUtils.handleMessage(AdfException.INFO,
       total + " data synch actions completed. Successful: " + ok + ", Failed: "
+ fails);
  }
}
```

3. Register your class in your application's `mobile-persistence-config.properties` file, as demonstrated by the following example:

```
datasync.manager.class=application.model.service.MyDataSynch
Manager
```

The `mobile-persistence-config.properties` file is in `ApplicationRootDirectory/ApplicationController/src/META-INF/` folder.

6.19.3 What You May Need to Know About Disabling Automatic Synchronization

Automatic synchronization of transactions from a MAF application can be disabled.

To disable automatic synchronization of transactions from a MAF application:

1. Create a new abstract Java class that extends the `oracle.maf.impl.cdm.persistence.service.EntityCRUDService`
2. Override the `synchronize` method and comment out the call to `super.synchronize` so the method performs no execution
3. Modify your service object classes to extend from the just-created subclass instead of extending from `EntityCRUDService`

If you disable automatic synchronization, configure your MAF application so that end users can explicitly start a synchronization action. Use the following statement to explicitly trigger data synchronization:

```
new DataSynchService().synchronize(true);
```

The boolean argument determines whether the synchronization happens in the background (`true`) or in the foreground (`false`). The `DataSynchService` class is located in the following package: `oracle.maf.api.cdm.persistence.service`. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

We do not recommend disabling automatic synchronization of transactions as it can lead to out-of-date data in your application and a confusing user experience. The following example use case for a user (John) illustrates this point. John performs the following actions in his application:

- Gets the latest list of departments when starting the application in online mode
- Modifies the name of department 10 in offline mode
- Removes department `20` in offline mode
- Creates a new department `280` in offline mode
- Leaves the application
- John starts the application again in online mode and the latest list of departments is retrieved from the server while the 3 pending sync actions are not yet processed
- John will now see the old department name of department `10`
- John will now see department `20` again although he already removed it
- If you (the MAF application developer) selected the **Delete Local Rows** checkbox for the **Find All Resource** for department data object, the new department (`280`) that John created disappears again.

Only when John manually starts a synchronization action, will he see the latest data again including the changes he made in offline mode once he refreshes the department

list again with the latest data from the server (through a user interface control or by restarting the application).

6.20 Understanding the Client Data Model's Support for Data Change Events

Describes the APIs that the MAF client data model uses to refresh the user interface of MAF applications in response to data change events in the underlying data collection.

The data and service classes that the MAF client data model generates provide ready-to-use support for both types of change events (property and provider) that MAF supports. This is because the `Entity` and `EntityCRUDService` classes that these types of classes extend from both extend from the `ChangeEventSupportable` class. For more information about the data change events that MAF supports, see Working with Data Change Events.

If you want to send a data change event from your data or service class, call the corresponding getter method to get an instance to send your change event:

- `getPropertyChangeSupport()`
- `getProviderChangeSupport()`

Always use the above getter methods rather than including your own instance of `propertyChangeSupport` or `providerChangeSupport` in a data or service class. This avoids breaking the built-in runtime code that the MAF client data model uses to refresh the user interface of your MAF application.

By default, each service class has a getter method that returns a list of data objects. A `DepartmentService` class, for example, includes the following generated method:

```
public List<Department> getDepartment() {
  return getEntityList();
}
```

When your MAF application makes a REST call to get the up-to-date list of departments, MAF automatically refreshes the user interface by invoking the `setEntityList` method from the `EntityCRUDService` superclass of the service class. This method calls another generated method in the service class (`getEntityListName`) to find out which property name to use in the `fireProviderRefresh` method call. The following example shows the generated `getEntityListName` method for a `DepartmentService` class:

```
protected String getEntityListName() {
  return "department";
}
```

The above implementation means that:

- If you write custom logic to change the content of a department list, you can call `setEntityList` to refresh the user interface with content changes.
- If you rename the generated method `getDepartment` to the more appropriate plural name `getDepartments`, you also need to change the method `getEntityListName` to return "`departments`" instead of "`department`". If you do not, the standard MAF client data model refresh code will no longer work.
- If you have added your own getter list methods to the service class to provide multiple filtered views on your set of data object instances, you can override the

`setEntityList` method to make sure your filtered lists also refresh in the user interface when a REST call completes. For more information, see Using Filtered Entity Lists.

Refresh Forms in Response to Data Change Events

To refresh the user interface in a form layout where only one data object instance displays, you need to use property change events because provider change events only refresh list views. While each data class includes the `PropertyChangeSupport` instance to send a property change event as explained above, the generated setter methods in your data object instances do not send change events by default. You are free to add this code yourself, as long as you use the `getPropertyChangeSupport()` method:

```
public void setName(String name)
{
  String oldValue = this.name;
  this.name = name;
  getPropertyChangeSupport().firePropertyChange("name", oldValue, name);;
}
```

If you want to refresh all properties (attributes) of a data object instance, you can also use the MAF convenience method `EntityUtils.refreshEntity`. This takes a data object instance as its only argument.

Send Data Changes in a Background Thread

When sending data change events in a background thread, you need to flush these data change events to the user interface layer by calling `AdfmfJavaUtilities.flushDataChangeEvent.` Furthermore, MAF requires you to use the `MafExecutorService` to prevent deadlocks when sending data change events in a background thread. The following example demonstrates how you use the execute method from the `MafExecutorService` to send data change events and flush the events once complete. The following example uses a Java 8 Lambda expression to pass in the code.

```
public void setName(String name) {
  String oldValue = this.name;
  this.name = name;
  if (AdfmfJavaUtilities.isBackgroundThread()) {
    MafExecutorService.execute(() ->
       {
         getPropertyChangeSupport().firePropertyChange("name", oldValue, name);
         AdfmfJavaUtilities.flushDataChangeEvent();

       });
  }
  else {
    getPropertyChangeSupport().firePropertyChange("name", oldValue, name);
  }
}
```

As an alternative to writing the above code for each attribute you want to refresh, you can instead use the `refreshUI` method provided by the MAF client data model's `Entity` class to send property change events. This method takes a list of attributes for which you want to send data change events, as the following example demonstrates:

```
List<String> attrs = new ArrayList<String>();
attrs.add("name");
attrs.add("managerId");
```

```
refreshUI(attrs);
```

The `refreshUI` method uses the previously-described `MafExecutorService` and flushes the data change events when running in a background thread.

Refresh the User Interface with Data Changes from a Child Data Collection

By default, the MAF client data model does not refresh a user interface with data changes to a child data collection when a MAF application makes a REST call that returns the parent data collection. This default behavior optimizes the performance of your application as it prevents a child data object being read from the database and loaded into memory when the page that triggers the REST call might only display the parent data object. You can extend a refresh to include a child data collection by overriding the `refreshUI` method from the `Entity` class in your data object's class. The following example demonstrates how you might do this in a `Department` class where you want to refresh a child data collection of employees:

```
@Override
public void refreshUI(List attrsToRefresh)
{
  getProviderChangeSupport().fireProviderRefresh("employees");
  super.refreshUI(attrsToRefresh);
}
```

6.21 Forcing Offline Mode in a MAF Application

MAF allows you to configure a MAF application as if the device it runs on is offline when it is connected to the Internet.

This can be helpful to prevent REST calls over slow network connections. For example, you might use it to prevent REST calls over 3G connections but allow them over Wi-Fi connections.

Use the following method to force offline mode:

```
new ConnectivityBean().setForceOffline(true);
```

> **Note:** While you create a new instance of `ConnectivityBean`, the value of `forceOffline` flag is saved in a static variable that is shared across all instances.

This allows you to use the `ConnectivityBean` class as a managed bean and force/unforce offline mode from the user interface. This might be helpful for demos where you want to show data synchronization capabilities of MAF. To do this, define the managed bean in the `adfc-mobile-config.xml` file as follows:

```
<managed-bean id="__3">
  <managed-bean-name>Connectivity</managed-bean-name>
  <managed-bean-class>oracle.maf.api.cdm.controller.bean.ConnectivityBean</managed-
bean-class>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

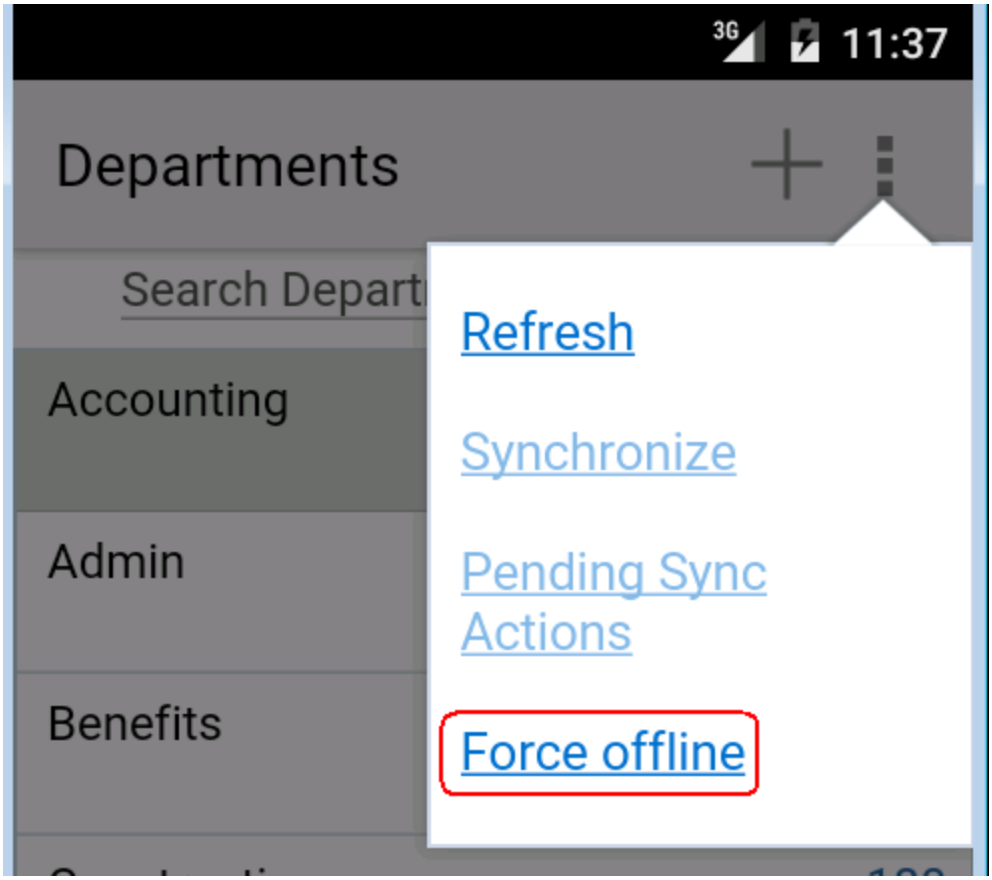
You can then add a "force offline" toggle option in your AMX pages as follows:

```
<amx:commandLink id="menFo" text="#{Connectivity.forceOffline ? 'Unforce offline' :
'Force offline'}">
    <amx:setPropertyListener id="menfospl" from="#{!Connectivity.forceOffline}"
```

```
to="#{Connectivity.forceOffline}"/>
</amx:commandLink>
```

The MAF User Interface Generator wizard, described in How to Use the MAF User Interface Generator, generates a menu entry that displays this option, as shown in Figure 6-25.

Figure 6-25 Force Offline Menu Entry from MAF User Interface Generator



If you want to force offline mode based on the strength of the network connection, use the Cordova network plugin which is pre-installed with MAF to set up a JavaScript callback handler that calls the `ConnectivityBean.forceOffline` method.

6.22 Using a Visual Indicator for Running Background Tasks

Describes how to render a visual indicator to end users to let them known that their MAF application is performing background tasks.

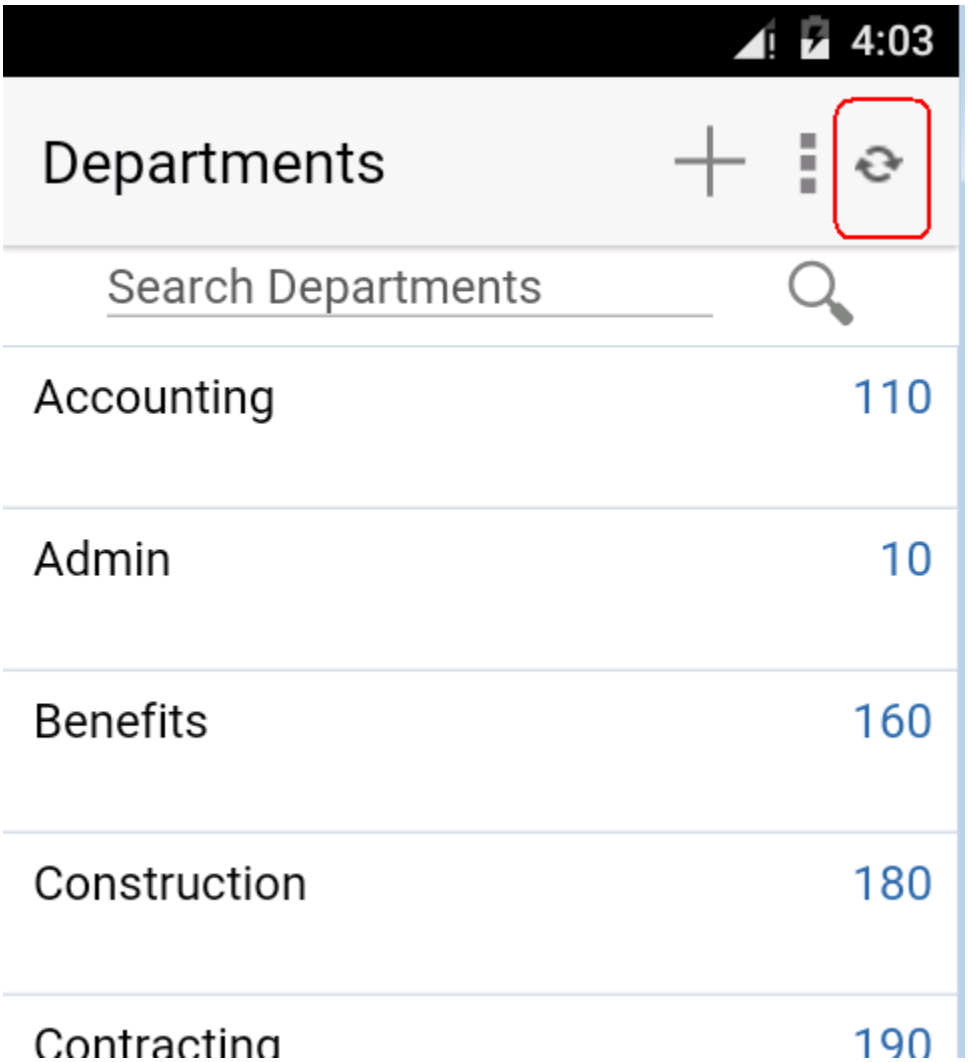
MAF makes all background REST calls through a thread pool that maintains a Boolean flag to return `True` if the MAF application is executing at least one task in the thread pool. Use the following EL expression to access the value of this Boolean flag:

```
#{applicationScope.maf_bgtask_running}
```

The MAF User Interface Generator wizard, described in How to Use the MAF User Interface Generator, generates the following entry in the AMX pages it creates to display the visual indicator, shown in Figure 6-26, when the MAF application processes tasks in the background.

```
  <amx:image id="bgRunImg" source="/images/reloading.gif" inlineStyle="margin-right:
5px;"
             rendered="#{applicationScope.maf_bgtask_running}"/>
```

Figure 6-26 Visual Indicator for Background Tasks from MAF User Interface Generator



7

Using Oracle Mobile Cloud Service Platform APIs in a MAF Application

This chapter describes how to use Oracle Mobile Cloud Service (MCS) platform APIs, such as the Storage API, update user information, plus send analytics and diagnostic events to MCS for analysis by the Analytics and Diagnostic features of MCS.

This chapter includes the following sections:

- Introduction to Using Oracle Mobile Cloud Service Platform APIs
- Accessing Oracle Mobile Cloud Service User Information
- Accessing Files in an Oracle Mobile Cloud Service Storage Collection
- Sending Analytics Information to Oracle Mobile Cloud Service
- Sending Diagnostic Information to Oracle Mobile Cloud Service

7.1 Introduction to Using Oracle Mobile Cloud Service Platform APIs

MAF provides support for MCS platform APIs such as the Storage API and the Analytics API. MAF also supports updating MCS user information and sending diagnostic information to MCS Diagnostics.

The first step to use any of these platform APIs is to connect your MAF application to MCS. To do this, you select the **MCS Connection** checkbox and specify the mobile backend ID and anonymous access key when you connect to the service to create the client data model. For more information about this task, see Connecting to a REST Service to Create the Client Data Model. Once you complete the wizard where you specify the connection details, JDeveloper writes the following entries to your application's `mobile-persistence-config.properties` file:

```
# MCS connection details, applicationScope EL Expressions
# are allowed for backend ID and anonymous key
mcs.connection=MCS
mcs.mobile-backend-id=bcda8418-8c23-4d92-b656-9299d691e120
mcs.anonymous-key=c3RldmVuLmtpbmc6U3RhYWYyMDE1IQ==
```

The access key creates the Authorization header when your application accesses an MCS platform API or custom API before you have authenticated with MCS. After you authenticate against MCS, MAF automatically injects the Authorization header into every REST call based on the user login credentials. That is, it ignores the anonymous access key that is specified in the `mobile-persistence-config.properties` file. For more information about this key, see What You May Need to Know About the MCS Anonymous Access Key.

The `oracle.maf.api.cdm.persistence.manager.MCSPersistenceManager` class handles all calls to MCS platform APIs. By default it uses the settings in the

`mobile-persistence-config.properties` file, but you can override the settings in this file by calling the following methods in custom code:

- `setConnectionName(String connectionName)`
- `setMobileBackendId(String mobileBackendId)`
- `setAnonymousKey(String anonymousKey)`
- `setAuthHeader(String authHeader)`
- `login(String userName, String password)`

If your application needs to support dynamic MCS connections, you can specify an EL expression for the `mcs.mobile-backend-id` and `mcs.anonymous-key` values in the `mobile-persistence-config.properties` file. The actual URL of the MCS connection can be changed at runtime using the `AdfmfJavaUtilities.overrideConnectionProperty` method. The following sample code demonstrates how to dynamically set the MCS connection URL based on a user preference:

```
String mcsHost =
(String)AdfmfJavaUtilities.evaluateELExpression("#{preferenceScope.application.connec
tion.host}");
AdfmfJavaUtilities.clearSecurityConfigOverrides("MCS");
AdfmfJavaUtilities.overrideConnectionProperty("MCS", "restconnection", "url", mcsHost
+"/mobile");
```

You typically include this code in your application lifecycle listener `start()` method. For more information, see Using Lifecycle Listeners in MAF Applications . For more information about using the `AdfmfJavaUtilities.overrideConnectionProperty` method, see How to Update Connection Attributes of a Named Connection at Runtime.

7.2 Accessing Oracle Mobile Cloud Service User Information

MAF's `MCSPersistenceManager` provides methods to log users in and out of MCS plus retrieve and update user information.

Use the following method to do a programmatic login against MCS:

```
String response = new MCSPersistenceManager().login(userName, password);
```

This returns the response payload from MCS. Although we describe this method for your information, we recommend that you use MAF's declarative support for user authentication. The latter option allows you to use basic authentication or OAuth. It also provides options to remember the user name and password in a secure way. In addition, by securing a MAF feature the login screen automatically appears when you access a secured feature for the first time.

Use the following method to do a programmatic logout from MCS:

```
String response = new MCSPersistenceManager().logout();
```

Again, we recommend you use MAF's declarative support for securing your application. When you use MAF's declarative support, you should use the following MAF API call to log out:

```
AdfmfJavaUtilities.logout();
```

For more information about MAF's declarative support for user authentication, see Securing MAF Applications .

Use the following method to retrieve all the attributes of a user stored in MCS:

```
String response = new MCSPersistenceManager().findUser(userName);
```

It returns a response payload like this:

```
{
  "id": "a8acf41f-50a7-473e-8376-d6346cf188be",
  "username": "GEVANS",
  "email": "george.evans@ebsss.com",
  "firstName": "George",
  "lastName": "Evans",
  "jobTitle": "Cloud Solutions Architect",
  "links": [
    {
      "rel": "canonical",
      "href": "/mobile/platform/users/GEVANS"
    },
    {
      "rel": "self",
      "href": "/mobile/platform/users/GEVANS"
    }
  ]
}
```

Note: Custom attributes that you defined for your user realm in MCS are also returned. In the above example, `jobTitle` is an example of a custom attribute.

Update user attributes with the following method:

```
String response = new MCSPersistenceManager().updateUser(userName, payload);
```

The payload you pass is a list of the attributes you want to change, for example:

```
{
  "email" : "gevans@ebsss.com",
  "jobTitle": "Senior Cloud Solutions Architect"
}
```

7.3 Accessing Files in an Oracle Mobile Cloud Service Storage Collection

Describes how to retrieve, download and filter storage objects from an MCS storage collection in a MAF application.

MAF provides both programmatic and declarative access to the MCS Storage API. To use declarative access, you create a bean data control from `oracle.maf.api.cdm.mcs.storage.StorageObjectService`. The generated data control provides a range of operations to access files in a storage collection. You can also access the `StorageObjectService` class programmatically. In addition to retrieving files, you can create and update files in an MCS storage collection and you can associate MCS storage files with your data objects. For example, you could create a list page showing employees with a small thumbnail employee image coming from MCS that navigates to an employee form page with the picture that an end user can update by taking a new picture. Offline support is fully implemented as the storage object metadata and file are stored on the device. Storage objects can be created and

updated in offline mode. MAF synchronizes these objects with MCS once the device returns online, as described in Synchronizing Offline Transactions from a MAF Application.

For more information about implementing the just-described use cases, see:

- How to Create the StorageObjectService Bean Data Control
- What Happens When You Create the StorageObject Bean Data Control
- How to Retrieve All Files from an Oracle Mobile Cloud Service Storage Collection
- How to Filter the List of Storage Objects from an MCS Storage Collection
- How to Retrieve a Single File from an Oracle Mobile Cloud Service Storage Collection
- How to Associate Storage Objects with Data in your MAF Application

7.3.1 How to Create the StorageObjectService Bean Data Control

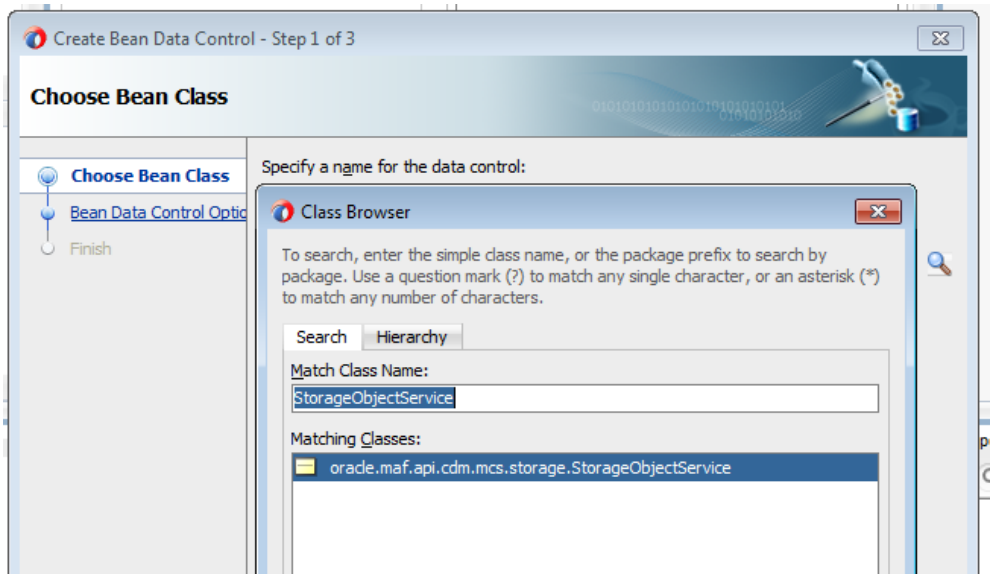
Describes how to create and use the `StorageObjectService` data control that you can use to retrieve and manage objects in an MCS storage collection.

To create the `StorageObjectService` bean data control:

1. In the Applications window, select any existing project and in the main menu, choose **File** and then **New** > **From Gallery**.
2. In the New Gallery, expand **Business Tier**, select **Data Controls** and then **Bean Data Control**, and click **OK**.
3. In the Choose Bean Class dialog, click the search icon to display the Class Browser dialog and enter `StorageObjectService` in the **Match Class Name** input field.

   JDeveloper retrieves the `oracle.maf.api.cdm.mcs.storage.StorageObjectService` class, as shown in Figure 7-1.

Figure 7-1 StorageObjectService in Class Browser



4. Click **OK**.

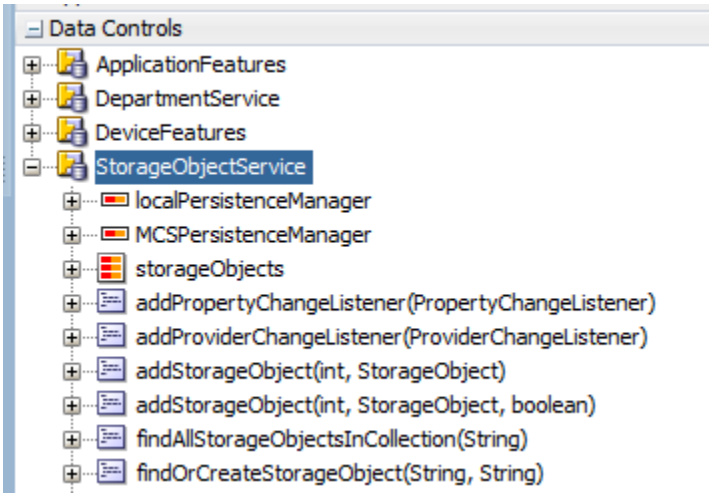
   JDeveloper specifies `StorageObjectService` as the name for the data control in the Choose Bean Class dialog.

5. Click **Next** and **Finish**.

7.3.2 What Happens When You Create the StorageObject Bean Data Control

JDeveloper generates the `StorageObjectService` data control in the Data Controls panel.

Figure 7-2 Generated StorageObjectService Data Control



You typically use the methods that this data control exposes to download and display a collection of files or a single file. The REST calls that your MAF application makes to MCS to retrieve storage objects execute in the background when the `StorageObject` mapping descriptor's `remoteReadInBackground` property is set to `true`, as shown in the following example from the `persistence-mapping.xml` file. Similarly, REST calls from your MAF application to create or update storage objects in MCS execute in the background when the `remoteWriteInBackground` property is set to `true`.

You cannot modify these properties using the Edit Persistence Mappings wizard. Instead, you edit these properties directly in the `persistence-mapping.xml` file that is in the `ApplicationController\src\META-INF` directory of your application's workspace.

```
   <classMappingDescriptor className="oracle.maf.api.cdm.mcs.storage.StorageObject"
persisted="true">
      <crudServiceClass
className="oracle.maf.api.cdm.mcs.storage.StorageObjectService"
autoIncrementPrimaryKey="true"

localPersistenceManager="oracle.maf.api.cdm.persistence.manager.DBPersistenceManager"


remotePersistenceManager="oracle.maf.impl.cdm.persistence.manager.MCSStoragePersisten
ceManager"
                        remoteReadInBackground="true"
                        remoteWriteInBackground="true"
                        showWebServiceInvocationErrors="true"
                        autoQuery="false"
                        enableOfflineTransactions="true"/>
```

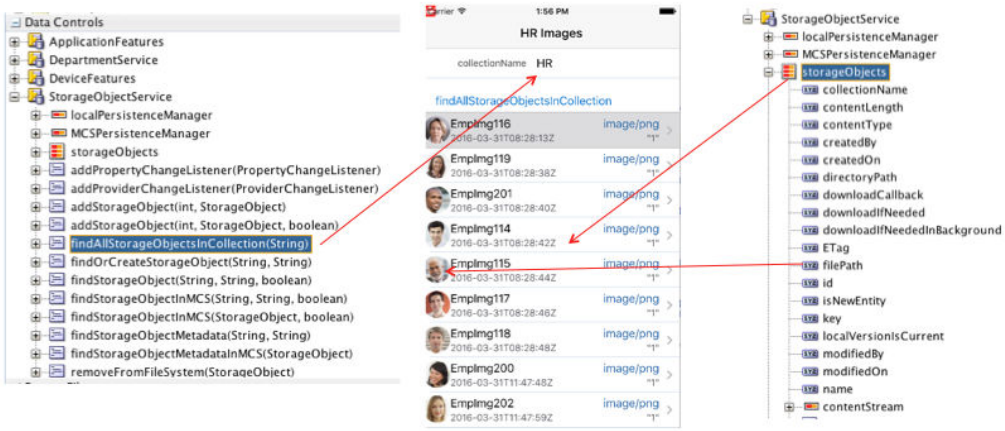
```
<table name="STORAGE_OBJECT">
   <primaryKeyColumn name="ID"/>
   <primaryKeyColumn name="COLLECTION_NAME"/>
</table>
<attributeMappings>
```

7.3.3 How to Retrieve All Files from an Oracle Mobile Cloud Service Storage Collection

Describes how to use the method that the `StorageObjectService` data control exposes to retrieve all storage objects from MCS.

Figure 7-3 illustrates how the `findAllStorageObjectsInCollection(String)` method from the StorageObjectService data control can be used to render thumbnail images for each employee in a collection of employees. Drag and drop this method to an AMX page and choose **MAF Parameter Form** in the context menu that appears. This creates an input field for the collection name and a button to execute the method. After the method executes, the data control's storageObjects collection populates with the content of the collection. You can then drag and drop the storageObjects collection onto your AMX page as, for example, a MAF List View.

Figure 7-3 Image Files from an MCS Storage Collection in a List View



Alternatively, you can drag and drop the findAllStorageObjectsInCollection method as a method activity onto a task flow and then add a control flow rule from this activity to your AMX page that shows the content of the collection. When you perform this drag and drop, you need to provide a value for the `collectionName` parameter. The value you provide can be hardcoded or an EL expression.

The storageObjects collection is initially populated with the local storage objects from the SQLite database, the call to MCS executes in the background, and the UI refreshes once the MCS results return.

Figure 7-3 shows how the storage file is used to display the image in each list item. To accomplish this, set the source attribute of the `amx:image` element to the value of the storage object's `filePath` attribute. The `filePath` attribute holds the reference to the downloaded file on the mobile device.

```
<amx:image source="#{row.filePath}" inlineStyle="width:40px;height:40px;"  id="i1"/>
```

The `findAllStorageObjectsInCollection` method returns the metadata of each storage object. In order to download the image files, drag and drop the storage object's `downloadIfNeededInBackground` attribute as an `outputText` element. At runtime, the MAF application calls the `getDownloadIfNeedInBackground` method on the `StorageObject` class. This method returns an empty string and triggers the

download of the image files. If the file has been downloaded before, MAF first makes a `HEAD` call to check if the storage object's `ETag` in MCS has the same value as the local `ETag` stored in the SQLite database. If the values differ, MAF downloads the file. The following sample shows the `listView` component's code that renders the list view shown in Figure 7-3.

If you do not want the page to load before MAF downloads all image files, drag and drop the `downloadIfNeeded` attribute instead.

```
<amx:listView var="row" value="#{bindings.storageObjects.collectionModel}"
              fetchSize="#{bindings.storageObjects.rangeSize}"

selectedRowKeys="#{bindings.storageObjects.collectionModel.selectedRow}"

initialScrollRowKeys="#{bindings.storageObjects.collectionModel.selectedRow}"

selectionListener="#{bindings.storageObjects.collectionModel.makeCurrent}"
              showMoreStrategy="autoScroll" bufferStrategy="viewport" id="lv1">
  <amx:listItem id="li1" rendered="#{row.contentType=='image/jpeg'}">
    <amx:tableLayout width="100%" id="tl1">
      <amx:rowLayout id="rl2">
        <amx:cellFormat width="40px" halign="center" rowSpan="2" id="cf6">
          <amx:image source="#{row.filePath}" inlineStyle="width:40px;height:40px;"
id="i1"/>
          <amx:outputText value="#{row.downloadIfNeededInBackground}" id="ot6"/>
        </amx:cellFormat>
        <amx:cellFormat width="60%"
height="#{deviceScope.device.os=='Android'?'36':'32'}px" id="cf4">
          <amx:outputText value="#{row.name}" id="ot4"/>
        </amx:cellFormat>
        <amx:cellFormat width="10px" rowSpan="2" id="cf3"/>
        <amx:cellFormat width="40%" halign="end" id="cf5">
          <amx:outputText value="#{row.contentType}" styleClass="adfmf-listItem-
highlightText" id="ot5"/>
        </amx:cellFormat>
      </amx:rowLayout>
      <amx:rowLayout id="rl1">
        <amx:cellFormat width="60%"
height="#{deviceScope.device.os=='Android'?'22':'19'}px" id="cf1">
          <amx:outputText value="#{row.createdOn}" styleClass="adfmf-listItem-
captionText" id="ot2"/>
        </amx:cellFormat>
        <amx:cellFormat width="40%" halign="end" id="cf2">
          <amx:outputText value="#{row.ETag}" styleClass="adfmf-listItem-
captionText" id="ot3"/>
        </amx:cellFormat>
      </amx:rowLayout>
    </amx:tableLayout>
  </amx:listItem>
</amx:listView>
```

7.3.4 How to Filter the List of Storage Objects from an MCS Storage Collection

Describes how to filter the list of storage objects that the StorageObjectService retrieves from an MCS storage collection.

The `StorageObjectService` API supports the retrieval of all objects in an MCS storage collection or just one object. You can filter the results on the device by using the SQLite database that stores all the storage object metadata. To do this, create a subclass that extends from `oracle.maf.api.cdm.mcs.storage.StorageObjectService`. Write code in

this subclass that filters the retrieved storage objects like the code you write in the service object classes to filter data objects in your MAF application, as described in Using Filtered Entity Lists.

Generate a data control from the subclass you created that extends from `oracle.maf.api.cdm.mcs.storage.StorageObjectService`.

7.3.5 How to Retrieve a Single File from an Oracle Mobile Cloud Service Storage Collection

Describes how to use the `findStorageObject` method that the StorageObjectService data control exposes to retrieve and download a single storage object from MCS.

The `findStorageObject` method takes the following two parameters:

- Name of the collection in MCS
- ID of the storage object

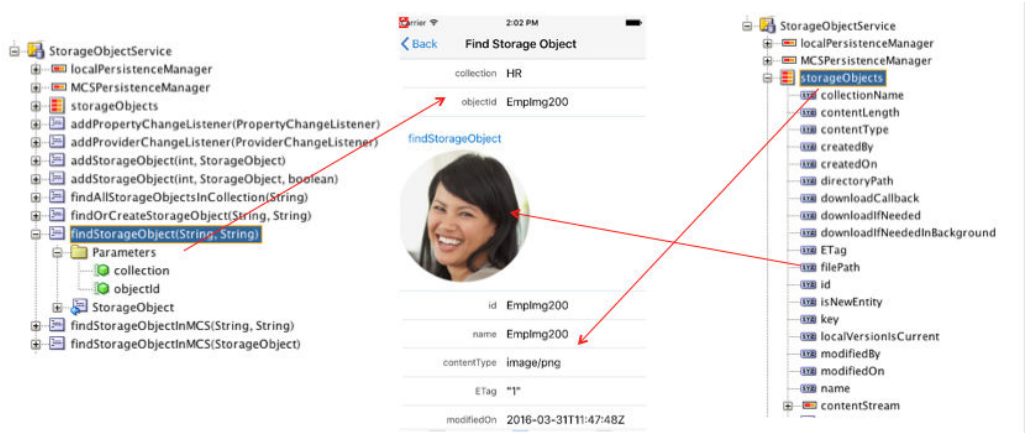
Drag and drop the `findStorageObject` method as a MAF Parameter Form to create input fields for the two parameters and a button to execute the method. You can drag and drop the result `StorageObjects` element of this method onto your AMX page as, for example a MAF Read-Only Form.

If the storage object that you download is an image, drag and drop the `filePath` attribute as an `outputText` component and then change it into an image component, as shown in the following example:

```
<amx:image id="i1" source="#{bindings.filePath.inputValue}" inlineStyle="height:
200px;"/>
```

Figure 7-4 shows an AMX page where the `findStorageObject` method downloaded an image file to render in the UI of the MAF application.

Figure 7-4 findStorageObject Method Downloading an Image File



7.3.6 How to Associate Storage Objects with Data in your MAF Application

Describes how to write Java code that uses the `StorageObjectService` and `StorageObject` classes to retrieve a storage object from an MCS storage collection and associate it with a data object in your MAF application.

Enriching existing data from a backend system of record with data that can be captured using a MAF application's device capabilities, such as a camera, is a frequent requirement when building MAF applications. The following code samples describe

how you can associate employee data with a picture that is retrieved from an MCS storage collection.

You can use the default constructor to create an instance of the `StorageObjectService` class or, alternatively, the constructor that enables you to override the values for `remoteReadInBackGround` and `remoteWriteInBackground` set in your application's `persistence-mapping.xml` file. This is easier in Java code so you directly work with the results of the REST calls.

The following code example demonstrates how to get all storage objects in an MCS storage collection using the latter constructor:

```
StorageObjectService sos = new StorageObjectService(false,false);
sos.findAllStorageObjectsInCollection("HR");
List<StorageObject> storageObjects = sos.getStorageObjects();
```

The following code samples show the Java code to add an employee's picture to an employee list view and form page where the employee data comes from a custom MCS API and an MCS storage collection ("HR") stores the picture(s). The code sample also demonstrates how to add a picture for a new employee using the device's camera.

We use the following naming convention for employee pictures to associate pictures with the correct employee(s):

```
EmpImg[EmployeeID suffix]
// Example: EmpImg100
```

First, add an image property of type `StorageObject` to the `Employee` data object, with the following getter and setters methods:

```
private StorageObject picture;

public StorageObject getPicture() {
  if (picture == null) {
    StorageObjectService sos = new StorageObjectService(false, false);
    picture = sos.findOrCreateStorageObject("HR", "EmpImg" + getId());
    picture.setDownloadCallback(() -> {
        refreshUI(Arrays.asList(new String[]{"picture"}));
      });
    picture.getDownloadIfNeededInBackground();
  }
  return picture;
}

public void setPicture(StorageObject picture) {
  this.picture = picture;
}
```

In the getter method we check whether the picture `StorageObject` is still null. If so, we instantiate the `StorageObjectService` and then call the `findOrCreateStorageObject` method where we pass in the name of the MCS collection and the ID of the employee picture file in the MCS collection. This method queries the local on-device database and returns the storage object if there is a matching row in the `STORAGE_OBJECT` table. If no row is found, a new `StorageObject` instance is created with the collection name and object ID set. We then specify a callback (`setDownloadCallback(() ->`) that allows us to refresh the UI once the employee picture downloads. Finally, we call the

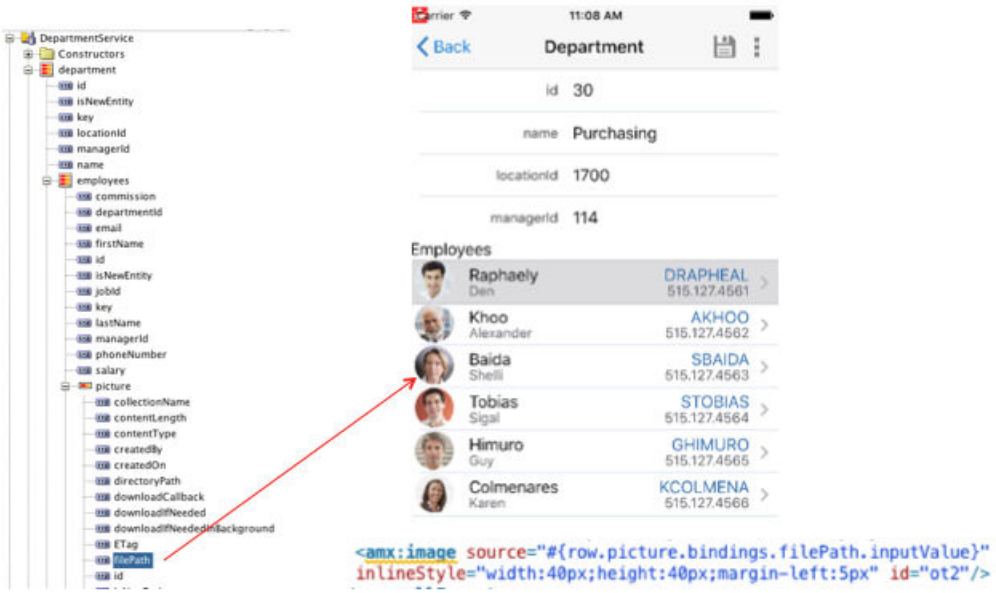
`getDownloadIfNeededInBackground` method which calls MCS to download the file and update the storage object metadata if needed.

> **Note:** If the file has been downloaded before, MAF first makes a `HEAD` call to check whether the `ETag` of the storage object in MCS has the same value as the local `ETag` value stored in the SQLite database. If the value has changed, the file will be downloaded.

The download location of the file on the device is a subdirectory named `/MCS/[collectionName]` under the application directory (the value returned by `AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.ApplicationDirectory)`). The name of the file is the name of the storage object in MCS, or the ID if the name is not set in MCS. You can override the default download directory by calling the StorageObject's `setDirectoryPath(String directoryPath)` method.

Now that you have added the picture `StorageObject` to the Employee class, you can drag and drop the picture `filePath` property to add an image to the list of employees within a department, as illustrated in Figure 7-5.

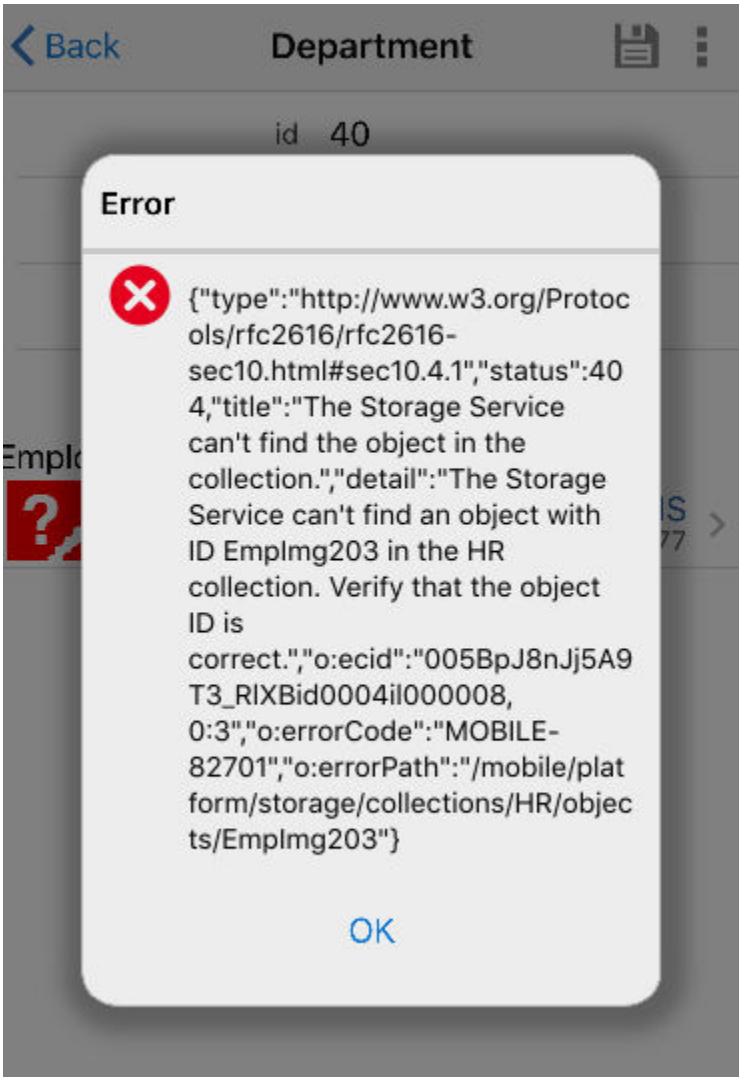
Figure 7-5 Adding an Image to a ListView



When you navigate to the Department page for the first time, you will see a slight delay before the employee pictures appear. For each employee in the department a parallel REST call is made in the background to fetch the picture from MCS. The next time you run the application, the images appear immediately.

By default, an error message appears when an employee does not yet have a picture in MCS because the REST call results in a 404 Not Found response, as shown in Figure 7-6.

Figure 7-6 Web Service Error Message



To avoid end users seeing web service invocation error messages, such as that shown in Figure 7-6, set the `showWebServiceInvocationErrors` property in the `persistence-mapping.xml` to `false` for the `StorageObject` descriptor, as shown by the following example.

```
<classMappingDescriptor className="oracle.maf.api.cdm.mcs.storage.StorageObject"
persisted="true">
  <crudServiceClass className="oracle.maf.api.cdm.mcs.storage.StorageObjectService"
             autoIncrementPrimaryKey="true"

localPersistenceManager="oracle.maf.api.cdm.persistence.manager.DBPersistenceManager"

remotePersistenceManager="oracle.maf.impl.cdm.persistence.manager.MCSStoragePersisten
ceManager"
             remoteReadInBackground="true" remoteWriteInBackground="true"
             showWebServiceInvocationErrors="false"
             autoQuery="false" enableOfflineTransactions="true"/>
```

To upload a new employee picture, add a `commandLink` component to the employee form page that calls a managed bean method that takes a picture using the camera or

selects an existing picture from the picture gallery. The following example demonstrates a possible implementation of such a managed bean method:

```
public void takePicture(ActionEvent actionEvent) {
  DeviceManager device = DeviceManagerFactory.getDeviceManager();
  int cameraType =
    device.hasCamera() ? DeviceManager.CAMERA_SOURCETYPE_CAMERA:
DeviceManager.CAMERA_SOURCETYPE_PHOTOLIBRARY;
  String picture = device.getPicture(50,
DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URI, cameraType, false,
                          DeviceManager.CAMERA_ENCODINGTYPE_PNG, 520, 380);
  Employee emp =
    (Employee)
AdfmfJavaUtilities.getELValue("#{bindings.employeesIterator.currentRow.dataProvider}"
);
  // the camera returns an URI starting with file://, we remove this prefix to get
proper file path
  emp.addPicture(picture.substring(7));
}
```

After taking the picture, obtain the current instance of employee and call the `addPicture` method on this instance, as demonstrated in the following example:

```
public void addPicture(String tempFilePath) {
  StorageObjectService sos = new StorageObjectService();
  StorageObject so = sos.findOrCreateStorageObject("HR", "EmpImg" + getId());
  so.setContentType("image/png");
  try {
    so.setContentStream(new FileInputStream(tempFilePath));
    sos.saveStorageObject(so);
    setPicture(so);
    // remove picture from temporary camera directory
    new File(tempFilePath).delete();
  }
  catch (FileNotFoundException e) {
    sLog.severe("Employee picture file not found");
  }
}
```

Similar to the `getPicture` method, `addPicture` first gets a `StorageObject` instance by calling the `findOrCreateStorageObject` method. We then set the `contentType` and `contentStream` and call the `saveStorageObject` method on the `StorageObjectService`. This method saves the storage object metadata in the SQLite database, streams the picture to the file system and calls MCS to add the file to the HR MCS storage collection. After the file is saved, you can remove the temporary file created by the device camera to free up disk space.

> **Note:** Instead of having the camera return a file URI, you can also obtain the image as a base64 encoded string by using `DeviceManager.CAMERA_DESTINATIONTYPE_DATA_URL` as the value for `destinationType`. This is not recommended because it loads the entire photo into memory and can easily crash your mobile application with an `OutOfMemory` error when taking high quality photos.

To remove an employee picture, add a `removePicture` method to the Employee class and drag and drop this method onto the Employee form page as a button or link.

```
public void removePicture() {
  StorageObjectService sos = new StorageObjectService();
  StorageObject so = sos.findOrCreateStorageObject("HR", "EmpImg" + getId());
  sos.removeStorageObject(so);
  // get new "empty" storageObject and refresh UI
  so = sos.findOrCreateStorageObject("HR", "EmpImg" + getId());
  setPicture(so);
  refreshUI(Arrays.asList(new String[]{"picture"}));
}
```

The `removeStorageObject` method deletes the storage object row in the SQLite database, removes the file from the file system and calls MCS to remove the file from the MCS collection.

For more information about the `StorageObject` and `StorageObjectService`, see the *Java API Reference for Oracle Mobile Application Framework*.

7.4 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.

JDeveloper'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_1601211149
22"

"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_1601211149
22"

"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_1601211149
22"
```

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:

- How to Configure the Transfer of Analytics to Oracle Mobile Cloud Service
- How to Programmatically Send Analytics to Oracle Mobile Cloud Service
- How to Send Context Events to Oracle Mobile Cloud Service
- How to Send Analytics to Other Repositories
- 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*.

7.4.1 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.api.
analytics.McsContextProvider
```

Of the properties listed in the previous example, only `connectionId` is mandatory. Table 7-1 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 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 7-1 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`

7.4.2 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
```

7.4.3 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 7-2 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 7-2 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.contextProviderClas
sName= oracle.maf.api.analytics.McsContextProvider
```

With this entry configured in your `logging.properties` file, MAF sends the context event information listed in Table 7-2 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.contextProviderClas
sName=oracle.maf.demo.CustomContextProvider
```

Where `oracle.maf.demo.CustomContextProvider` is a class that implements `oracle.maf.api.analytics.ContextProvider`, as shown in Example 7-1. The generated context event contains the timezone, carrier, and manufacturer information.

Example 7-1 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;
  }
  }
```

7.4.4 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.McsCo
ntextProvider
```

7.4.5 How to Add Location Information to Events

By default, client data model does not include latitude and longitude of the current device location in the context information of each event. To add this information, you should set `analytics.use.location` property in `mobile-persistence-config.properties` to true.

You need to set the property, as follows:

```
analytics.use.location=true
```

7.4.6 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 7-3 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 7-3 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
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

Table 7-3 (Cont.) MAF Framework Events (Custom Category)

Event Name	Logging Level	Enable or Disable Logging
`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 7-4 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 7-4 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 not.
`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 How to Send Context Events to Oracle Mobile Cloud Service

7.5 Sending Diagnostic Information to Oracle Mobile Cloud Service

Describes how to insert diagnostic information from MAF applications that access REST web services hosted by Oracle Mobile Cloud service (MCS).

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();
```

For more information about creating an adapter, see Creating a Rest Service Adapter to Access Web Services.

8

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 JDeveloper provides to create and edit resource bundles.

This chapter includes the following sections:

- Introduction to MAF Application Localization
- Setting Resource Bundle Options for a MAF Application
- Defining Text Resources in a Base Resource Bundle
- Creating Locale-Specific Resource Bundles
- Editing Resources in Resource Bundles
- Localizing Image Files in a MAF Application
- MAF Support of Languages
- Localizable MAF Properties

8.1 Introduction to MAF Application Localization

Localization is the process that adapts a product to a specific locale. A localized MAF application uses the same language as the mobile device on which it runs. For example, if you set French as the device language, the text resources of the application appear in French.

To help you localize an application, MAF provides design-time menus in one or more resource bundles. Use these menus to define the text resources that the user interface must display. Define your MAF application's text resources in a base resource bundle. If your application has no locale-specific resource bundle for the device's locale, the application on the user's device renders these text resources on its user interface. Create locale-specific resource bundles for each locale that you want supported. In these locale-specific resource bundles, provide translations of the text resources that you had defined in the base resource bundle.

Figure 8-1 shows an example where an application renders `commandButton` and `outputText` components. The language that you set on the mobile device decides the text and image resources that appear in the components. On the left, the components render a text resource in English because the base resource bundle contains text resources with English values. 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. The MAF application contains a locale-specific resource bundle (`_fr`) with translations of the values in the base resource bundle. French is set as the mobile device's language.

Figure 8-1 Localized Text Resources

Use the following steps to localize your MAF application:

1. Find the number of resource bundles in your MAF application, as described in Setting Resource Bundle Options for a MAF Application.
2. Define the text resources in your base resource bundle so that they are rendered when your MAF application runs in a locale for which no a locale-specific resource bundle is provided.
3. Create locale-specific resource bundles where you provide translations of the text resources that you had defined in the base resource bundle.
4. Create locale-specific versions of image files and other resources that are needed to localize your MAF application.

8.2 Setting Resource Bundle Options for a MAF Application

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

A MAF application has two default resource bundles:

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

The naming convention for the application-level resource bundle uses the application name and appends `Bundle.xlf`. So, a MAF application named `MyLocalizedMAFapp` has an application-level resource bundle named `MyLocalizedMAFappBundle.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 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.

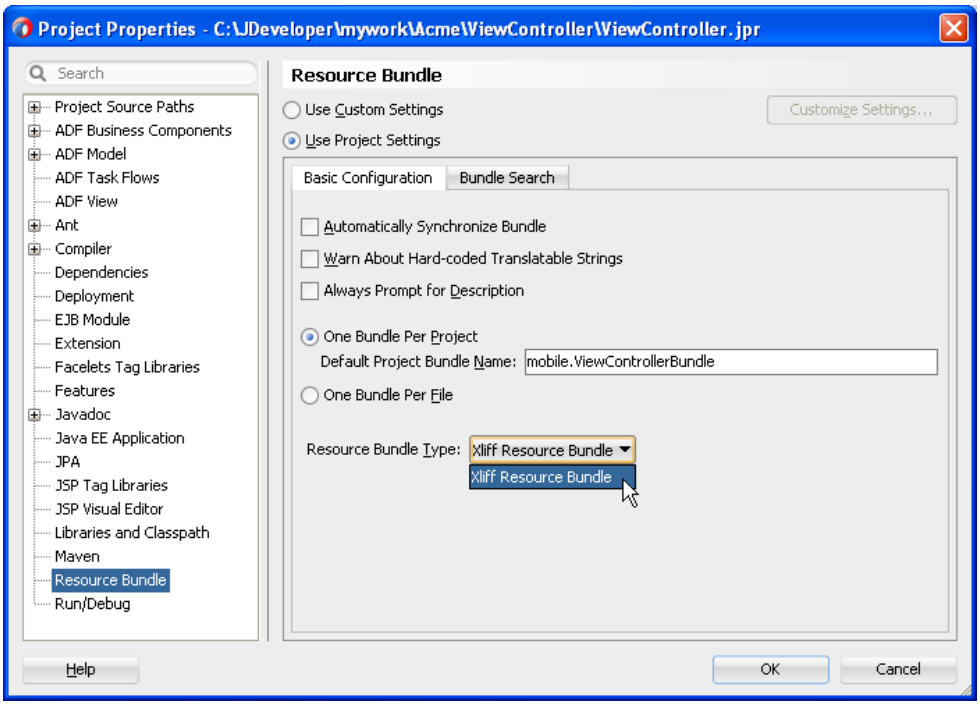
8.2.1 How to Set the Resource Bundle Options for a MAF Application

You set the resource bundle options for the view controller and application controller projects using the Resource Bundle page in the Project Properties dialog. Options that you can set include the number of project-level resource bundles that JDeveloper creates.

To set the resource bundle options for a project:

1. In the Applications window, double-click the project.
2. In the Project Properties dialog, select **Resource Bundle** to view the Resource Bundle page, as shown in Figure 8-2.

Figure 8-2 Project Properties Resource Bundle Page



3. Select **Always Prompt for Description** if you want to prevent the Select Text Resource dialog from closing before you enter a description of the text resource in the dialog.
4. Select one of the following resource bundle file options:
   - **One Bundle Per Project** —JDeveloper creates one resource bundle in a file named `<ProjectName>.xlf`.
   - **One Bundle Per File**— Creates a resource bundle each time you define a text resource for a file (`maf-feature.xml`, `maf-application.xml`, or an `.amx` file). This option limits the number of resource bundles to one per file; if you select this option, JDeveloper prevents you from creating a second bundle.
5. Click **OK**.

8.3 Defining Text Resources in a Base Resource Bundle

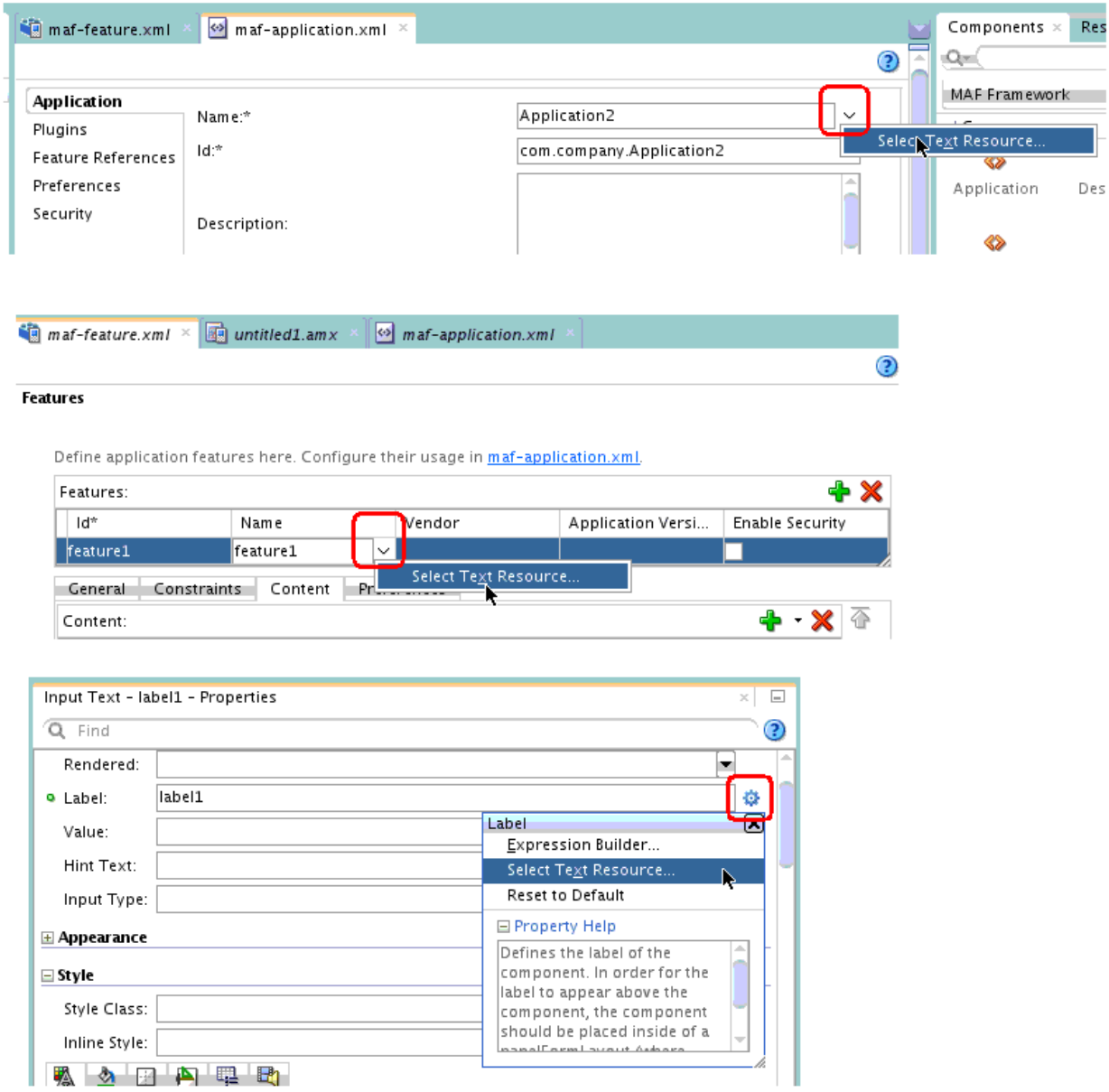
JDeveloper provides the Select Text Resource dialog where you can define values in resource bundles for a MAF application, application feature, and MAF AMX UI component text resources that appear to end users.

Access the Select Text Resource dialog by clicking the icon beside the property for which you want to define a text resource in the Properties window, and selecting Select Text Resource. 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 Localizable MAF Properties.

Figure 8-3 demonstrates how you display the Select Text Resource context menu for a MAF application's name, an application feature's name and an inputText component's Label property.

Figure 8-3 Context Menu to Display Select Text Resource Dialog



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

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

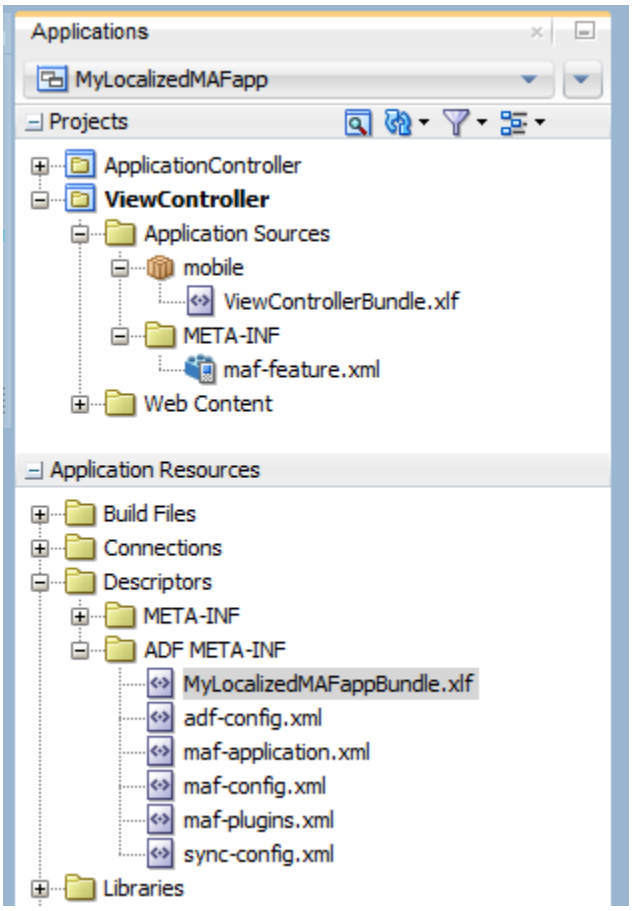
To define a text resource in a resource bundle:

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.
2. In the Properties window, click the icon beside the property for which you want to define a text resource and select **Select Text Resource** in the context menu that appears.
3. In the Select Text Resource dialog, define a text resource by entering a display name, key, and then clicking **Save and Select**.

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

JDeveloper writes the display value and key that you enter to a resource bundle. When you first define a text resource for a MAF application or project, JDeveloper 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), JDeveloper creates an application-level resource bundle (*`ApplicationName`*`Bundle.xlf`). You can view the bundle in the Application Resources panel, as shown in Figure 8-4. If you define text resources for a project-level property (for example, an application feature's name property), JDeveloper creates a project-level resource bundle (`ViewControllerBundle.xlf`). You can view the bundle in the Projects panel, as shown in Figure 8-4.

Figure 8-4 Newly-Created Resource Bundles



JDeveloper creates one application-level resource bundle for a MAF application. It may create additional project-level resource bundles depending on the options that you set, as described in Setting Resource Bundle Options for a MAF Application.

The XML syntax that JDeveloper writes to the resource bundle for the key and display value is the same for application or project-level resource bundles, 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>
...
```

Text resources are defined in resource bundles. Configurable properties reference the text resources. JDeveloper makes the changes shown in Example 8-1 for the properties files.

- When you localize an attribute the first time, JDeveloper adds an `<adfmf:loadbundle>` element. The `basename` attribute of the element refers to the newly created resource bundle.
- JDeveloper changes the localized attribute string to an EL expression. The EL expression refers to the key of the text resource that is defined in the resource bundle.
- JDeveloper adds every additional localized string to the same resource bundle file. Only one resource bundle file exists at the application level. At the project level, the behavior depends on the resource bundle settings that you selected in Setting Resource Bundle Options for a MAF Application.

Example 8-1 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}"/>
```

8.4 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 8-5 shows a MAF application where the locale-specific versions of the application-level (`MyLocalizedMAFappBundle.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:

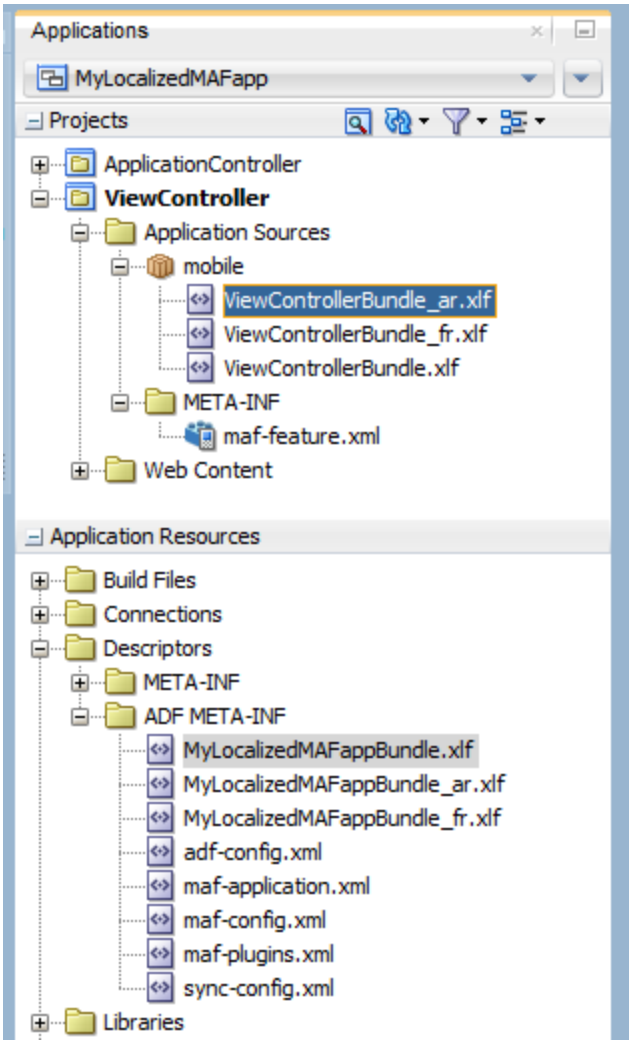
```
<BASE_RESOURCE_BUNDLE_NAME>_<LANGUAGE_TOKEN>.xlf
```

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 MAF Support of Languages.

Figure 8-5 Base Resource Bundle and Locale-Specific Resource Bundles



8.4.1 How to Create a Locale-Specific Resource Bundle

JDeveloper facilitates the creation of project-level resource bundles by providing an option in the **XML** category under **General** of its **New Gallery** menu to create a new XML Localization File (XLIFF).

Alternatively, open the base bundle for which you want to create a locale-specific resource bundle. Save a copy of the file with the right language code in the file name. The second option has the following benefits:

- You create the locale-specific resource bundle in the same directory as the base resource bundle. This 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: double-click the `.xlf` file in the **ADF META-INF** node under **Descriptors** of the Application Resources panel.
- Project-level base resource bundle: double-click the `.xlf` file in the **Application Sources** node under **ViewController** of the Projects panel.

2. In JDeveloper, click **File**, and then **Save As**. 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 to include the appropriate language codes.

   The following example demonstrates the edits that 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 Editing Resources in Resource Bundles.

8.5 Editing Resources in Resource Bundles

JDeveloper provides an **Edit Resources Bundle** dialog where you can add, delete, or edit the resources in the resource bundles that a MAF application contains.

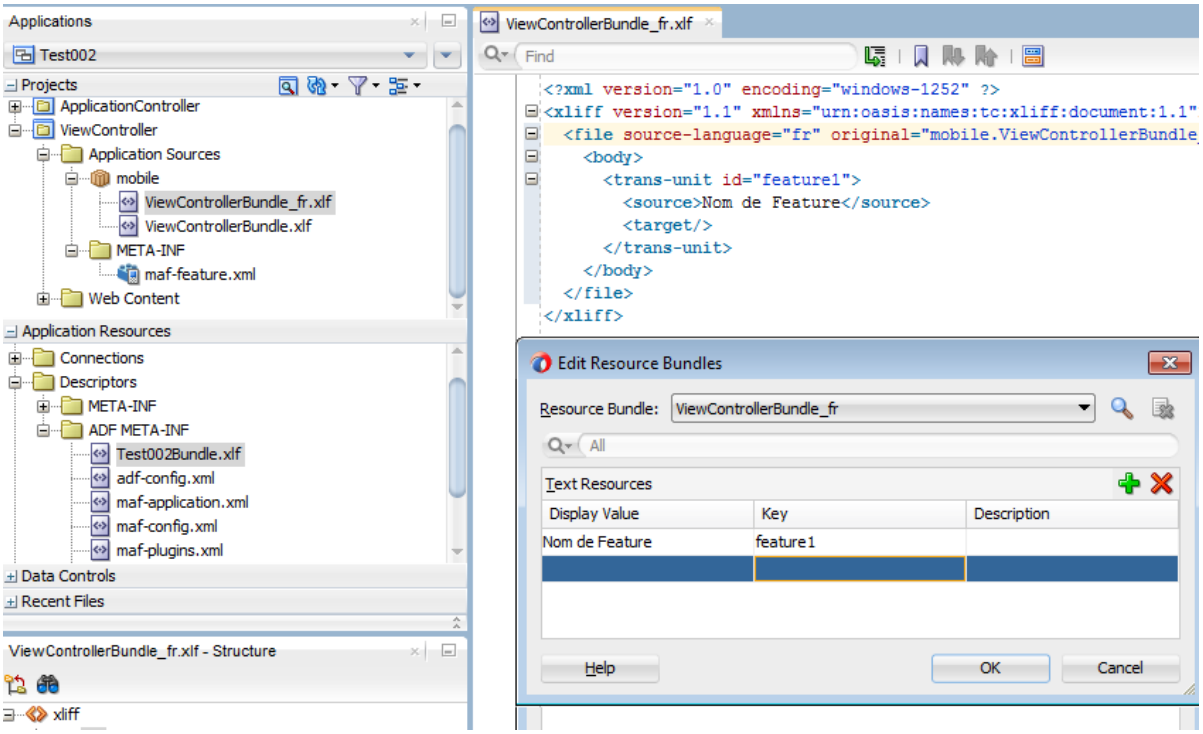
To edit text resources in a resource bundle:

1. In JDeveloper, double-click the resource bundle where you want to edit text resources.

2. In JDeveloper's main menu, click **Application**, and then **Edit Resource Bundles**.

   The **Edit Resources Bundle** dialog opens for the resource bundle that you selected in Step 1. Figure 8-6 shows the dialog that opens for a French locale-specific resource bundle. Double-click in the **Display Value** and **Description** fields to edit existing text resources. Use the controls (**Add**, **Delete**, and **Search**) to add and remove text resources, or to search for and open other resource bundles in the MAF application.

Figure 8-6 Editing Text Resources in Resource Bundles



3. Click **OK** to close the dialog and save the changes.

8.6 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 8-7.

Figure 8-7 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 8-7 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>
```

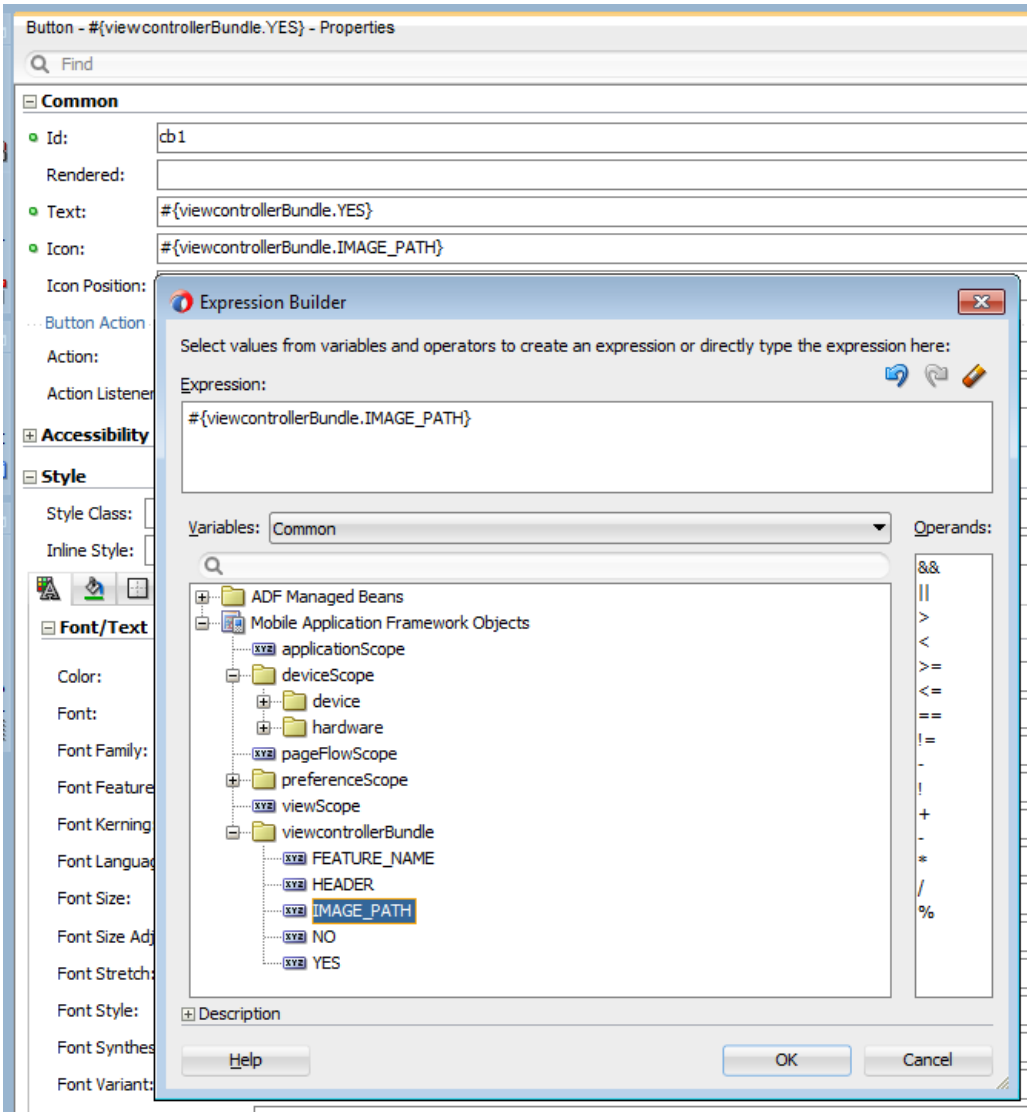
The French resource bundle (`ViewControllerBundle_fr.xlf`) specifies a different image to render when the MAF application runs in a French locale, as shown in the following example.

```
<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, or use the Edit Resource Bundle dialog, as described in Editing Resources in Resource Bundles.

Once you have defined the path to the image in the source bundle, you can use the Expression Builder, as shown in Figure 8-8, to write an EL expression that references the entry in the resource bundle.

Figure 8-8 EL Expression Referencing an Image in a Resource Bundle



8.7 MAF Support of Languages

Lists the languages supported by MAF and the resource bundle names that can be used to provide additional UI strings.

Table 8-1 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 8-1. 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 Creating Locale-Specific Resource Bundles.

During deployment, JDeveloper 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 8-1 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

Table 8-1 (Cont.) Resource Bundle Names for Languages Supported by MAF

Language	Resource bundle name to provide additional UI strings
Italian	BASE_RESOURCE_BUNDLE_NAME_it.xlf
Japanese	BASE_RESOURCE_BUNDLE_NAME_ja.xlf
Korean	BASE_RESOURCE_BUNDLE_NAME_ko.xlf
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

8.8 Localizable MAF Properties

The `maf-application.xml` and `maf-feature.xml` files both expose properties that can reference text resources in resource bundles.

Table 8-2 and Table 8-3 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 8-4 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 8-2.

Table 8-2 Localizable MAF Application Attributes

Element	Attribute
`<adfmf:Application>`	`name`

Table 8-2 (Cont.) Localizable MAF Application Attributes

Element	Attribute
<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 localize application feature-related attributes listed in Table 8-3.

Table 8-3 Localizable Application Feature Attributes

Element	Attribute
<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>`). Table 8-4 lists these MAF AMX UI components.

Table 8-4 Localizable Attributes of MAF AMX UI Components

Component	Attribute
<amx:inputDate>	label
<amx:inputNumberSlider>	label
<amx:panelLabelAndMessage>	label

Table 8-4 (Cont.) Localizable Attributes of MAF AMX UI Components

Component	Attribute
<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

9

Skinning MAF Applications

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

This chapter includes the following sections:

- Introduction to MAF Application Skins
- Adding a Custom Skin to an Application
- Specifying a Skin for an Application to Use
- Registering a Custom Skin
- Versioning MAF Skins
- What Happens When You Version Skins
- Overriding the Default Skin Styles
- What You May Need to Know About Skinning
- Adding a New Style Sheet to a Skin
- Enabling End Users Change an Application's Skin at Runtime
- What Happens at Runtime: How End Users Change an Application's Skin

9.1 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 How to Debug UI Code on the Android Platform and How to Debug UI Code on the iOS Platform. Use Visual Studio to debug user interface code in MAF applications that you deploy to the Universal Windows Platform. For more information, see How to Debug UI Code on the Universal Windows Platform.

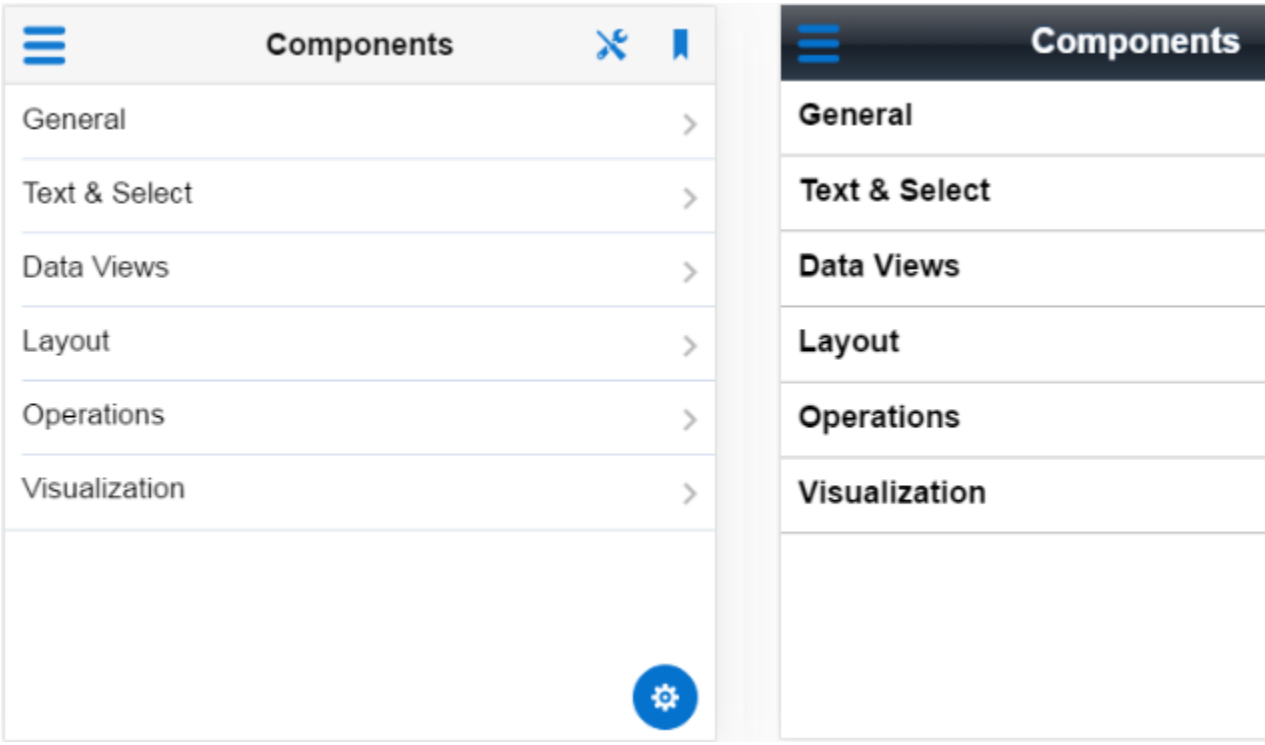
The following are the supported skin families and versions that MAF uses to define the selectors that determine the appearance of MAF AMX pages:

```
amx
  mobileAlta-1.0
    mobileAlta-1.1
      mobileAlta-1.2
         mobileAlta-1.3
            mobileAlta-1.4
               mobileAlta-1.5
                 mobileAlta-1.6
```

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 Specifying a Skin for an Application to Use.

Figure 9-1 demonstrates the difference in look and feel between the `mobileAlta` skin family and another skin family by showing the same application screen rendering using the different skins. The `mobileAlta` skin family renders on the left.

Figure 9-1 Comparison of Look and Feel Provided by mobileAlta and Another Skin



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 JDeveloper's **Build > 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. Furthermore, some of the selectors defined in these files may not be supported.

You use the `maf-config.xml` file, described in About the maf-config.xml File, and the `maf-skins.xml` file, described in 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.

9.1.1 About the maf-config.xml File

After you create a MAF application, JDeveloper 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 in the following example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-family>mobileAlta</skin-family>
  <skin-version>v1.5</skin-version>
  ....
</adfmf-config>
```

Note:

You can determine the skin value at runtime using EL expressions. For more information, see 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:

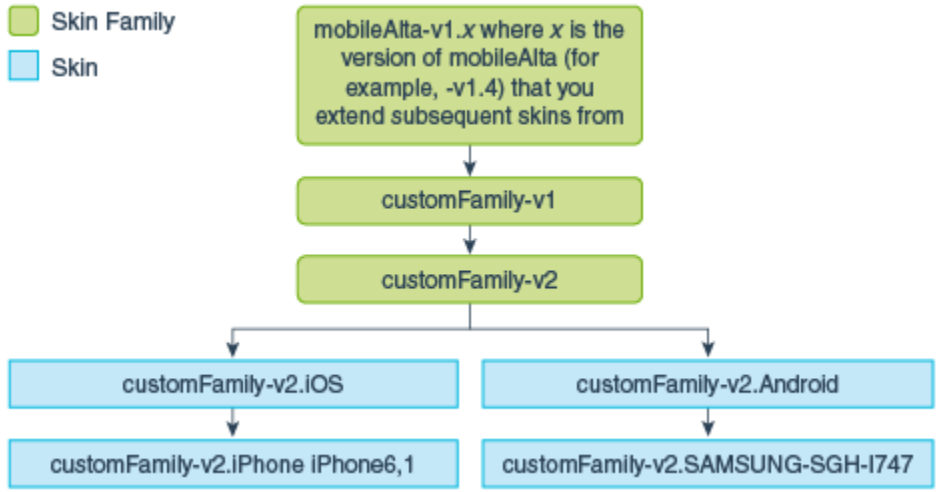
1. `mobileAlta.<DeviceModel>` (for example, `mobileAlta.iPhone5,3`)
2. `mobileAlta.iOS` or `mobileAlta.Android`
3. `mobileAlta`

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 MAF Sample Applications.

Figure 9-2 provides a visual illustration of how MAF applies this hierarchy of skins at runtime. Note also that the `SkinningDemo` sample application, described in 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.iPhone5,3` definition for the same selector. The `<extends>` element, described in 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 What You May Need to Know About Skinning.

Figure 9-2 MAF Skin Hierarchy Application at Runtime



9.1.2 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 9-1 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 9-1 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 9-1 (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-v1.6</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 Versioning MAF Skins.

Table 9-2 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 9-2 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-v1.6.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 following example 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-v1.6.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 9-3 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 MAF Sample Applications, to retrieve information about the device model.

Table 9-3 Platform- and Device-Specific Styling

Device	Example

iPhone	`<skin-addition>`
	`  <skin-id>mobileAlta-v1.6.iPhone5,1</skin-id>`
	`  <style-sheet-name>iPhoneStylesheet.css</style-sheet-name>`
	`</skin-addition>`

iPad	`<skin-addition>`
	`  <skin-id>mobileAlta-v1.6.iPad4,2</skin-id>`
	`  <style-sheet-name>iPadStylesheet.css</style-sheet-name>`
	`</skin-addition>`

iPhone Simulator	`<skin-addition>`
	`  <skin-id>mobileAlta-v1.6.iPhone Simulator x86_64</skin-id>`
	`  <style-sheet-name>iPhoneSimStylesheet.css</style-sheet-name>`
	`</skin-addition>`

All iOS Devices	`<skin-addition>`
	`  <skin-id>mobileAlta-v1.6.iOS</skin-id>`
	`  <style-sheet-name>iOSSimStylesheet.css</style-sheet-name>`
	`</skin-addition>`

9.2 Adding a Custom Skin to an Application

To add a custom skin to your application, create a CSS file within JDeveloper, 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, right-click the **ApplicationController** project and choose **New > CSS File**.
2. In the **Create Cascading Style Sheet** dialog, specify a name and directory for the CSS file.
3. Click **OK**.

You can now open the CSS in the CSS editor and define styles for your application.

9.3 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 **Applications** window, double-click the `maf-config.xml` file. By default, this is in the Application Resources pane under the **Descriptors** and **ADF META-INF** node.
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.

   Example 9-1 shows the configuration required to make a MAF application use the `mobileAlta-v1.6` skin.

Example 9-1 Configuration to Specify a Skin for an Application

```
<adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-family>mobileAlta</skin-family>
  <skin-version>v1.6</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 Enabling End Users Change an Application's Skin at Runtime.

9.4 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 **Applications** window, expand **ApplicationController** > **Application Sources** > **META-INF** and double-click **maf-skins.xml**.
2. In the **Structure** window, right-click the **adfmf-skins** node and choose **Insert Inside adfmf-skins** > **skin**.
3. In the **Insert skin** dialog, complete the fields 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 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-v1.6` skin, enter `mobileAlta-v1.6`.

- **style-sheet-name**—Enter or select the name of the style sheet.

4. Click **OK**.

9.5 Versioning MAF Skins

You can version your MAF skins if you want to distinguish between 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 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 **Applications** window, double-click the `maf-skins.xml` file. By default, this is in the **META-INF** node of the application controller project.
2. In the **Structure** window, right-click the **skin** node for the skin that you want to version and choose **Insert inside skin** > **version**.
3. In the **Insert version** dialog, select **true** from the default list 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 Specifying a Skin for an Application to Use.
4. Enter a value in the name field. For example, enter `v1` if this is the first version of the skin.
5. Click **OK**.

9.6 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.

Example 9-2 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. You can override this behavior by specifying a value for the <skin-version> element in the maf-config.xml file, as described in 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.iPhone5,3 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 Example 9-2, the last skin to be defined is customFamily-v3.iPhone5,3.

Example 9-2 maf-skins.xml File with Versioned Skin Files

```
<?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.iPhone5,3</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.iPhone5,3</id>
    <extends>customFamily-v1.iOS</extends>
    <style-sheet-name>iphone-v3.css</style-sheet-name>
    <version>
      <name>v3</name>
    </version>
  </skin>
</adfmf-skins>
```

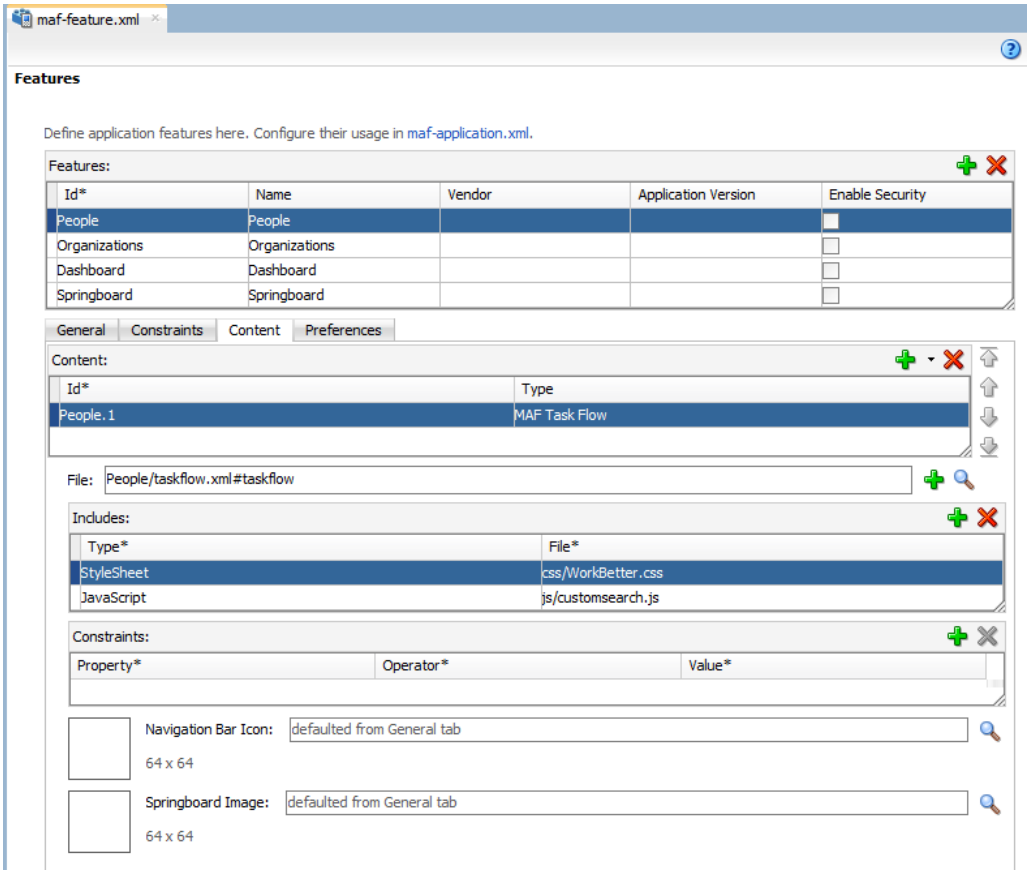
9.7 Overriding the Default Skin Styles

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 9-3 The Includes Table



Before you begin:

Create a MAF task flow as described in Creating Task Flows. Create or add a CSS file for the skin. You can create the CSS file by selecting the view controller project and then choosing **New > CSS File**. Alternatively, you can package the CSS file in a JAR file as follows:

1. From the main menu, choose **Application > Project Properties**.
2. In the **Project Properties** dialog, select the **Libraries and Classpath** page and click **Add JAR/Directory**.
3. In the **Add Archive or Directory** dialog, navigate to the JAR file that contains the skin you want to import and click **Select**.

   The JAR file appears in the **Classpath Entries** list.
4. Click **OK**.

How to add a style sheet to an application feature:

1. Click **Add** to create a new row in the Includes table.
2. In the **Insert Include** dialog, complete the following fields:

- **File**: Browse to select the CSS style sheet to add.
- **Type**: Select **StyleSheet** from the drop-down list.

3. Click **OK**.

Note:

The `.CSS` file for the style sheet that you select must reside within the view controller project.

9.8 What You May Need to Know About Skinning

You can customize styles that can be applied when the application is deployed to different devices.

The CSS files defined in the `maf-skins.xml` file, illustrated in Example 9-3, 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. This sample application is in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.maf/Samples`

For example, at the iOS level, the stylesheet (`mobileAlta` in Example 9-3) 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.

Example 9-3 Skinning Levels Defined in the maf-skins.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/skins">
    <skin>
        <id>mobileAlta-v1.6.iPhone5,3</id>
        <family>mobileAlta</family>
        <extends>mobileAlta-v1.6.iOS</extends>
        <style-sheet-name>skins/mobileAlta-v1.6.iphone.css</style-sheet-name>
    </skin>
    <skin>
        <id>mobileAlta-v1.6.iPad iPad4,1</id>
        <family>mobileAlta</family>
        <extends>mobileAlta-v1.6.iOS</extends>
        <style-sheet-name>skins/mobileAlta-v1.6.ipad.css</style-sheet-name>
    </skin>
    <!--  Skin Additions -->
    <skin-addition>
        <skin-id>mobileAlta-v1.6.iPhone5,3</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition1.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileAlta-v1.6.iPhone5,3</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition2.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileAlta-v1.6.iOS</skin-id>
        <style-sheet-name>skins/mystyles.ios.addition2.css</style-sheet-name>
```

```
    </skin-addition>
</adfmf-skins>
```

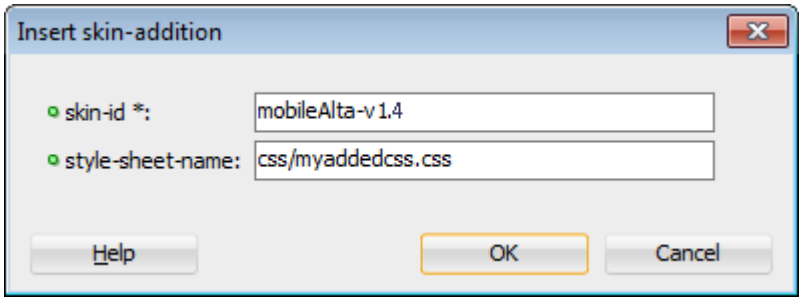
9.9 Adding a New Style Sheet to a Skin

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. Drag and drop a `<skin-addition>` element from the **Components** window to the **Structure** window.
2. Populate the `<skin-addition>` element with the elements described in Table 9-2 by completing the **Insert skin-addition** dialog, shown in Figure 9-4.
   - Enter the identifier of the skin to which you want to add a new style.
   - Retrieve the location of the CSS file.

Figure 9-4 The Insert skin-addition Dialog



3. Click **OK**.

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.

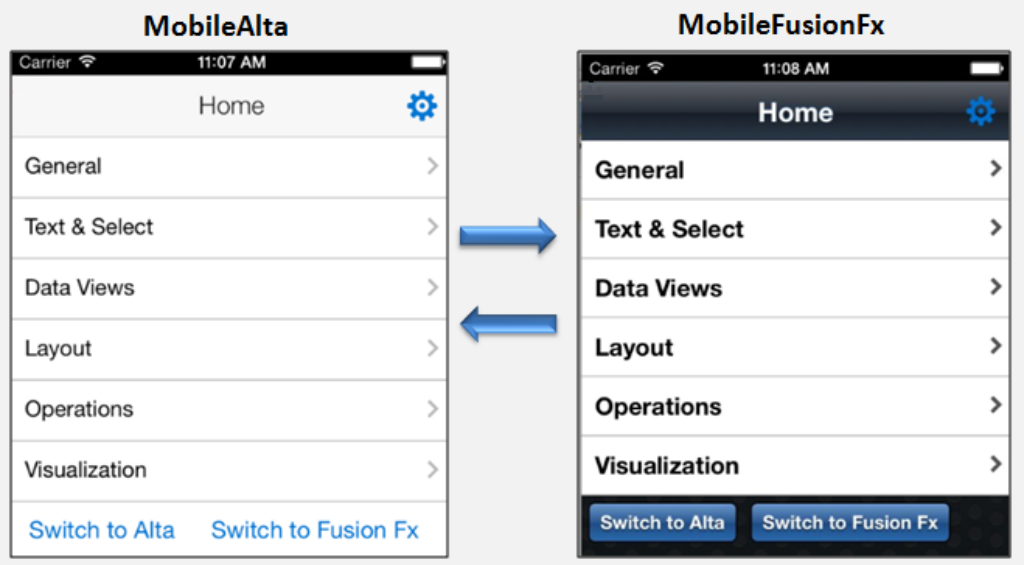
9.10 Enabling End Users Change an Application's Skin at Runtime

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 9-5 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 `maf-config.xml` file.

Figure 9-5 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 `maf-config.xml` file.

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 `maf-config.xml` file.

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 9-5.

   The following example shows how you configure `amx:commandButton` components that allow end users to choose available skins at runtime, as shown in Figure 9-5. 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 9-4 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 9-4 also shows a method that resets all features in the application to use the new skin. Example 9-4 also makes use of the `PropertyChangeSupport` and `PropertyChangeListener` objects described in Working with Data Change Events.

4. In the **Applications** window, 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 9-4 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());
        }
    }
}
```

9.11 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 9-4 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.

10

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:

- Introduction to Using Plugins in MAF Applications
- Enabling a Core Plugin in Your MAF Application
- Registering Additional Plugins in Your MAF Application
- Deploying Plugins with Your MAF Application
- Importing Plugins from a Feature Archive File
- Using a Plugin in a MAF Application
- Providing Usage Descriptions for Plugins that Access Device Capabilities on iOS

10.1 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 overview editor of `maf-application.xml`. 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 in the overview editor of the `maf-application.xml` file.

Select a plugin in the Core Plugins list, as shown in Figure 10-1 to view a description of the plugin. By default, a newly-created MAF application enables only one core plugin, the Network Information plugin. Enable or disable these core plugins, as described in 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. See Introduction to custom Cordova plugin development at http://blogs.oracle.com/mobile/entry/introduction_to_custom_cordova_plugin and 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. See Using a Plugin in a MAF Application.

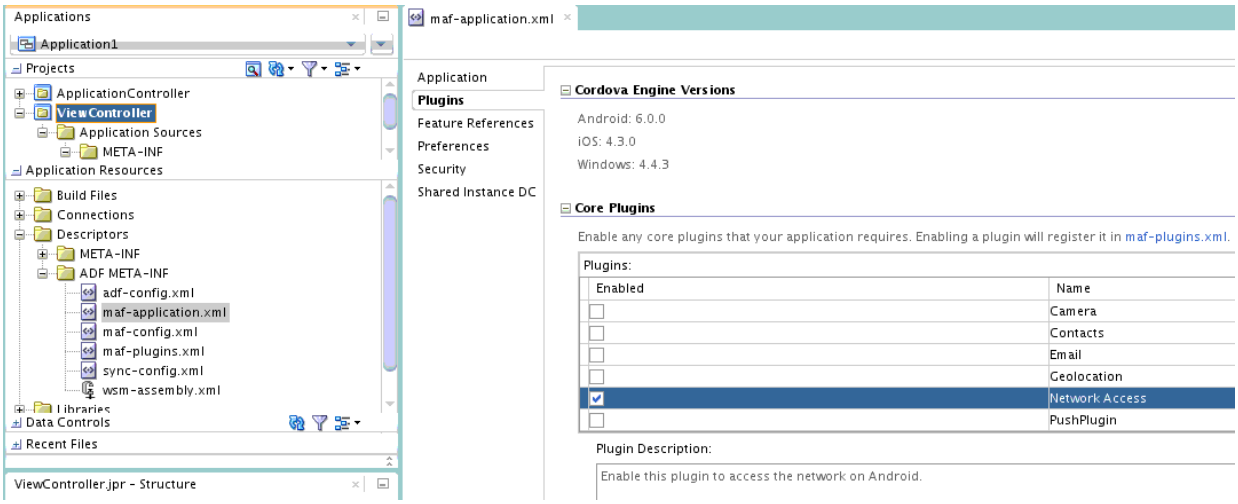
If your MAF application fails to deploy after you register additional plugins, it may be due to filename conflicts between plugins that your MAF application uses. Alternatively, it may be due to the absence of dependent plugin that additional plugins you registered require to function correctly. For more information, see Deploying Plugins with Your MAF Application. MAF applications may fail to deploy to the iOS platform if you do not provide usage descriptions when your MAF application uses plugins that access private data (contacts, photos, and so on) on the iOS device. For more information about these usage descriptions, see Providing Usage Descriptions for Plugins that Access Device Capabilities on iOS.

To migrate a MAF application created with an earlier release of MAF, see Migrating Cordova Plugins from Earlier Releases to MAF 2.3.0 in *Installing Oracle Mobile Application Framework*.

Note:

Editing the `maf-application.xml` file to manage plugins in an application results in revisions to the `maf-plugins.xml` file. The ADF-META-INF node of the Application Resources pane from which both files are accessed, is shown in Figure 10-1.

Figure 10-1 Plugins in the Overview Editor of maf-application.xml



10.2 Enabling a Core Plugin in Your MAF Application

A new application enables only the core Network Information plugin. Additional plugins must be registered before they can be used.

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.

10.2.1 How to Enable a Core Plugin in Your MAF Application

Use the procedure to enable a core plugin using the overview editor of the `mafapplication.xml` file in a MAF application.

You enable a core plugin using the overview editor of the `maf-application.xml` file of the MAF application.

To enable a core plugin in your MAF application:

1. In the Applications window, expand the Application Resources panel.
2. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the **maf-application.xml** file and in the overview editor that appears, click the **Plugins** navigation tab
4. Expand the Core Plugins section and select the plugin that allows your application access features.

   For example, if you want your MAF application to be able to send an SMS message, select the checkbox for the SMS plugin.

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

JDeveloper edits the `maf-plugins.xml` file of the application with entries that identify the enabled plugins in the application.

Once you enable a plugin in the overview editor, JDeveloper edits the application's `maf-plugins.xml` file with entries that identify the enabled plugins in your MAF application. Example 10-1 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 10-1 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>
```

10.3 Registering Additional Plugins in Your MAF Application

You can use additional plugins in a MAF application after registering the plugins.

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

10.3.1 How to Register an Additional Plugin

Use the procedure to register plugins using the overview editor of the `maf-application.xml` file of a MAF application. Registration is necessary if you want to use additional plugins.

To register an additional plugin for a MAF application, use the overview editor of the `maf-application.xml` file of the application .

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 JDeveloper, using a relative path, successfully registers the plugin with your application.

To register an additional plugin for a MAF application:

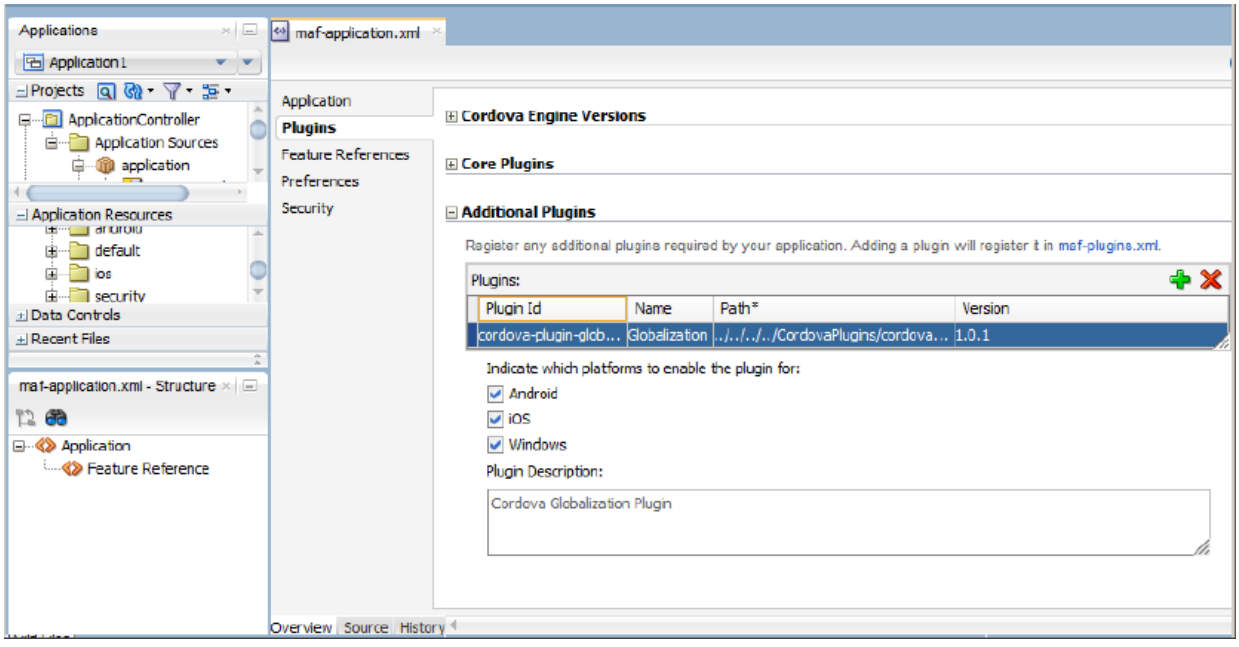
1. In the Applications window, expand the Application Resources panel.
2. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the **maf-application.xml** file and in the overview editor that appears, click the **Plugins** navigation tab.
4. Expand the Additional Plugins section, and click the **Add** icon to display the dialog.
5. Browse to and select the directory that stores the plugin to be registered with the application.

10.3.2 What Happens When You Register an Additional Plugin for Your MAF Application

When an additional plugin is registered, JDeveloper edits the `maf-plugins.xml` file of the application with entries that identify the enabled plugin in the MAF application.

Once you select the source files for the plugin you want your MAF application to use, JDeveloper edits the application's `maf-plugins.xml` file with entries that identify the enabled plugins in your MAF application. Example 10-2 shows the entries in a `maf-plugins.xml` file where the Globalization plugin shown in Figure 10-2 has been registered with the MAF application.

Figure 10-2 Additional Plugins in the Overview Editor of maf-application.xml



Example 10-2 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>
```

10.4 Deploying Plugins with Your MAF Application

A plugin may be deployed to a FAR, or a Mobile Application Archive file. A plugin may be deployed using an Android, iOS, or Windows deployment profile.

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 Profile

During deployments using an Android, iOS or Windows deployment profile, JDeveloper 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"/>
```

Usage Descriptions for MAF Applications Deployed to iOS

Deployment to the iOS platform fails if you enable a plugin that requires your MAF application to access device features (contacts and photos, for example). Provide a

usage description as described in Providing Usage Descriptions for Plugins that Access Device Capabilities on iOS.

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, JDeveloper 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. See 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 10-3.

Example 10-3 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>
```

10.5 Importing Plugins from a Feature Archive File

Plugins can be imported into an application 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. JDeveloper 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, JDeveloper 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, JDeveloper 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 Registering Additional Plugins in Your MAF Application.

10.6 Using a Plugin in a MAF Application

A plugin that is enabled in a MAF application can be invoked to interact with device features, such as the camera, or use other applications on the device, such as email or barcode scanner applications.

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 10-3 Platform-Specific Content and Constraint To Access a Plugin

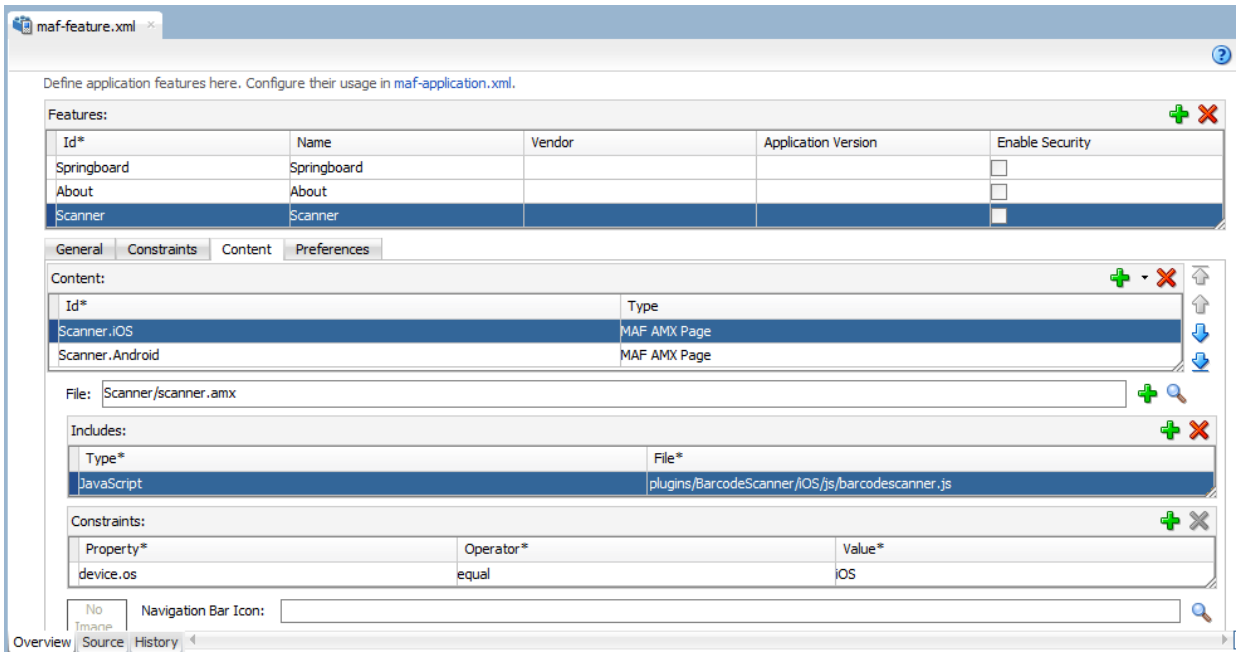
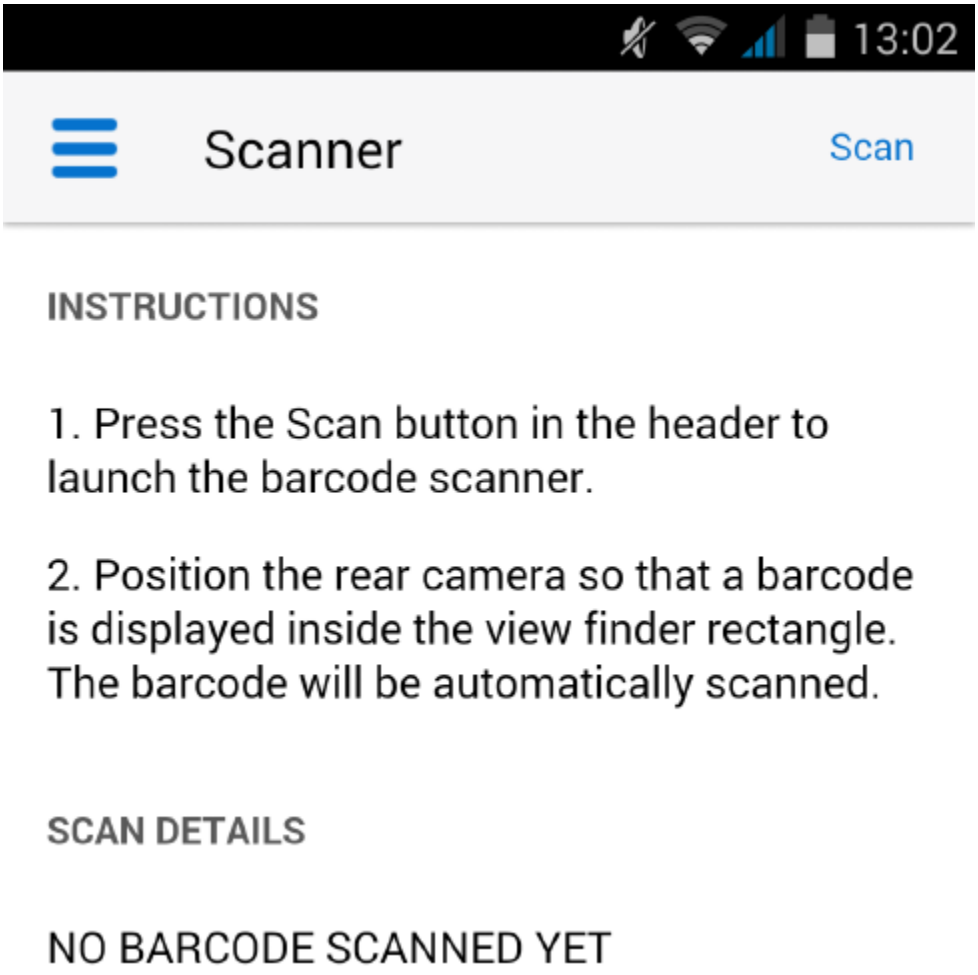


Figure 10-4 shows a button (**Scan**) in the MAF AMX page that the `BarcodeDemo` sample application renders on the user's device at runtime. This button invokes a managed bean method and the managed bean method invokes a JavaScript function that calls the BarcodeScanner plugin.

Figure 10-4 Command Button Invoking Managed Bean Method to Access Plugin



Example 10-4 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`, `FakeBeacon`, and `DatePicker`. See MAF Sample Applications.

Example 10-4 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.getFeatureId(),
                                                              "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.-->
```

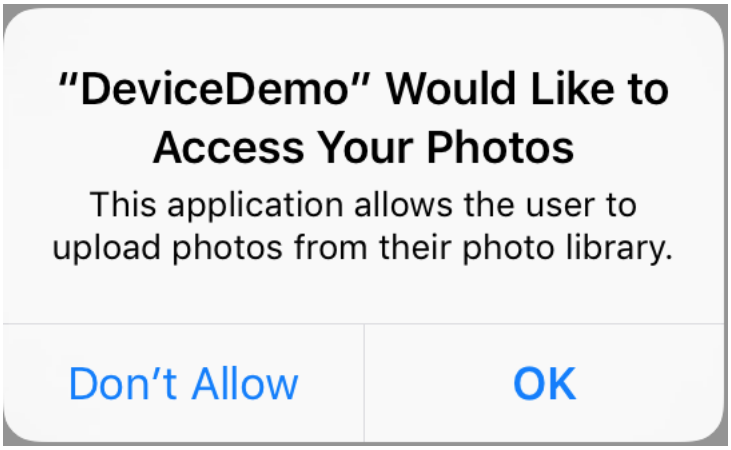
```
function scanBarcodeFromJavaBean(options)
        {
            cordova.plugins.barcodeScanner.scan(
              function(result)
               function onSuccess(result) {
        adf.mf.api.setValue( { "name": "#{viewScope.BarcodeBean.barcodeError}",
                               "value": ""},
                               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);
    }
```

10.7 Providing Usage Descriptions for Plugins that Access Device Capabilities on iOS

MAF applications that you deploy to iOS require usage descriptions if the MAF application uses hardware capabilities, such as the device camera, that allow it to access user data.

The iOS platform requires these descriptions to display in the system dialog that it uses to prompts an end user to allow an application access to the functionality and potentially sensitive user data. Figure 10-5 shows an example where the text "`This application allows the user to upload photos from their photo library`" is the usage description from the MAF application.

Figure 10-5 Usage Description in the UI of a MAF Application Deployed to iOS



Basic usage of MAF does not enable functionality that requires you to provide usage descriptions. However, if you enable a core plugin that requires a usage description or use other plugins that require usage description, you need to provide a usage description to successfully deploy the application. Your application may crash or the Apple App Store may reject it if you do not provide a usage description.

The Camera, Contacts, and Geolocation core plugins require usage descriptions. MAF provides the following generic usage descriptions for these core plugins:

- `Camera`: The camera plugin enables access to the device camera and photo libraries.

  This plugin requires usage descriptions for the following Cocoa keys: `NSCameraUsageDescription` and `NSPhotoLibraryUsageDescription`.

- `Contacts`: The contacts plugin enables access to the address book on the device.

  This plugin requires a usage description for the following Cocoa key: `NSContactsUsageDescription`.

- `Geolocation`: The geolocation plugin uses device location services on the device.

  This plugin requires a usage description for the following Cocoa key: `NSLocationWhenInUseUsageDescription`.

These generic usage descriptions meet the technical requirements of the iOS platform to successfully deploy the MAF application. We recommend that you provide a specific usage description that explains to your end users why your application needs to access the functionality and data requested by the plugin. For additional plugins that you register with your MAF application that require usage descriptions, you must provide the usage description.

You provide a usage description in your application's resource bundle `.XLF` file. The usage descriptions uses an iOS platform Cocoa key(s) as the value for the `trans-unit` element's `id` attribute. The following example shows a usage description that appears when a MAF application prompts an end user to grant access to the user's photo library.

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

```
      <trans-unit id="NSPhotoLibraryUsageDescription">
        <source>This application allows the user to upload photos from their photo
library.</source>
        ...
```

Also provide usage descriptions in locale-specific resource bundles if your MAF application supports more than one locale. The following example shows the corresponding usage description that renders when the locale is French.

```
...
  <file source-language="fr" original="mobile.ViewControllerBundle_fr" datatype="x-
oracle-adf">
    <body>
      <trans-unit id="NSPhotoLibraryUsageDescription">
        <source>Cette application permet à l'utilisateur de télécharger des photos à
                partir de leur bibliothèque de photos.</source>
        ...
```

At deployment time, MAF populates the usage descriptions that you define in the resource bundle into the `info.plist` file of the application that is deployed to the iOS device.

For more information about creating resource bundles in a MAF application, including how to create locale-specific resource bundles, see Localizing MAF Applications .

For a list of the Cocoa keys that identify usage descriptions, see the keys that append `UsageDescription` to their key name in Apple's Cocoa Keys documentation at https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html.

11

Customizing MAF Application Artifacts with MDS

This chapter describes how to use Oracle Metadata Services (MDS) to customize MAF application-level artifacts.

This chapter includes the following sections:

- Introduction to Applying MDS Customizations to MAF Files
- Configuring Customization Layers
- Creating Customization Classes
- Consuming Customization Classes
- Understanding a Customization Developer Role
- Enabling Customizations in Resource Bundles
- Upgrading a MAF Application with Customizations

11.1 Introduction to Applying MDS Customizations to MAF Files

MDS can be used to adapt an application to different industries, locations, or user groups by means of rebranding, customization, and personalization.

Use Oracle Metadata Services (MDS) to re-brand, customize, and personalize a MAF application at design time. With MDS, an application adapts to different industries, locations, or user groups. For example, MDS can tailor the application look and feel to a user group or user responsibility.

A customized application contains a base application along with one or more layers of customization. Each layer can have multiple layer values. These layer values can be applied in a specified order, in terms of precedence, on top of the base metadata.

MAF supports the MDS seeded customization pattern. By defining layers of customization that are applied at design time, a general application is adapted to a particular group, such as a specific industry or a site. These seeded customizations exist as part of the deployed application, and persist for the life of a given deployment.

Use MDS to customize the following artifacts of a MAF application:

- The `maf-feature.xml` file
- The `maf-skins.xml` file
- The `maf-application.xml` file
- The `maf-config.xml` file

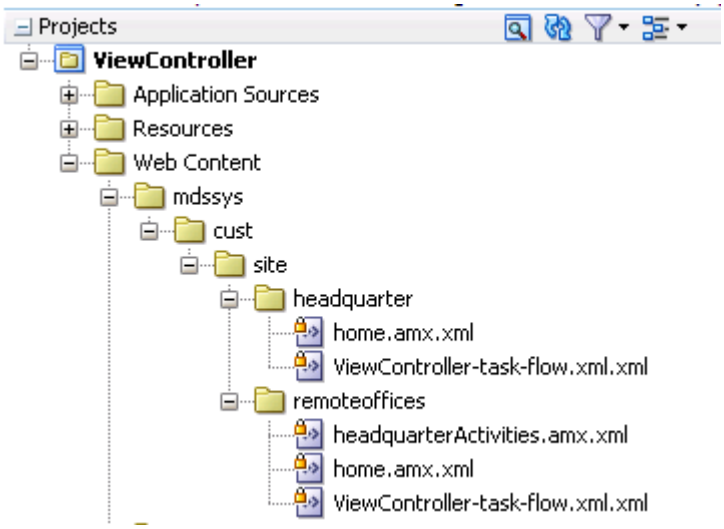
- MAF AMX files and metadata files (see Customizing MAF AMX Application Feature Artifacts).

11.2 Customizing MAF Applications with MDS

Customize a MAF application using MDS by defining global or application-specific customization layers, creating a customization class, and packaging it as a JAR file for JDeveloper to access. Use the procedure to customize a MAF application using MDS.

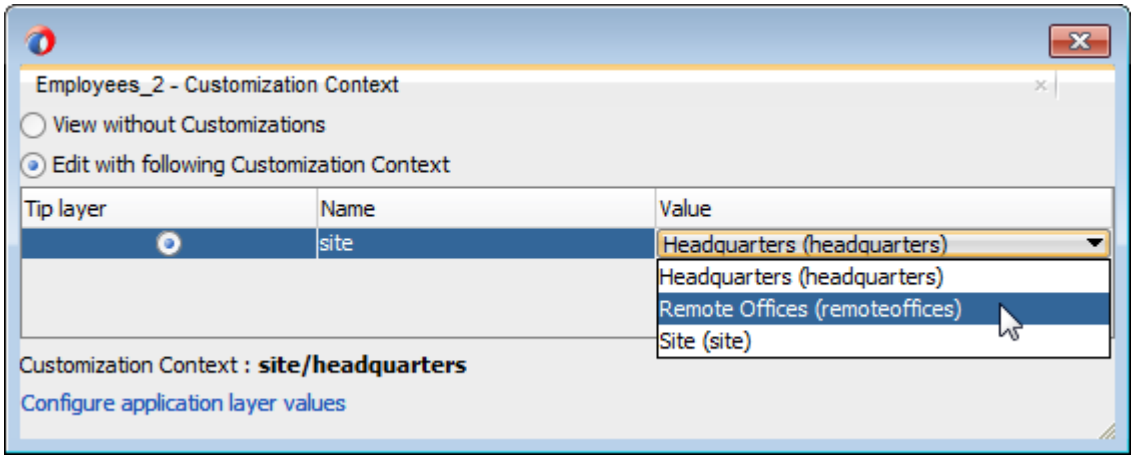
To customize a MAF application using MDS, perform the following:

1. Define one or more global or application-specific customization layers. See Configuring Customization Layers.
2. Create a customization class that MDS uses to determine which customization to apply to the base application. Each customization class defines a base customization layer. See Creating Customization Classes.
3. Enable JDeveloper design time to access the customization by packaging the customization class (a `.java` file) as a JAR file. Then add the JAR file to one of the projects of the MAF application. See Consuming Customization Classes .
4. Add the customization class to the `cust-config` section of the `adf-config.xml` file to register the customization classes in the order of precedence.
5. Launch JDeveloper in the Customization Developer role (or switch to that role). See Understanding a Customization Developer Role.
6. Modify the files as required. The changes are recorded by MDS in the `mdssys` directory of the ViewController project.



7. Select the customization layer from the Customization Context window, as shown in Figure 11-1.

Figure 11-1 Selecting the Customization Layer (Tip Layer)



Note:

When working in the Customization Developer role, the layer and layer value that you select in the Customization Context window is called the tip layer. The changes made while in the Customization Developer role are applied only to this layer.

8. Deploy the application to a device, emulator, or as a platform-specific application package. The Customization Developer role must be used to deploy a customized application, as follows:

   a. Launch the application in the Customization Developer role.

   b. In the Customization Context window, shown in Figure 11-1, select the layer and value for which you want to implement customizations.

   c. Click **Application** , and then click **Deploy** to select the deployment option and the deployment profile. See Deploying MAF Applications .

   d. Deploy each customization context separately.

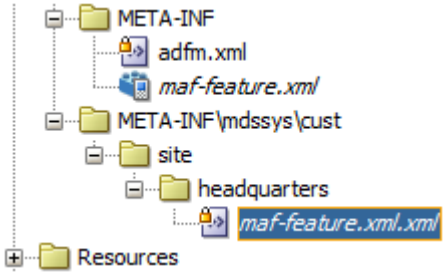
   During deployment, the base file and the delta files are merged to create the customized version of the application at runtime. The deployed application has no MDS dependencies.

   Tip:

   To perform additional customization and upgrades, deploy the customized application as a MAF Application Archive (`.maa`) file, and then import it into an application. The delta files included in the `.maa` file are merged with the base files after deployment. See Upgrading a MAF Application with Customizations.

   When the customization process is complete, JDeveloper creates both a metadata file for the customizations and a subpackage to store them. The metadata file contains the customizations for the customized object, which are applied over the base metadata at runtime. JDeveloper gives the new metadata file the same name as the base file for the object, but includes an additional `.xml` extension, as illustrated by `maf-feature.xml.xml` in Figure 11-2.

Figure 11-2 maf-feature.xml Metadata File



11.3 Configuring Customization Layers

Customize an application by specifying the customization layers and their values in the `CustomizationLayerValues.xml` file so that they are recognized by JDeveloper. JDeveloper reads the `adf-config.xml` file, and the customization layers and their values in the `CustomizationLayerValues.xml` file so that they can be used in applications.

When you open a customizable application in the Customization Developer role, JDeveloper reads the `adf-config.xml` file to determine the customization classes to use and their order of precedence. JDeveloper also reads the `CustomizationLayerValues.xml` file to determine the layer values to make available in the Customization Context window. If there are layer values defined in the `CustomizationLayerValues.xml` file that are not defined in the customization classes listed in the `adf-config.xml` file, they are not displayed in the Customization Context window.

Therefore, you can have a comprehensive list of layer values for all of your customization projects in the `CustomizationLayerValues.xml` file, and only those appropriate for the current application are available in the Customization Context window. Conversely, you could have a comprehensive list of customization classes for a MAF application in the `adf-config.xml` file, and only the subset of layer values on which you would work in your `CustomizationLayerValues.xml` file.

Note:

At design time, JDeveloper retrieves customization layer values from the `CustomizationLayerValues.xml` file. However, at runtime the layer values are retrieved from the customization class.

The names of the layers and layer values that you enter in the `CustomizationLayerValues.xml` file must be consistent with those specified in your customization classes. The following example shows the contents of a sample `CustomizationLayerValues.xml` file.

```
<cust-layers  xmlns="http://xmlns.oracle.com/mds/dt">
   <cust-layer name="industry" id-prefix="i">
      <cust-layer-value value="financial"
                        display-name="Financial"
                        id-prefix="f"/>
      <cust-layer-value value="healthcare"
                        display-name="Healthcare"
                        id-prefix="h"/>
   </cust-layer>
```

```
   <cust-layer name="site" id-prefix="s">
      <cust-layer-value value="headquarters"
                        display-name="HQ"
                        id-prefix="hq"/>
      <cust-layer-value value="remoteoffices"
                        display-name="Remote"
                        id-prefix="rm"/>
   </cust-layer>
</cust-layers>
```

For each layer and layer value, you can add an `id-prefix` token. This helps to ensure the uniqueness of the `id`, so that customizations are applied accurately. When you add a new element during customization, JDeveloper adds the `id-prefix` of the layer and layer value (determined by the selected tip layer) to the auto-generated identifier for the element to create an `id` for the newly added element in the customization metadata file. In the preceding example, the site layer has an `id-prefix` of `s` and the headquarters layer value has an `id-prefix` of `hq` . Therefore, when you select `site/headquarters` as the tip layer and add an element, the `id` of that element will be set to `shqel` in the metadata customization file.

For each layer value, you can also add a `display-name` token to provide a human-readable name for the layer value. When you are working in the Customization Developer role, the value of the `display-name` token is shown in the Customization Context window for that layer value.

For each layer, you can optionally provide a `value-set-size` token that defines the size of the value set for the customization layer. This can be useful, for example, when using a design-time, application-specific `CustomizationLayerValues.xml` file. By setting `value-set-size` to `no_values` you can exclude runtime-only layers at design time.

```
<cust-layer name="runtime_only_layer" value-set-size="no_values"/>
```

You can either define the customization layer values globally for JDeveloper, or in an application-specific file. If you use an application-specific file, it takes precedence over the global file. For more information on configuring layer values globally for JDeveloper, see How to Configure the Layer Values Globally. For more information on configuring application-specific layer values, see Using the Studio Developer Role.

11.3.1 How to Configure the Layer Values Globally

Customization layer values are configured globally for JDeveloper in design time in the `CustomizationLayerValues.xml` file located in the `jdev` subdirectory of the JDeveloper installation directory. Use the procedure to configure design time customization layer values globally for JDeveloper.

Before you begin:

- Create your customization classes, as described in Creating Customization Classes.
- Make your classes available to JDeveloper, as described in Consuming Customization Classes .

To configure design time customization layer values globally for JDeveloper:

1. Open the `CustomizationLayerValues.xml` file located in the `jdev` subdirectory of your JDeveloper installation directory (`jdev_install\jdev\CustomizationLayerValues.xml`).

2. For each layer, enter a `cust-layer` element, as shown in the following example:

```
<cust-layers xmlns="http://xmlns.oracle.com/mds/dt">
   <cust-layer name="company" id-prefix="c">
     <!-- Generated id-prefix would be "c1" and "c2" for values
 "company1" and "company2".-->
     <cust-layer-value value="company1" display-name="Acme Inc"
 id-prefix="1"/>
     <cust-layer-value value="company2" display-name="My Corp" id-prefix="2"/>
     <!-- Generated id-prefix would be "s" for value "site"
    since no prefix was specified on the value -->
   </cust-layer>
 </cust-layers>
```

3. For each layer value, enter a `cust-layer-value` element, as shown in the preceding example.

4. Save and close the `CustomizationLayerValues.xml`file.

5. After you have made changes to the global `CustomizationLayerValues.xml` file, restart JDeveloper.

> **Note:** For procedures to configure the CustomizationLayerValues.xml file for a specific application, see Using the Studio Developer Role and Using the Customization Developer Role.

11.3.2 How to Configure the Application-Level Layer Values

Application-level customization layer values can be configured using either the Studio Developer role or the Customization Developer role.

When configuring layer values for an application, you can use either the Studio Developer role (see Using the Studio Developer Role) or the Customization Developer role (see Using the Customization Developer Role). Note that when you configure an application-specific `CustomizationLayerValues.xml` file, you can create and modify layer values, but you cannot create additional customization layers. It is not necessary to restart JDeveloper to pick up changes made to the application-specific layer values.

When you create an application-specific `CustomizationLayerValues.xml` file, JDeveloper stores it in an application-level directory (for example, `workspace-directory\.mds\dt\customizationLayerValues \CustomizationLayerValues.xml`). You can access this file in the Application Resources window of the Applications window, under the **MDS DT** node.

11.3.2.1 Using the Studio Developer Role

After you have created customization classes that are available to JDeveloper, use the procedure to configure the `CustomizationLayerValues.xml` file for a specific application from the Studio Developer role.

The following procedure describes how to configure the `CustomizationLayerValues.xml` file for a specific application from the Studio Developer role.

Before you begin:

- Create your customization classes, as described in Creating Customization Classes

- Make your classes available to JDeveloper, as described in Consuming Customization Classes

To configure design-time customization layer values at the workspace level from the Studio Developer role:

1. In the Application Resources window, expand the **Descriptors > ADF META-INF** node, and then double-click **adf-config.xml**.
2. In the Overview editor, click the **MDS Configuration** navigation tab.
3. On the MDS Configuration page, below the table of customization classes, click **Configure Design Time Customization Layer Values** to open the workspace-level `CustomizationLayerValues.xml` file in the Source editor.

   Note:

   If the override file does not exist, JDeveloper displays a confirmation dialog. Click **Yes** to create and open a copy of the global file.

4. In the file, specify layer values as necessary, as described in Configuring Customization Layers.
5. Save your changes.

11.3.2.2 Using the Customization Developer Role

After you have created customization classes and made them available to JDeveloper, use the procedure to configure the `CustomizationLayerValues.xml` file for a specific application from the Customization Developer role.

The following procedure describes how to configure the `CustomizationLayerValues.xml` file for a specific application from the Customization Developer role.

Before you begin:

- Create your customization classes, as described in Creating Customization Classes
- Make your classes available to JDeveloper, as described in Consuming Customization Classes

To configure design-time customization layer values at the workspace level from the Customization Developer role:

1. In the Customization Context window, click **Configure application layer values** to open the `CustomizationLayerValues.xml` file in the Source editor.

   Note:

   If the override file does not exist, JDeveloper displays a confirmation dialog. Click **Yes** to create and open a copy of the global file.

2. In the file, specify layer values as necessary, as described in Configuring Customization Layers.
3. Save your changes.

After you make changes to the application-specific `CustomizationLayerValues.xml` file while you are in the Customization Developer role, any tip layer you have selected in the Customization Context window is deselected. You can then select the desired tip layer.

11.4 Creating Customization Classes

Customizations are used to adapt a MAF application for a specific industry domain. Create a Java application for the customization class, complete the Java class, import it into the MAF application, and then use the procedure to create a customization class.

A customization class is a POJO class that extends `oracle.mds.cust.CustomizationClass`. It evaluates the current context and returns a String result. This String result is used to locate the customization layer.

The customization class provides the following information:

- A name that represents the name of the layer.
- An IDPrefix, for objects created in the layer. When new objects are created in a customization layer, they need a unique ID. The IDPrefix is added to the auto-generated identifier for the object to create an ID for the newly added object. Each layer needs a unique IDPrefix so that objects created at different customization layers have unique IDs.
- A cache hint (CacheHint), for the layer defined by the customization class. In MAF, the cache hint defines a static customization layer and the `getCacheHint` method always returns `ALL_USERS` which means the customization is applied globally (unconditionally) for a given deployment.

Note:

Since customization classes are likely to be executed frequently, once for each document being accessed to get the layer name and layer value, you must ensure their efficiency.

Customizations can be used to tailor a MAF application to suit a specific industry domain (verticalization). Every such domain denotes a customization layer and is depicted using a customization class.

Static customizations have only one layer value, in effect, for all executions of the application. A static customization has the same value for all users executing the application.

In the customization class used in a MAF application, the `getCacheHint` method always returns `ALL_USERS` meaning that the customization layer is always static.

All objects could have a static customization layer, depending on how the customization classes are implemented.

Do not create the customization file in the MAF application that you plan to customize. Instead, create a separate Java application for the customization class. After you complete the Java class, import it into the MAF application to be customized.

To create a customization class:

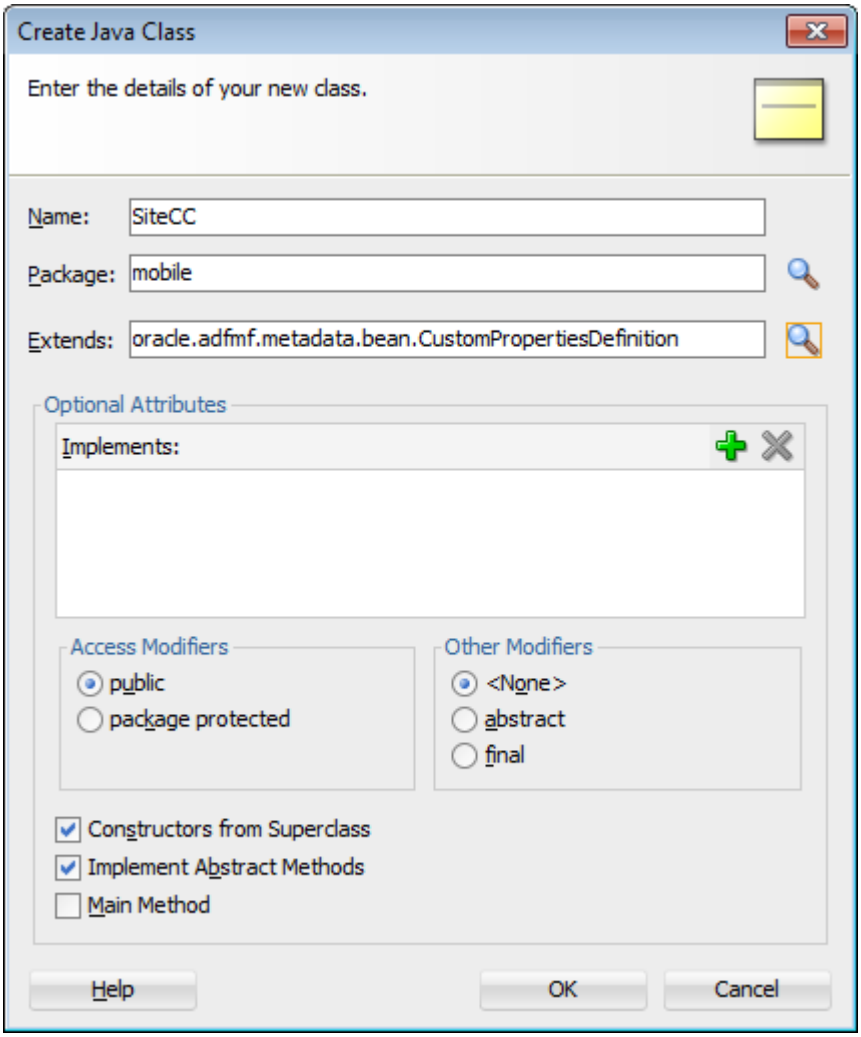
1. Create a Java application.
2. On the **File** menu, click **New**, and then **Project**.

3. In New Gallery, select **Java Application Project**, and then complete the wizard.
4. In the Applications window, right-click the Java application project, and then select **Project Properties**.
5. In the Project Properties dialog, select **Libraries and Classpath**, and then click **Add Library**.
6. In the Add Library dialog, select **MDS Runtime** and then click **OK**. Click **OK** to close the Project Properties dialog.
7. In the Applications window, right-click the Java application project and then select **New** and then **Java Class**.
8. In the Create Java Class dialog, enter the name of the class and package.
9. In the **Extends** field, browse the class hierarchy and retrieve `oracle.mds.cust.CustomizationClass`, as shown in Figure 11-3, and then click **OK**.

Note:

Implement Abstract Methods (the default setting) must be selected in the Create Java Class dialog.

Figure 11-3 Creating the Customization Class



10. Update the stub file. The following example illustrates a customization class.

```
package mobile;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import oracle.mds.core.MetadataObject;
import oracle.mds.core.RestrictedSession;
import oracle.mds.cust.CacheHint;
import oracle.mds.cust.CustomizationClass;

public class SiteCC extends CustomizationClass {

   public SiteCC() {
         super();
     }

     public CacheHint getCacheHint()
        {
            return CacheHint.ALL_USERS;
        }
```

```
        /** {@inheritDoc} */
        public String getName()
        {

            return "company";
        }


        /** {@inheritDoc} */
        public String[] getValue(RestrictedSession rs,
                                 MetadataObject metadataObject)
        {
            String sites[] = {"company1", "company2"};
            return sites;
        }
}
```

11. Rebuild the Java application project.

11.5 Consuming Customization Classes

Customization classes that have been created are available at design time in the Customization Developer role, and at runtime in the application. Use the procedures to add customization classes to a JAR, register the customization class with the MAF application, and to identify customization classes in the `adf-config.xml` file.

Customization classes are reusable components. Create a separate project to store them, and package them in their own JAR file. To make the customization classes available to JDeveloper, import the JAR into the consuming application.

Package the customization class as a JAR file, and then register the class with the MAF application. To package the customization class and any related artifacts into a JAR file, create a deployment profile using the Create Deployment Profile wizard. See About Automatically Generated Deployment Profiles.

To add customization classes to a JAR:

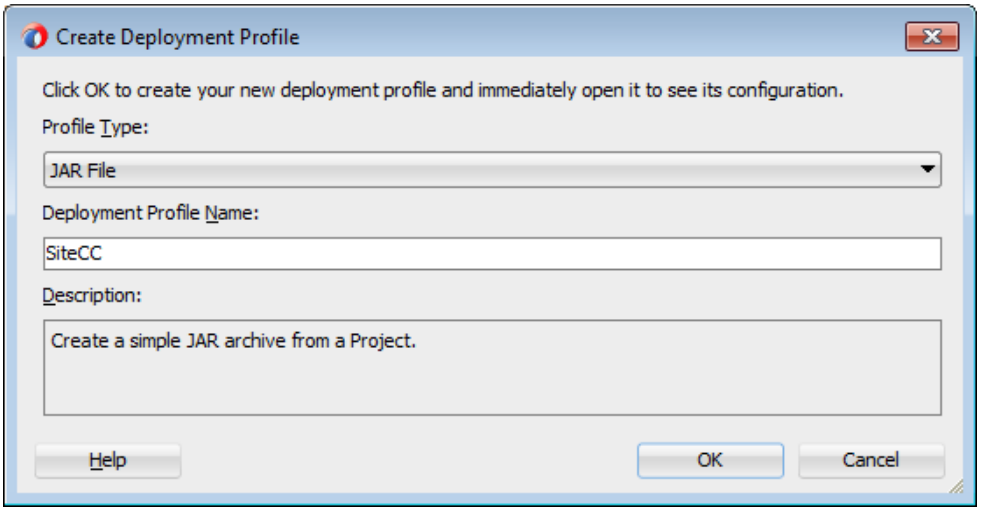
1. In the Applications window, right-click the Java application project and select **New**, and then **From Gallery**.
2. In New Gallery, expand **General**, select **Deployment Profiles,** then select **JAR File**, and click **OK**.

   Tip:

   Click the **All Features** tab if the **Deployment Profiles** node does not appear in the **Categories** tree.

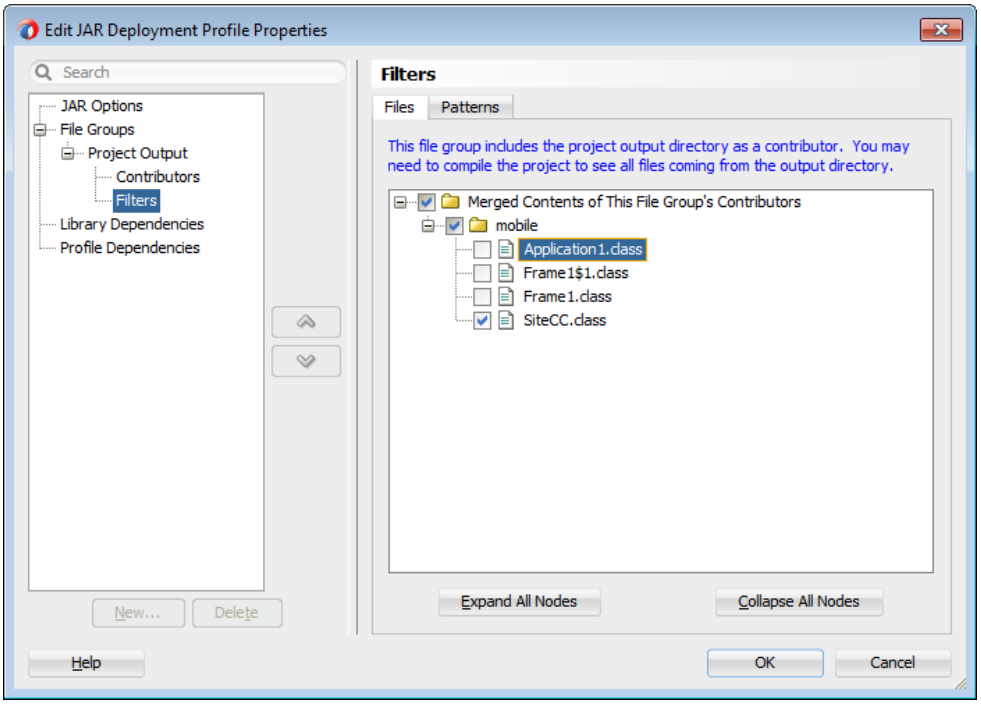
3. In the Create Deployment Profile -- JAR File dialog, enter a name for the project deployment profile (for example, `SiteCC` in Figure 11-4) and then click **OK**.

Figure 11-4 Creating the Deployment Profile for the Customization Class



4. In the Edit JAR Deployment Profile Properties dialog, select **JAR Options**.
5. If needed, enter a location for the JAR file. Otherwise, accept the default location.
6. Expand **Files Groups**, then **Project Output**, and then **Filters** to list the files that can be selected to be included in the JAR.
7. On the Filters page, under the **Files** tab, select the customization classes that you want to add to the JAR file, as illustrated in Figure 11-5.

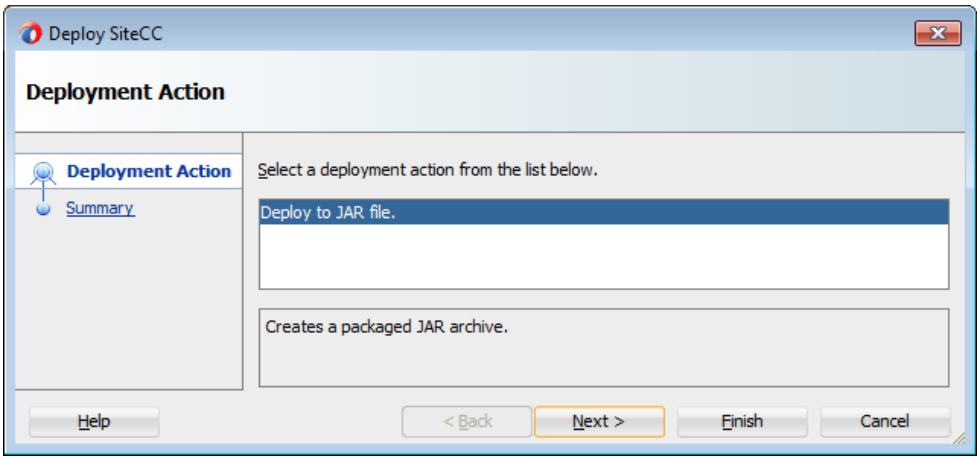
Figure 11-5 Including the Customization Class in the JAR File



8. Click **OK** to exit the Edit JAR Deployment Profile Properties dialog.
9. Click **OK** again to exit the Project Properties dialog.

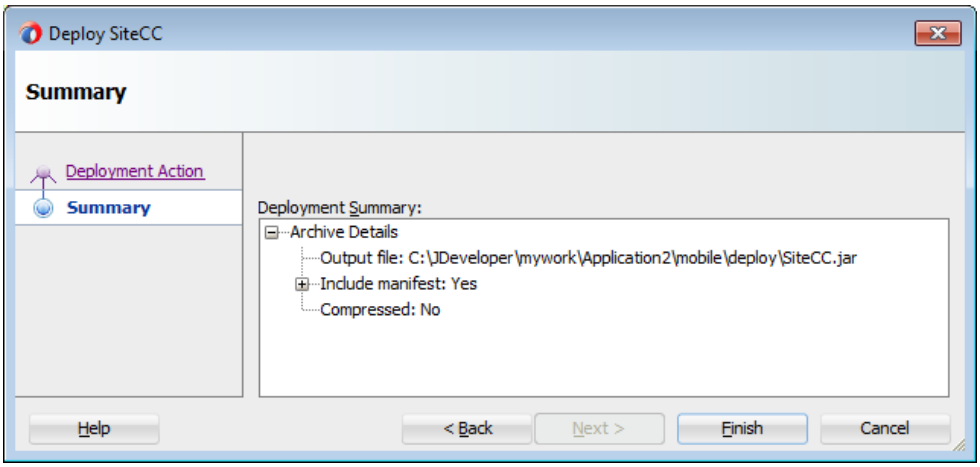
10. In the Applications window, right-click the Java application project and then select the deployment profile. On the Deployment Action page, illustrated in Figure 11-6, **Deploy to JAR** is selected by default. Click **Next**.

Figure 11-6 Deploying the Customization Class to a JAR File



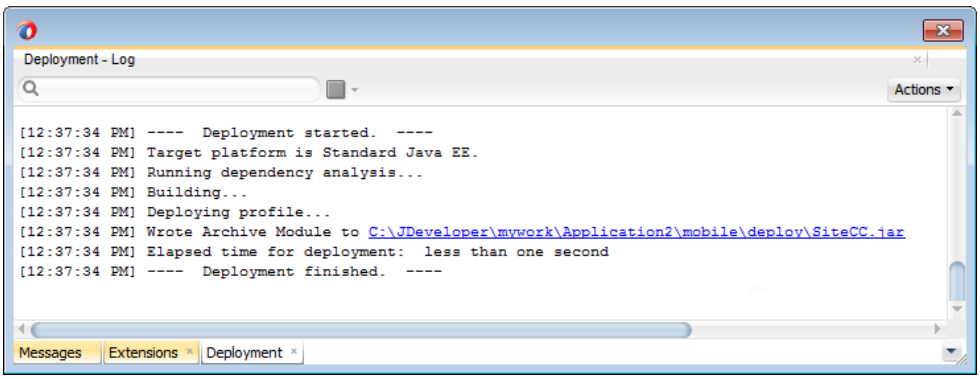
11. Review the confirmation for the output location of the JAR file. Click **OK**.

Figure 11-7 Summary Page (Showing the Output Location for the JAR File)



The log file window, shown in Figure 11-8, displays the status of the deployment.

Figure 11-8 Deployment Log



Use the following procedure to make the customization classes visible to the application, and then add the customization classes to the `cust-config` section of the `adf-config.xml` file.

> **Note:**
>
> The following procedure is not required if you had created your customization classes in the data model project of the consuming application.

Before you begin:

- Create your customization classes in an external project.
- Create a JAR file that includes the customization classes.
- Launch JDeveloper using the Studio Developer role, and open the application that you want to customize.

To register the customization class with the MAF application:

1. In the Applications window, click the Application Menu icon and select **Application Properties**.
2. In the Application Properties dialog, select **Libraries and Classpath**, and click **Add JAR/Directory**.
3. In the Add Archive or Directory dialog, select the JAR file you created that contains the customization classes, and click **Open**.
4. Click **OK**.

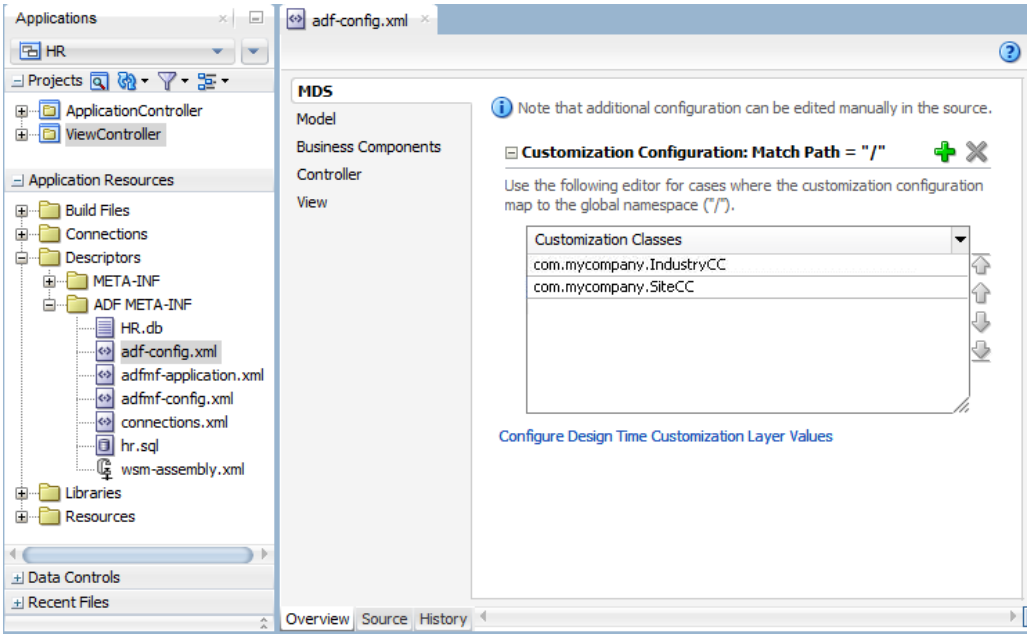
The next step is to add the customization class to the `adf-config.xml` file. The `adf-config.xml` file of the application must have an appropriate `cust-config` element in the `mds-config` section. The `cust-config` element allows clients to define an ordered and named list of customization classes. Use the Overview editor for the `adf-config.xml` file to add customization classes (see Figure 11-9).

To identify customization classes in the `adf-config.xml` file:

1. In the Application Resources window, expand the **Descriptors** and **ADF META-INF** nodes, and then double-click **adf-config.xml**.
2. In the Overview editor, select **MDS** navigation tab and then click **Add** (+).
3. In the Edit Customization Class dialog, search for or navigate to the customization classes you have already created.
4. Select the appropriate classes and click **OK**.
5. After you have added all of the customization classes, you can use the arrow icons to arrange them in the appropriate order.

Figure 11-9 shows the Overview editor for the `adf-config.xml` file with two customization classes added.

Figure 11-9 adf-config.xml Overview Editor



The order of the `customization-class` elements defines the precedence of customization layers. For example, in the following code that represents the customization class order in the `adf-config.xml` file, the `IndustryCC` class is listed before the `SiteCC` class. This means that customizations at the industry layer are applied to the base application, and then customizations at the site layer are applied.

```
<adf-config xmlns="http://xmlns.oracle.com/adf/config">
   <adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
      <mds-config xmlns="http://xmlns.oracle.com/mds/config" version="11.1.1.000">
         <cust-config>
            <match path="/">
               <customization-class name="com.mycompany.IndustryCC"/>
               <customization-class name="com.mycompany.SiteCC"/>
            </match>
         </cust-config>
      </mds-config>
   </adf-mds-config>
</adf-config>
```

Upon completion, the customization classes are available to JDeveloper for customization and for running your project locally in JDeveloper. The customization classes will also be packaged to the EAR class path when you package the application.

11.6 Understanding a Customization Developer Role

Customization features are available only in the Customization Developer role that is used to customize the metadata in a project.

In JDeveloper, the Customization Developer role is used to customize the metadata in a project. Customization features are available only in this role. When working in a Customization Developer role, you can do the following:

- Create and update customizations.
- Select and edit the tip layer of a customized application.

- Remove existing customizations.

When you use JDeveloper in the Customization Developer role, the Source editor is read-only and the following JDeveloper features are disabled:

- Workspace migration.
- Creation, deletion, and modification of application and IDE connections. You must configure connections in the Default role before opening an application in Customization Developer role.

When working with an application in the Customization Developer role, new objects and files cannot be created, and noncustomizable objects cannot be modified. In addition, you cannot edit noncustomizable files, such as Java classes, resource bundles, security policies, deployment descriptors, and configuration files.

Note:

When you are working in the Customization Developer role, noncustomizable files are indicated by a lock icon .

You are also restricted from modifying project settings, and you cannot refactor or make changes to customizable files that would, in turn, necessitate changes in noncustomizable files.

See the Working with JDeveloper Roles in *Developing Applications with Oracle JDeveloper*.

11.6.1 How to Switch to the Customization Developer Role in JDeveloper

Users who did not select the Customization Developer role when starting JDeveloper can follow the given steps to switch to the role even after JDeveloper has been started.

The customization features of JDeveloper are available to you in the Customization Developer role. To work in this role, you can either select it when you start JDeveloper or, if JDeveloper is already running, you can use the **Switch Roles** menu to switch to the Customization Developer role.

To switch to the Customization Developer role in JDeveloper:

From the main menu in JDeveloper, select **Tools**, then **Switch Roles**, and then **Customization Developer**.

Optionally, you can select **Tools**, and then **Switch Roles** to toggle the **Always Prompt for Role Selection at Startup** menu to specify whether or not you want to select the role when JDeveloper is launched. If deselected, JDeveloper launches in the role in which it was last closed.

11.6.2 What You May Need to Know About the Tip Layer

Changes made by users in the Customization Developer role are applied to the tip layer, the combination of the layer and layer value that is selected in the Customization Context window.

When working in the Customization Developer role, the layer and layer value combination that is selected in the Customization Context window is called the tip layer. The changes you make while in the Customization Developer role are applied to this layer.

Note:

When working in the Customization Developer role, if the Customization Context window is not displayed, you can access it from the JDeveloper Window menu.

The metadata displayed in the JDeveloper editors is a combination of the base metadata and the customization layers up to and including the tip layer, according to the precedence set in `adf-config.xml`, with the values specified in the Customization Context window for each layer.

When working in the Customization Developer role, you can also see the noncustomized state of the application. When you select **View without Customizations** in the Customization Context window, there is no current tip layer. Therefore, what you see is the noncustomized state. While you are in this view, all customizable files show the lock icon (in the Applications window), indicating that these files are read-only.

When you make customizations in a tip layer, these customizations are indicated by an orange icon in the Properties window. A green icon indicates non-tip layer customizations. When you see an orange icon beside a property, you have the option of deleting that customization by selecting Remove Customization from the drop-down menu for that property.

11.7 Enabling Customizations in Resource Bundles

In the Studio Developer role, create an application resource bundle or a project resource bundle, and then edit it to customize it.

To implement customization for resource keys, you must create additional resource bundle files (You cannot use the base resource bundle file.).

In the Studio Developer role, create one of the following:

- An application resource bundle (see How to Create an Application Resource Bundle).
- A project resource bundle (see How to Create a Project Resource Bundle).

Edit the bundle that you create to define string values for resource keys.

Before you begin:

See How to Use Multiple Resource Bundles in *Developing Fusion Web Applications with Oracle Application Development Framework*.

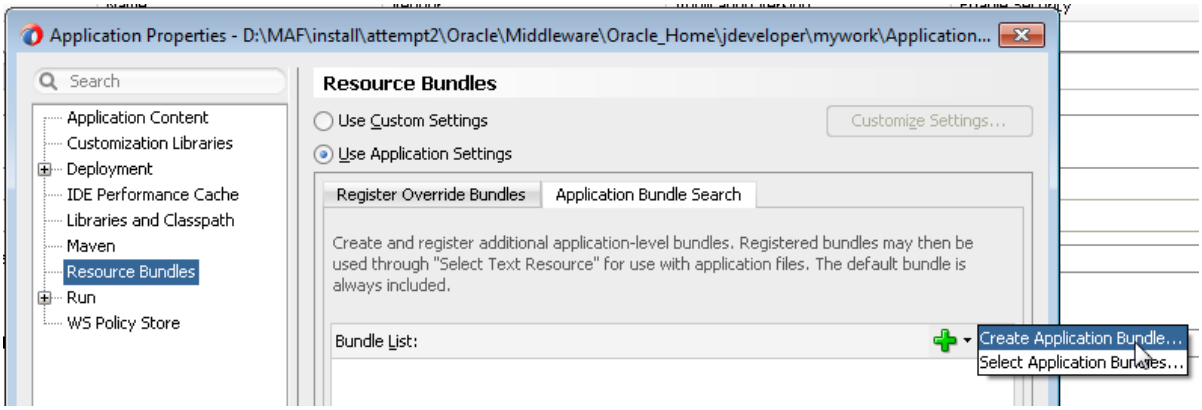
11.7.1 How to Create an Application Resource Bundle

In the Studio Developer role, use the procedure to create an application resource bundle from the Resource Bundle page.

To create an application resource bundle:

1. In the Studio Developer role, click **Application**, then **Application Properties**, and then **Resource Bundles**.
2. In the Resource Bundle page, click **Application Bundle Search**, then click the drop-down menu icon to the right of the **Add bundle** icon and select **Create Application Bundle**, as shown in Figure 11-10.

Figure 11-10 Creating an Application Resource Bundle



3. In the Create Xliff File dialog that appears, enter a name for the resource bundle and click **OK**.
4. Edit the resource bundle, as described in Editing Resources in Resource Bundles.

Note:

MAF does not support the **Overridden** property in the application-level Resource Bundle page.

---

5. In the Customization Developer role, open the Select Text Resource dialog and select from among the resource bundles that contain the appropriate string. Because you cannot change strings or create new ones in the Customization Developer role, you can only select from the strings in the selected bundle.

Note:

Do not select strings from the base resource bundle in the Customization Developer role, as doing so may cause problems when upgrading the application.

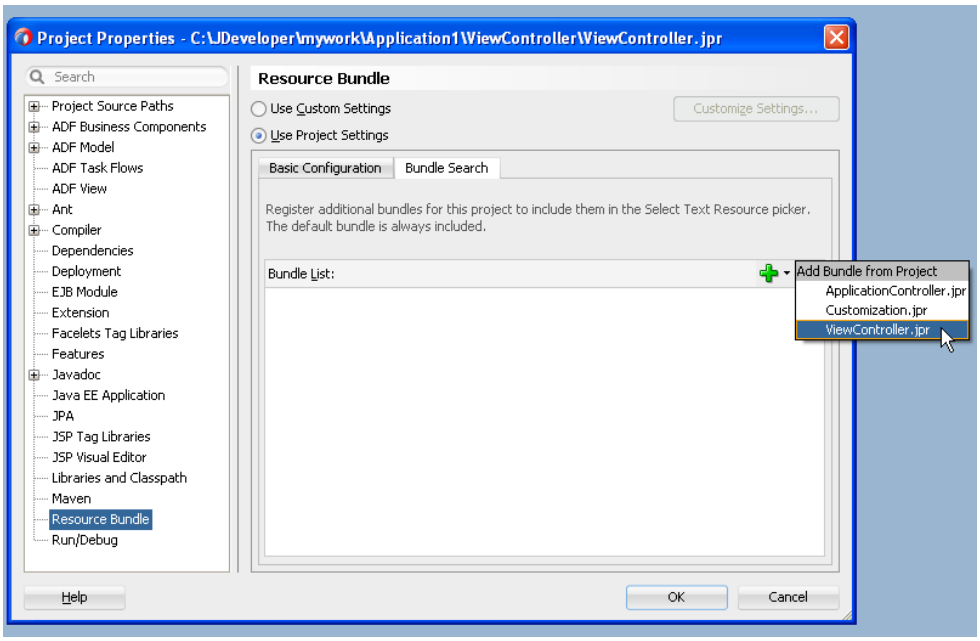
11.7.2 How to Create a Project Resource Bundle

Use the procedure to create a project resource bundle by selecting the **General XML** option from the New Gallery menu.

To create a project resource bundle:

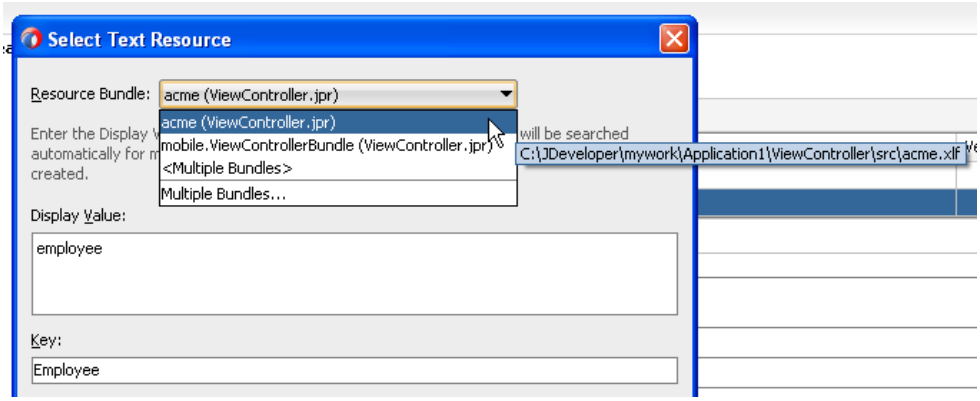
1. In the Studio Developer role, right-click the project where you want to create the resource bundle and select **New**, then **From Gallery**, followed by **General XML**, and **XML Localization File (XLIFF)**.
2. In the Create Xliff File dialog that appears, enter a name for the resource bundle and click **OK**.
3. Edit the resource bundle, as described in Editing Resources in Resource Bundles.
4. In the **Bundle Search** tab of the Resource Bundle page, register the resource bundle by selecting a project (`.jpr`) file, as shown in Figure 11-11.

Figure 11-11 Selecting a Resource Bundle



Registering a resource bundle includes it in the Select Text Resource dialog, shown in Figure 11-12.

Figure 11-12 Selecting a Resource Bundle for a Text Resource



5. Use the Select Text Resource dialog to define the key as follows:

   a. Select the bundle from the **Resource Bundle** drop-down list.

      The dialog displays the strings that are currently defined in the selected resource bundle.

   b. Enter a new string and then click **Save and Select**.

      JDeveloper writes the string to the selected resource bundle.

6. In the Customization Developer role, open the Select Text Resource dialog and select from among the resource bundles that contain the appropriate string. Because you cannot change strings or create new ones in the Customization Developer role, you can only select from the strings in the selected bundle.

Note:

Do not select strings from the base resource bundle in the Customization Developer role, as doing so may cause problems when upgrading the application.

11.8 Upgrading a MAF Application with Customizations

Use the procedure to use the Upgrade Mobile Application from Archive wizard and upgrade an application to a higher version while retaining the customizations.

Customizations are upgrade-safe because they are saved separately from the base applications. Because customizations retain changes, they enable you to upgrade an application by applying these changes to newer versions of the application. The MAF Application Archive (`.maa`) file provides the mechanism for upgrading MAF applications. When you create an application from an `.maa` file, you can upgrade the application using an updated version of the `.maa` file.

Using the Upgrade Mobile Application from Archive wizard, you can upgrade an application to a higher version while retaining the customizations made prior to the upgrade.

Before you begin:

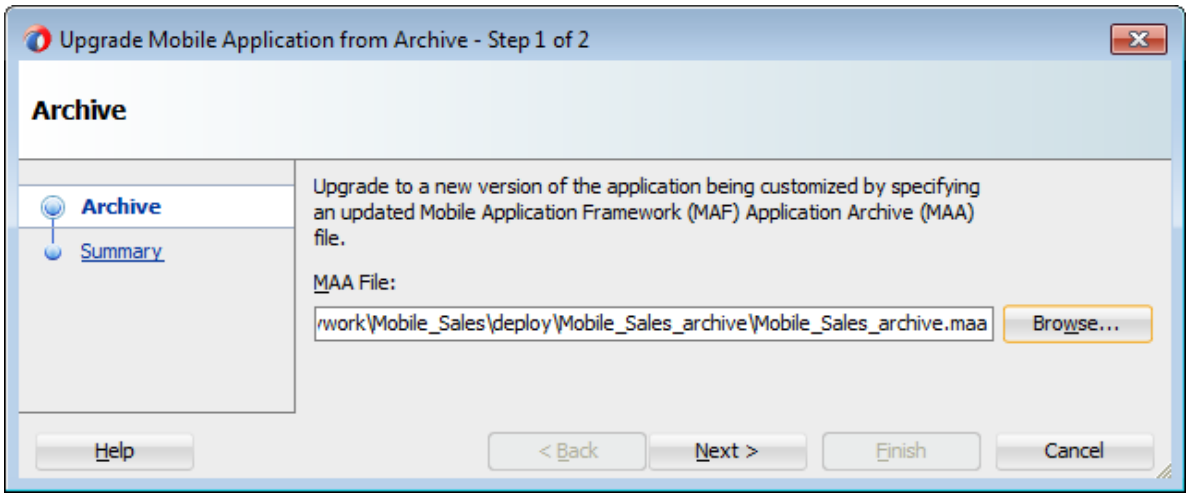
You may want to familiarize yourself with the MAF Application Archive (`.maa`) file. See Creating a Mobile Application Archive File and Creating a New Application from an Application Archive.

Ensure that the application that is packaged into the `.maa` file and used for the upgrade has the same application ID as the application to which it will be applied. It must also have a higher version number than the application targeted for the upgrade.

To upgrade a MAF application:

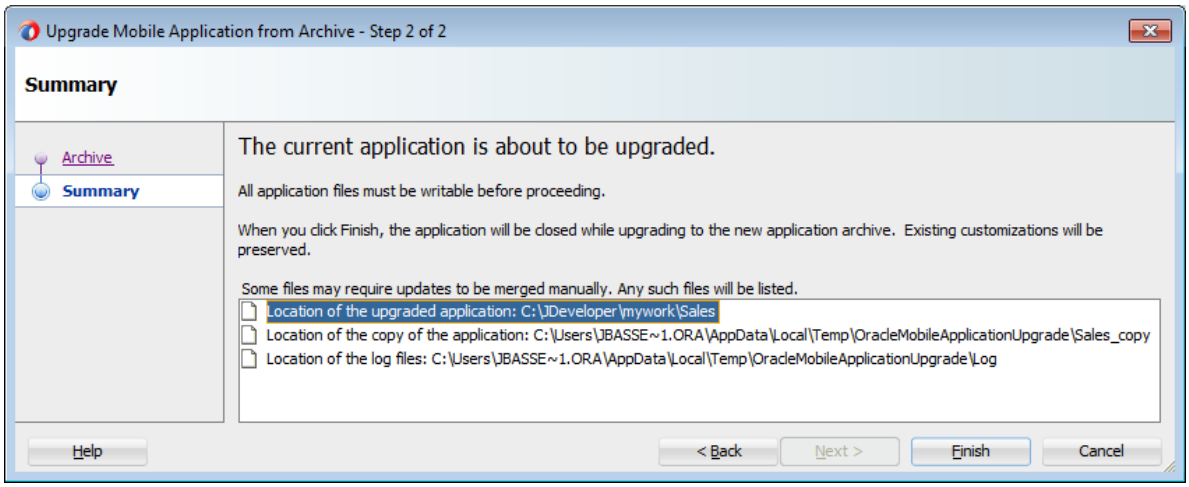
1. Create a MAF application from an `.maa` file.
2. Apply customization to the MAF application, as described in Enabling Customizations in Resource Bundles.
3. Click **Application**, and then select **Select Mobile Application from Archive**.
4. Browse to and select the `.maa` file. The wizard discontinues the upgrade if the application packaged in the `.maa` has the same (or lower) version number than the current application, or a different application ID.

Figure 11-13 Selecting the .maa File



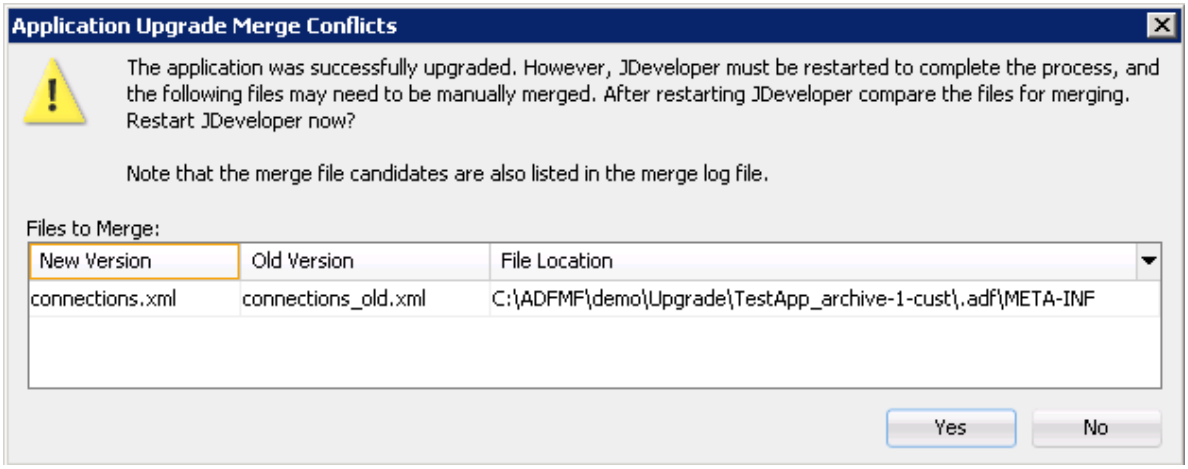
5. Review the Summary page for files that require a manual merge. As noted in Figure 11-14, MAF saves the initial version (Version 1) of the application in the `Temp` directory. The Summary page also notes the temporary location of the log files.

Figure 11-14 Application Upgrade Information



6. If the upgrade completes successfully, restart JDeveloper. JDeveloper notifies you if different versions of a configuration file require reconciliation, as shown in Figure 11-15.

Figure 11-15 Manual Merge Notification



During the upgrade, MAF copies a set of files that cannot be customized for both Version 1 of the application and Version 2 (the upgraded version of the application). These files include the `connections.xml` and `adf-config.xml` files. If MAF detects differences between Version 1 and Version 2 `connections.xml` and `adf-config.xml` files, it retains both copies of these files and writes an entry to the merge log file. MAF differentiates Version 1 by appending a version number to the file name if version numbers exist. If version numbers do not exist, MAF adds `_old` to the file name, as illustrated by `connections_old.xml` in Figure 11-15. If needed, you can manually merge the differences into the new version. As shown in Figure 11-16, MAF places the merge file log in the temporary location noted in the Summary page. MAF names the files as *`workspace name_timestamp`*.

Figure 11-16 The Merge Log File

```
Sun Nov 24 05:24:10 PST 2013
Please check the following files and manually merge the changes in the old version to the new version if needed

New Version             Old Version                              File Location
=========================================================================
connections.xml      connections old.xml               C:\ADFMF\demo\Upgrade\TestApp archive-1-cust\.adf\META-INF
```

11.8.1 What Happens in JDeveloper When You Upgrade Applications

When an application is upgraded with the Upgrade Mobile Application from Archive wizard, JDeveloper saves, imports, copies, and updates relevant files.

In addition to copying Version 1 to the `Temp` directory and creating Version 1 and Version 2 copies of the non-upgradable configuration files, MAF also performs the following when you upgrade an application using the Upgrade Mobile Application from Archive wizard:

- Saves the libraries and resource bundles settings for each project in a map keyed with the project file name.
- Saves the resource bundle settings for the workspace.
- Saves the registered customization class in the `adf-config.xml` file.
- Imports the Version 2 `.maa` file to the temporary directory.
- Copies the application from the `.maa` file used for the upgrade to Version 1.

- Updates each Version 2 project (`.jpr`) file with the registered resource bundle and library dependency map. The new version of the library overrides the previous version. However, the Version 1 library remains unchanged if it shares the same name as the library used in Version 1.
- Updates the Version 2 workspace (`.jws`) file with the registered resource bundle settings.
- Updates the Version 2 `adf-config.xml` file to register the customization class.

11.8.2 What You May Need to Know About Upgrading FARs

A FAR file that was not included in the `.maa` file that was used to create or upgrade an application must be upgraded separately, by itself.

If the application includes a FAR file that was not packaged in the original `.maa` file that was used to create the application (or included in the `.maa` file that is used to upgrade the application), then you must upgrade the FAR file separately. For example, you can create an application from a `.maa` file, add a FAR file, and then perform customization. You can upgrade the application to use a newer version of the FAR by adding the updated FAR from the Resources window as described in Using FAR Content in a MAF Application.

12

Using Lifecycle Listeners in MAF Applications

This chapter describes the lifecycle listeners that MAF provides for you to write code that can execute in response to events in your MAF application or application features.

This chapter includes the following sections:

- Introduction to Lifecycle Listeners in MAF Applications
- Registering a Lifecycle Listener for a MAF Application or an Application Feature
- What Happens When You Register a Lifecycle Listener

12.1 Introduction to Lifecycle Listeners in MAF Applications

Lifecycle listeners contain code that runs in response to specific application events. Review the information about the interfaces to be implemented for communication with event notifications, and how they can create lifecycle listeners.

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 the database of the application when the application starts, as described in Using the Local SQLite Database, or to update a security configuration from URL parameters, as described in 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:

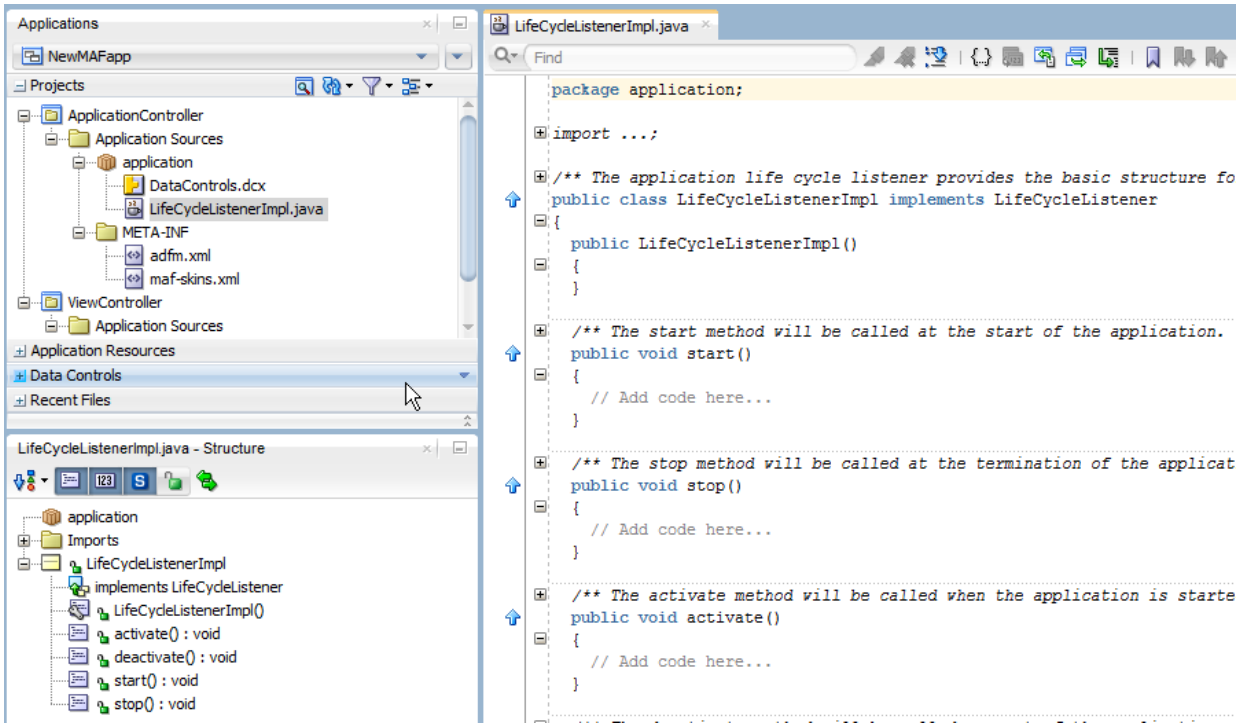
  - `activate()`

- `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 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 ApplicationController project in the application , as shown in Figure 12-1.

Figure 12-1 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 12-1 describes the specific times that MAF invokes application lifecycle methods during an application startup, shutdown, and hibernation.

Table 12-1 Timing of the MAF 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.

Table 12-1 (Cont.) Timing of the MAF Invocation of Application Lifecycle Methods

Method	Timing	When Called	Usage
`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 12-2 describes the specific times that MAF invokes feature lifecycle methods during a feature activation and deactivation.

Note: MAF also provides another interface (`oracle.maf.api.feature.FeatureLifeCycleListener`) that extends `oracle.adfmf.feature.LifeCycleListener` and provides additional methods to write code before and after your restart an application feature, as described in Restarting an Application Feature in a MAF Application.

Table 12-2 Timing of the MAF 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.

Table 12-2 (Cont.) Timing of the MAF Invocation of Feature Lifecycle Methods

Method	Timing	When Called	Usage
`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`, `oracle.adfmf.feature.LifeCycleListener`, and `oracle.maf.api.feature.FeatureLifeCycleListener` interfaces, see *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.xml` of the MAF application and `maf-feature.xml` files. For more information about this and other sample applications, see MAF Sample Applications.

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

Use the procedure to register an application lifecycle listener using the overview editor for the `mafapplication.xml` file, and register a feature lifecycle listener using the overview editor for the `maf-features.xml` file.

You register an application lifecycle listener by using the overview editor for the `maf-application.xml` file and a feature lifecycle listener using the overview editor for the `maf-features.xml` file.

To register an application lifecycle listener:

1. In the Applications window, expand the Application Resources panel.
2. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click **maf-application.xml**.
4. In the **Application** navigation tab, specify the Java class that implements the `oracle.adfmf.application.LifeCycleListener` interface in the **Lifecycle Event Listener** field. By default, this is set to `application.LifeCycleListenerImpl`.

   If you want to package the application lifecycle listener in a JAR library that will be distributed for use elsewhere, you might use a custom class different from the default implementation provided by MAF.

To register an application feature lifecycle listener:

1. In the Applications window, expand the **ViewController** project, **Application Sources**, and then **META-INF**.

2. Double-click the `maf-feature.xml` file.
3. In the Features list, select the feature for which you want to register a feature lifecycle listener.
4. In the **Lifecycle Event Listener** field, specify the Java class that implements the `oracle.adfmf.feature.LifeCycleListener` interface.

12.3 What Happens When You Register a Lifecycle Listener

The `application.LifeCycleListenerImpl.java` class in the ApplicationController project, which is registered by the listener-class attribute in the `maf-application.xml` file, implements the 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 ApplicationController 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>
```

JDeveloper 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 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>
```

13

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:

- Introduction to the MAF AMX Application Feature
- Creating Task Flows
- Creating Views

13.1 Introduction to the MAF AMX Application Feature

The MAF AMX framework works within MAF to provide the UI components necessary to create a feature that behaves identically on all platforms.

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:

- Getting Started with MAF Application Development
- Creating the MAF AMX User Interface
- Using Bindings and Creating Data Controls in MAF AMX

13.2 Creating Task Flows

Task flows define how users navigate between AMX pages in an application. You can create MAF AMX application features that have both bounded and unbounded task flows.

Task flows allow you to define the navigation between MAF AMX pages. Using your application workspace in JDeveloper (see 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 `ViewController-task-flow.xml` (see What You May Need to Know About the ViewController-task-flow.xml File).

A MAF sample application called Navigation (located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer) 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 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 Passing Parameters to a Bounded Task Flow) and generating return values (see 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. Like the overview editor for task flows, this tool includes a diagrammer (see 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 What You May Need to Know About Task Flow Activities and Control Flows) from the Components window. 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.

You can add each task flow as an application feature to your MAF application. See Defining the Application Feature Content as a MAF AMX Page or Task Flow.

13.2.1 How to Create a Task Flow

Create a bounded task flow from the `maf-feature.xml` file, or follow the steps in the task to create a task flow from the New Gallery.

A task flow is composed of the task flow itself and a number of activities with control flow rules between those activities (see 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 use the navigation diagrammer to declaratively create a bounded task flow for your MAF AMX application feature. When you use the diagrammer, JDeveloper creates the XML metadata needed for navigation to work in your MAF AMX application feature in the `ViewController-task-flow.xml` file (default).

Before you begin:

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

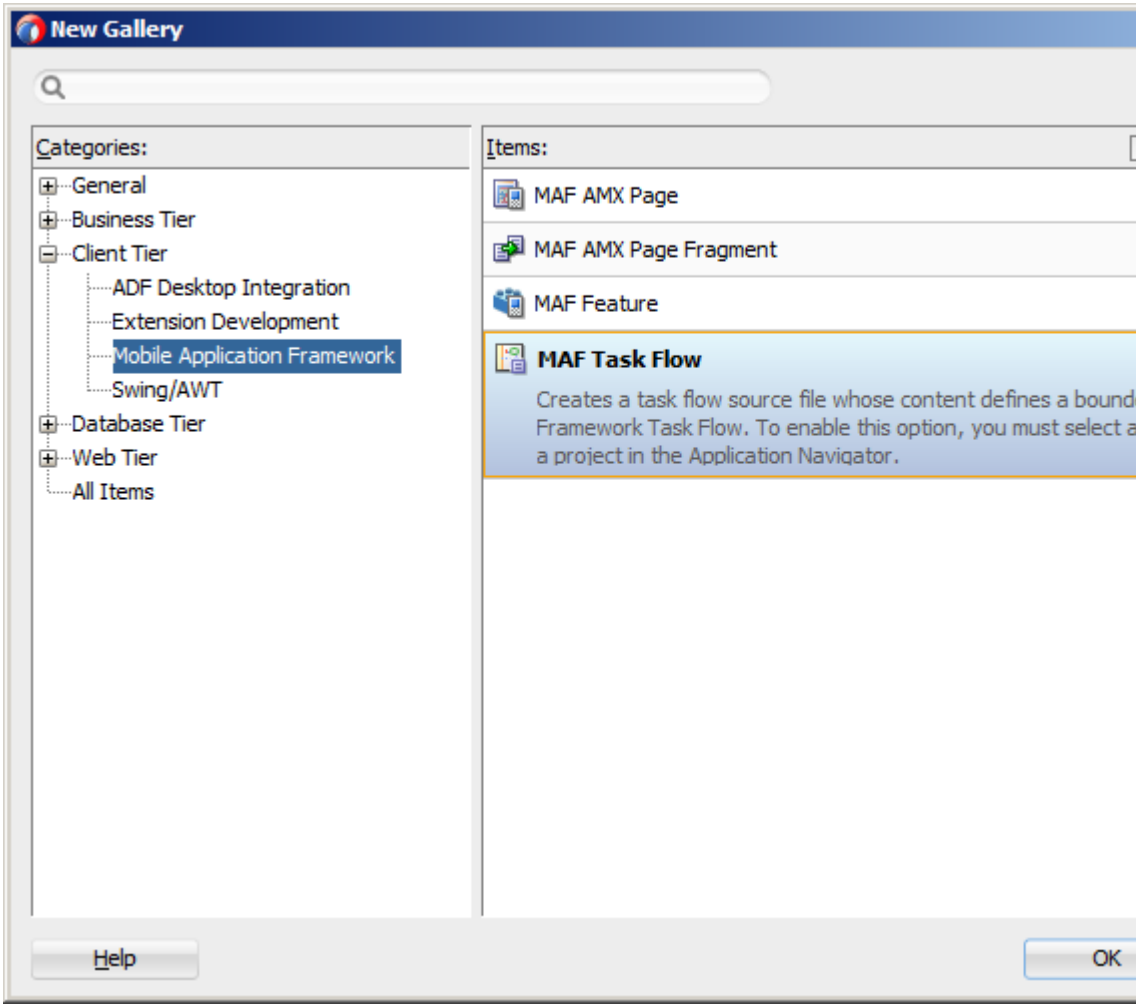
There are two ways to create a task flow in MAF:

- You can create a bounded task flow from the `maf-feature.xml` file. For more information, see Defining the Application Feature Content as a MAF AMX Page or Task Flow.
- You can use the New Gallery in JDeveloper. For more information, refer to the following procedure.

To create a task flow from the New Gallery:

1. From the top-level menu in JDeveloper, click **File**, and then select **New > From Gallery**.
2. In the **New Gallery**, expand the **Client Tier** node, select **Mobile Application Framework** , and then **MAF Task Flow** (see Figure 13-1). Click **OK**.

Figure 13-1 Creating New MAF Task Flow



3. In the **Create MAF Task Flow** dialog (see Figure 13-2), specify the file name and location for your new task flow, and then click **OK** to open the new `<Name>-flow.xml` file in the navigation diagrammer that Figure 13-3 shows.

Note:

Task flows should be created within the HTML root of the View Controller project of your MAF application.

Note:

JDeveloper increments the number of the task flow according to the number of bounded task flows that already exist in the same pattern.

Figure 13-2 Create MAF Task Flow Dialog

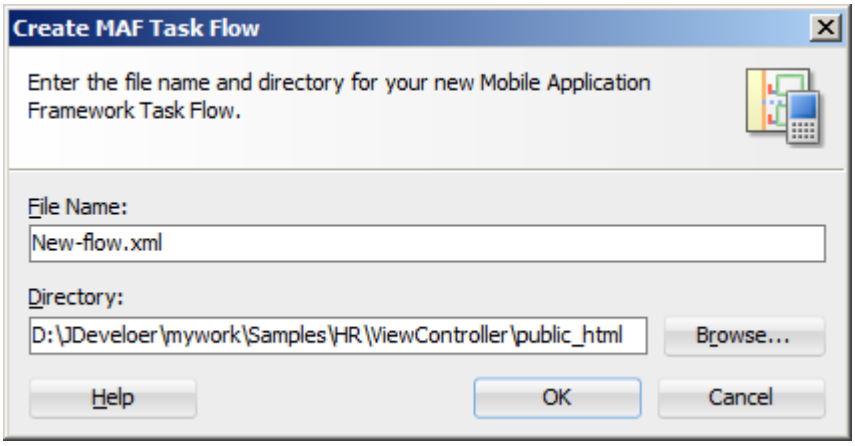
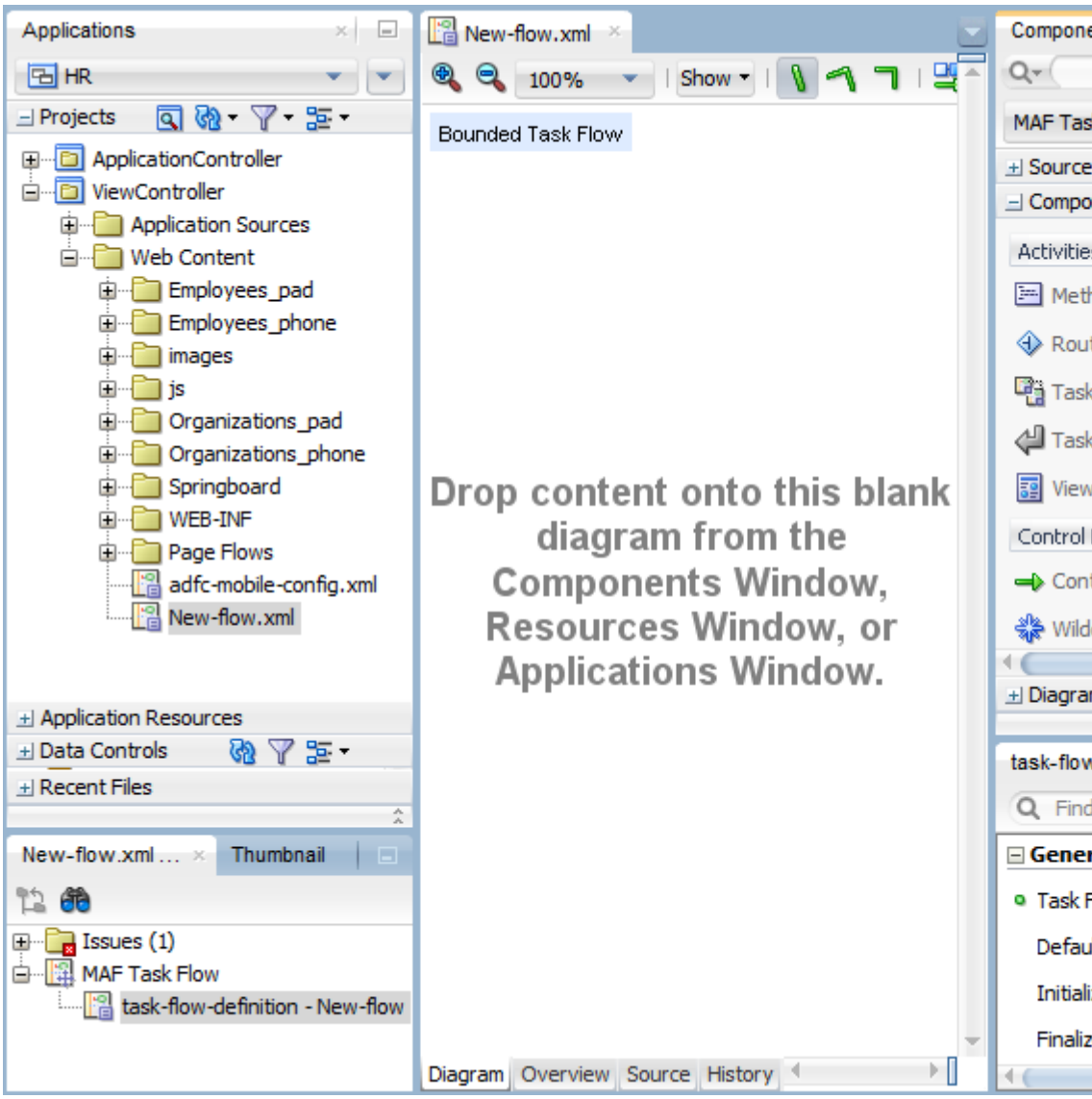


Figure 13-3 New Blank Task Flow



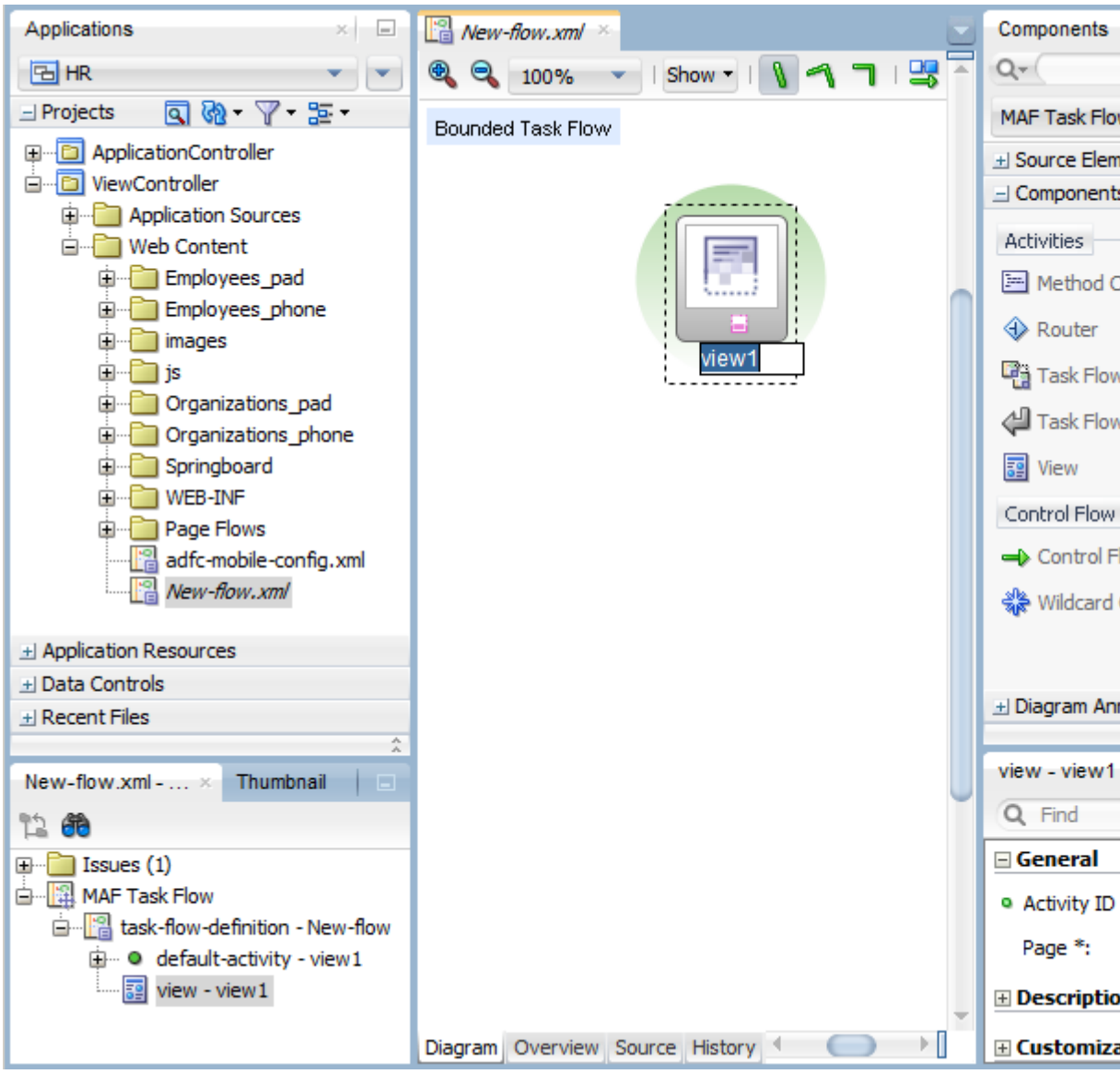
4. In the **Components** window, select **MAF Task Flow**.

   Tip:

   If the **Components** window is not displayed, choose **Window > Components** from the main menu. By default, the **Components** window is displayed in the upper right-hand corner of JDeveloper.

5. From **MAF Task Flow > Components**, select the component you wish to use and drag it onto the diagram. JDeveloper redraws the diagram with the newly added component, as Figure 13-4 shows.

Figure 13-4 Adding Components to Task Flow

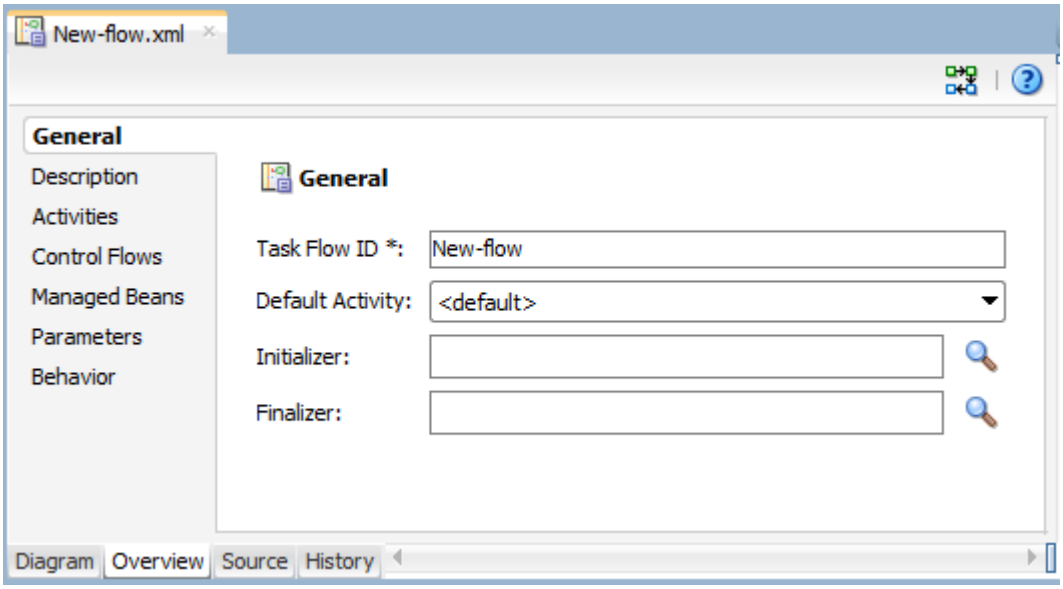


For information on how to add activities to a task flow, see How to Add and Use Task Flow Activities.

For information on how to add control flows, see How to Define Control Flows.

For information on how to define behavior of the new task flow, see What You May Need to Know About Behavior of New Bounded Task Flows.

You can also open the **Overview** tab and use the overview editor to create navigation rules and navigation cases. Press F1 for details on using the overview editor to create navigation.



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, JavaScript 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 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 Creating Views .

13.2.1.1 What You May Need to Know About Behavior of New Bounded Task Flows

When a new bounded task flow is created, MAF adds a `page-flowscope-behavior` element to the `Name-flow.xml` file. Setting the element to push-new, creates a new page flow scope, whereas setting it to preserve persists the page flow variables when the task flow is executed.

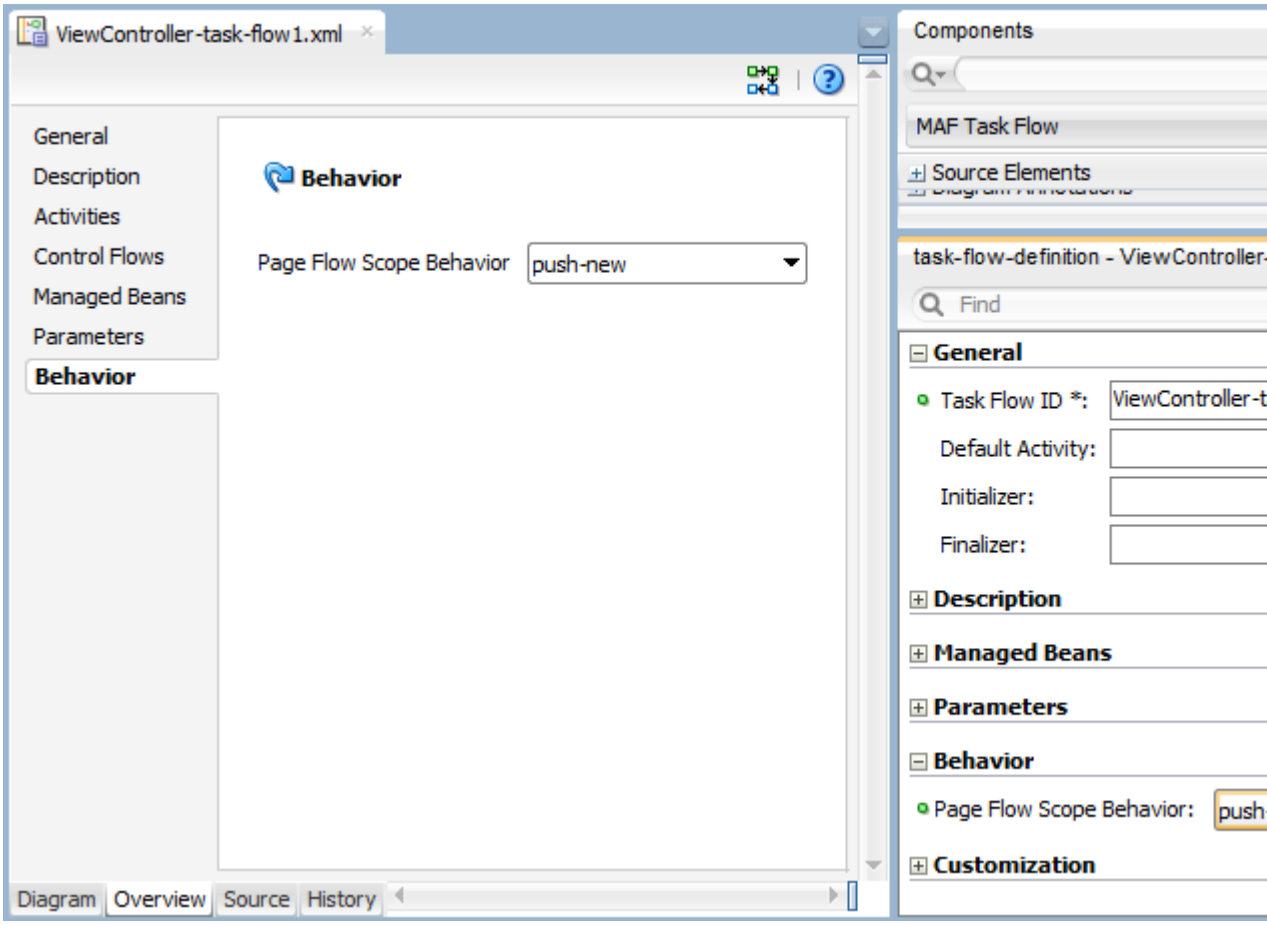
When a new bounded task flow is created, MAF automatically adds a `page-flow-scope-behavior` element to the *<Name>*`-flow.xml` file. This element is added as a child of the top-level `task-flow-definition` element.

Note:

The `page-flow-scope-behavior` element is appended to all newly created task flows, even if they are created in projects built using previous versions of MAF.

The value of the `page-flow-scope-behavior` element is set to `push-new` by default and is displayed in the **Overview** and **Source** editors for the new task flow, as well as the **Properties** window for the `task-flow-definition` element, as Figure 13-5 shows.

Figure 13-5 Page Flow Scope Behavior for Task Flows



If the **Page Flow Scope Behavior** is set to `push-new`, a new page flow scope is created and the old `pageFlowScope` variables are saved and pushed on to a stack. This allows for the previous page flow scope to be restored upon the execution of a task flow return. If the **Page Flow Scope Behavior** is set to `preserve`, the `pageFlowScope` variables are not cleared when the task flow is entered upon execution of a task flow call resulting in the new task flow variables containing old values.

In existing task flows, if the `page-flow-scope-behavior` element is not present, then you should define it as either `push-new` or `preserve`.

For more information about the `pageFlowScope`, see About the Managed Beans Category.

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

Task flows use activities and control flow cases to define the transitions between activities.

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 13-1.

Table 13-1 Task Flow Activities

Activity	Description
View	Displays a MAF AMX page. See 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 a MAF AMX application feature to invoke logic based on control flow rules. For additional information, see 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 a 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. See 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. See 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 (see Reusing MAF Application Content .). The task flow call activity supports task flow input parameters and return values. See Adding Task Flow Call Activities.
Task Flow Return	Identifies the point in the control flow of an application 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. See Adding Task Flow Return Activities.

The MAF Task Flow designer supports control flows listed in Table 13-2.

Table 13-2 Control Flows

Control Flow	Description
Control Flow Case	Identifies how control passes from one activity to the next in the MAF AMX application feature. See 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. See Adding a Wildcard Control Flow Rule.

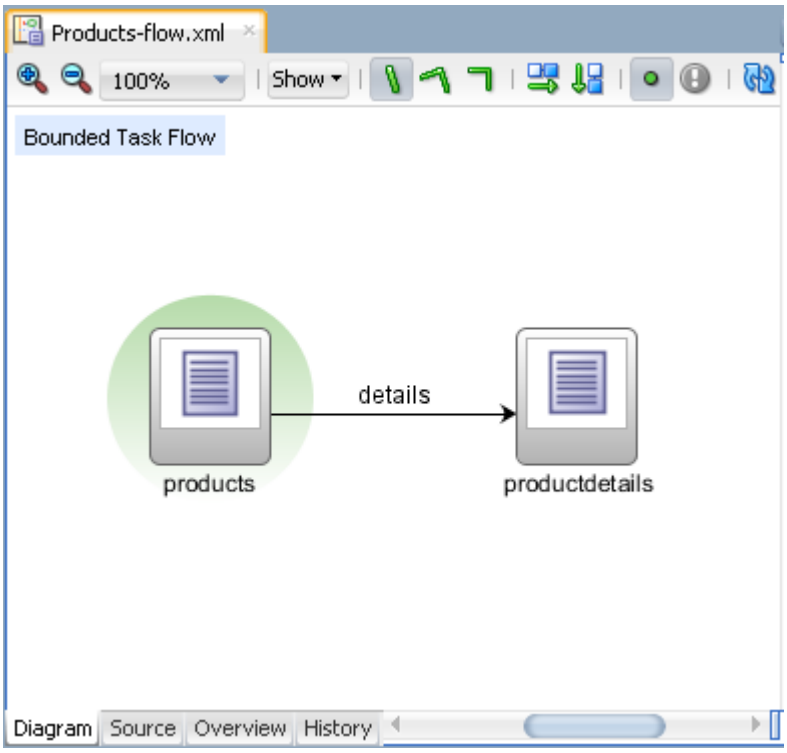
13.2.3 What You May Need to Know About the ViewController-task-flow.xml File

Open the `ViewController-task-flow.xml` file in an editor to design the interactions between MAF AMX pages .

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

Figure 13-6 shows a sample task flow file called `Products-flow.xml`. In this file, the control flow is directed from the `products` page to the `productdetails` page. To return to the `products` page from the `productdetails` page, the built-in `__back` navigation is used (see What You May Need to Know About MAF Support for Back Navigation).

Figure 13-6 Task Flow File



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

The MAF Task Flow diagrammer is an editor in which you can design a feature with activities and task flows.

As illustrated in Figure 13-6, the task flow diagram and **Components** window 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 **Components** window or from the **Applications** window. See How to Add and Use Task Flow Activities.

13.2.5 How to Add and Use Task Flow Activities

Follow the steps in the task to add a Task Flow activity in an open task flow source file. Drag an activity from the available components to the diagram.

Before you begin:

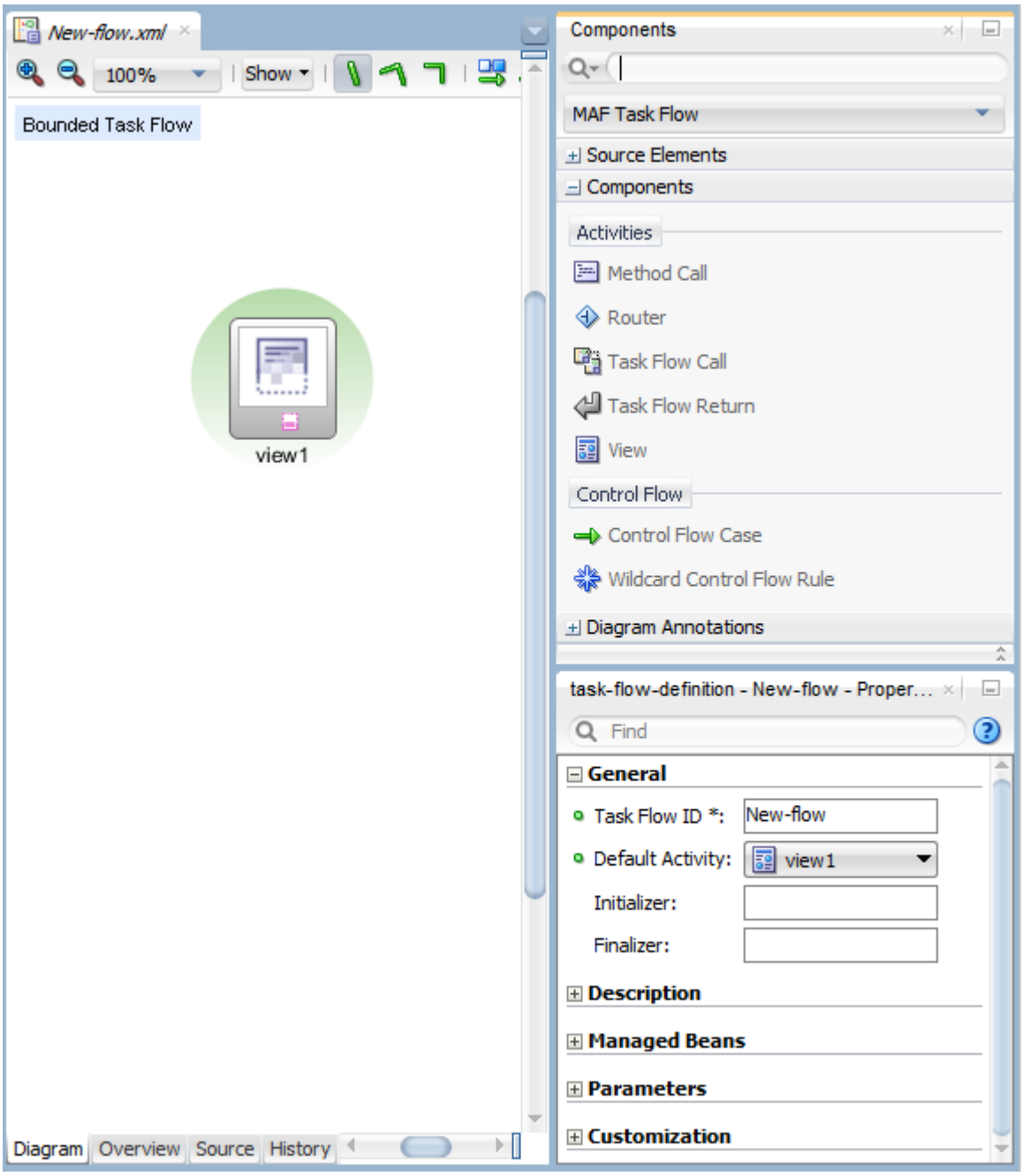
You must select **MAF Task Flow** from the **Components** window, as Figure 13-7 shows.

To add an activity to a MAF task flow:

1. In the **Applications** window, double-click a task flow source file (such as `ViewController-task-flow.xml`) to display the task flow diagram and the **Components** window, as Figure 13-7 shows. The diagrammer displays the task flow editor. The **Components** window automatically displays the components available for a MAF task flow.

2. Drag an activity from the **Components** window onto the diagram. If you drag a view activity onto the diagram and double-click on it, you can invoke the **Create MAF AMX Page** wizard (see Adding View Activities).

Figure 13-7 The Diagrammer for the Task Flow Editor



Note:

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

13.2.5.1 Adding View Activities

View activity is a task flow activity that displays a MAF AMX page. You can configure view activities in your task flow so that one activity passes control to another activity at runtime.

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 Action attribute of the command component 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 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 Action attribute of the component is an action outcome that always triggers the same control flow case. When an 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 Action attribute of the component 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 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.

The following two examples demonstrate what happens when you pass control between View activities:

1. This example shows a control flow case defined in the XML source file for a bounded or unbounded task flow.

```
<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>
```

2. In this example, a Button on a MAF AMX page associated with the Start view activity specifies `toOffices` as the `action` attribute. When the user clicks the button, control flow passes to the `WesternOffices` activity specified as the `to-activity-id` in the control flow metadata.

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

See:

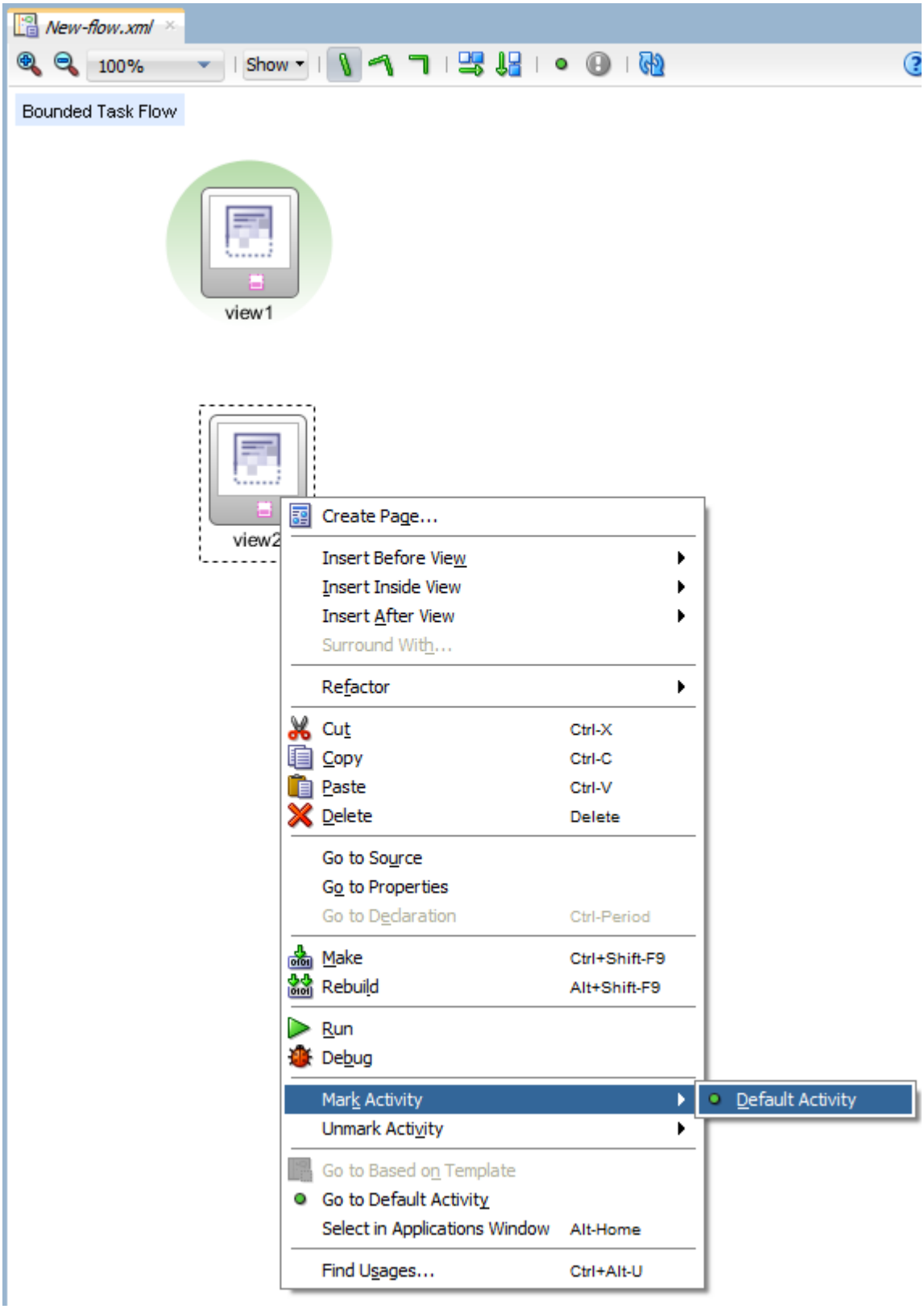
- Adding View Activities
- Creating MAF AMX Pages

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 **Components** window. You can create an actual MAF AMX page by double-clicking the View activity in the **Diagram** window and then define characteristics for the page through the displayed dialog. You can also create a View activity by dragging and dropping a MAF AMX file in the **Applications** window onto the **Diagram** tab of the **Overview** editor.

If you are creating a bounded task flow, you may want to designate a specific activity as the default activity (see 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, JDeveloper 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 select **Mark Activity** , and then **Default Activity**(see Figure 13-8).

Figure 13-8 Defining Default Activity



13.2.5.2 Adding Router Activities

If a routing condition can be expressed in an EL expression, use a router activity, which uses the runtime evaluation of EL expressions, to route control to activities. You can create an expression for which you can configure properties and add router cases.

Each control flow corresponds to a different router case. Each router case uses the following elements to select 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.

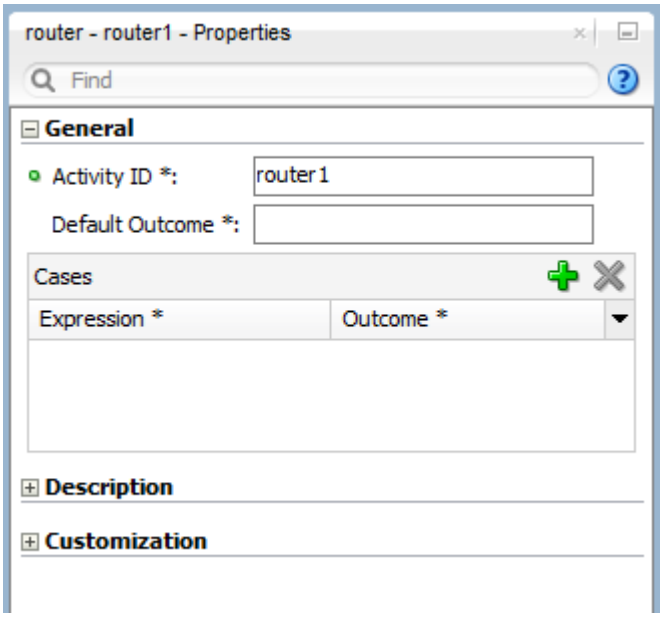
When you drag a Router activity onto the diagram, you can use the **Properties** window to create an expression whose evaluation determines which control flow rule to follow. Using the **Properties** window, you configure the **Activity ID** and **Default Outcome** properties of the router activity and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

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

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 **Cases** section that Figure 13-49 shows.

Figure 13-9 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, JDeveloper writes values to the source file of the task flow based on the values that you specify for the properties of the router activity.

13.2.5.3 Adding Method Call Activities

A method call activity calls a custom or built-in method to invoke the logic of an AMX application feature from within the control flow of the feature. You can add a Method Call activity, and use the Properties window to configure 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 control flow of the application feature. 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 (see Figure 13-10).

- `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 (see Figure 13-10).

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.

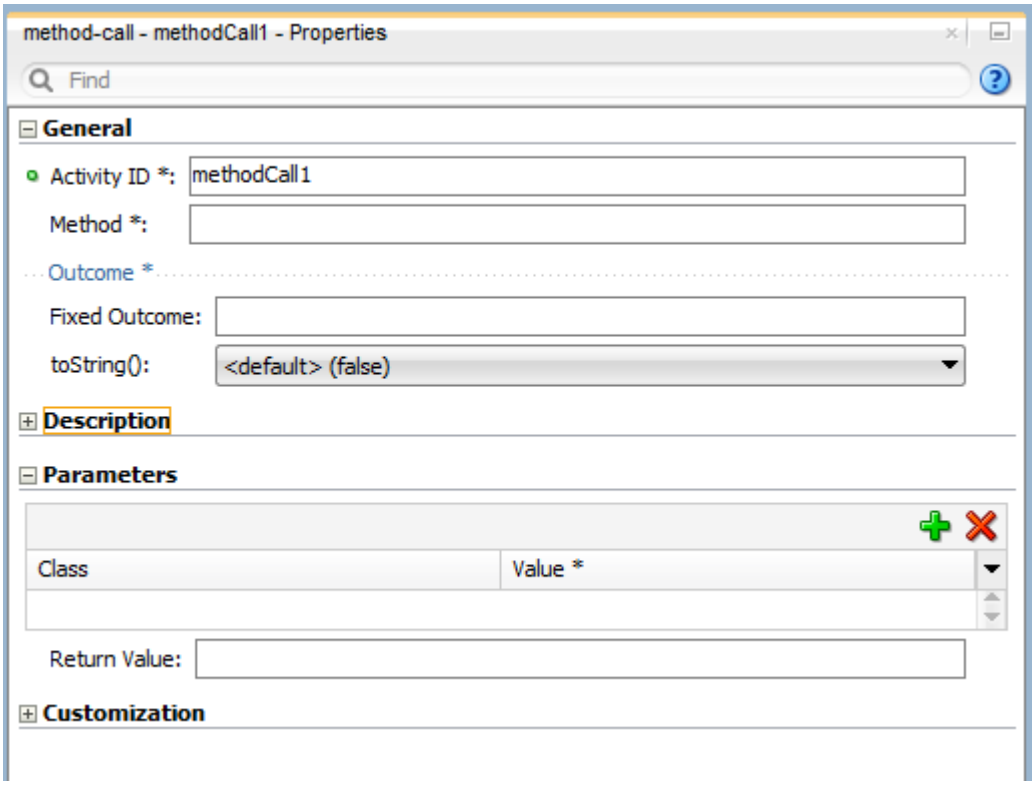
You configure the method call by modifying its activity identifier in the **Activity ID** field if you want to change the default value. If you enter a new value, the new value appears under the method call activity in the diagram.

In the **Method** field, enter an EL expression that identifies the method to call.

Note:

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. See What You May Need to Know About Generated Drag and Drop Artifacts.

Figure 13-10 Configuring Method Call Activity



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

- Select **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 (see Figure 13-10). 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}
  ```

13.2.5.4 Adding Task Flow Call Activities

A task flow call activity calls a bounded task flow from either a bounded task flow or an unbounded task flow. Follow the steps in the task to call a bounded task flow, and then specify input parameters on a task flow call activity.

You can use a task flow call activity to call a bounded task flow from either the unbounded task flow (see Unbounded Task Flows) or a bounded task flow (see 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. See 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 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 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 Specifying Input Parameters on a Task Flow Call Activity).

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

Follow the steps in the task to open the source file for a bounded task flow to select an activity, add a Task Flow Call activity, and configure its identifiers.

To call a bounded task flow:

1. Open a bounded task flow file in the **Diagram** view.
2. From the **Components** window, select **Components**, and then **Activities**.
3. Drag and drop a **Task Flow Call** activity onto the diagram.
4. Select 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 (see Figure 13-2) where you define settings for a new bounded task flow.
    - Drag an existing bounded task flow from the **Applications** window 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:

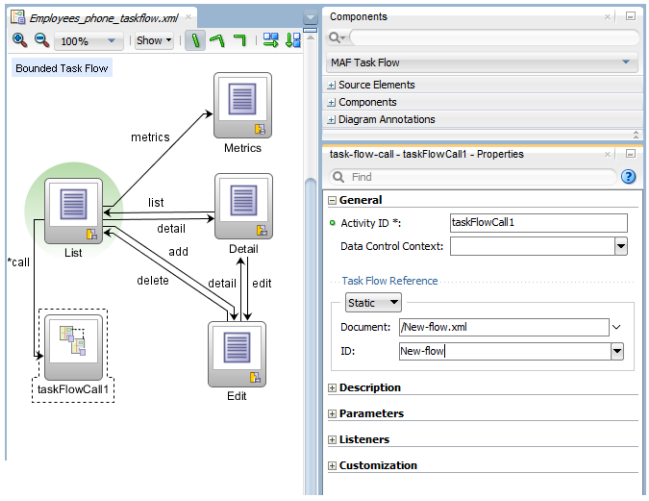
        a. In the **Diagram** view, select the **Task Flow Call** activity.

        b. In the **Properties** window shown in the following illustration, expand the **General** section, and then 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.

Figure 13-11 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 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.

13.2.5.4.2 Specifying Input Parameters on a Task Flow Call Activity

New input parameters can be specified for a selected Task Flow Call activity. Follow the steps in the task to configure input parameter definitions for the called bounded task flow and map parameters between a task flow call activity and its called bounded task flow.

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 **Applications** window and drop it on the task flow call activity. The task flow call activity input parameters are created automatically based on the input parameter definition of the bounded task flow. See 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. See Passing Parameters to a Bounded Task Flow.

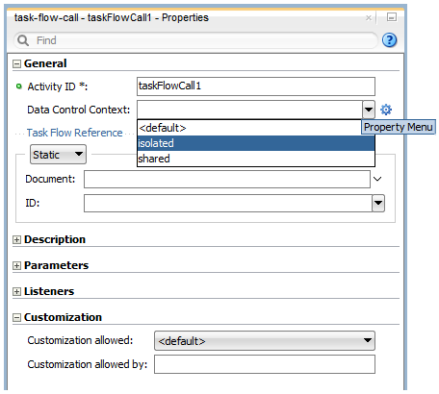
13.2.5.4.3 Specifying the Data Control Context

A shared data control context allows task flow calls to share data control whereas an isolated data control context allows the task flows to work independently. Follow the steps in the task to specify the Data Control Context property for a task flow of a selected Task Flow Call activity.

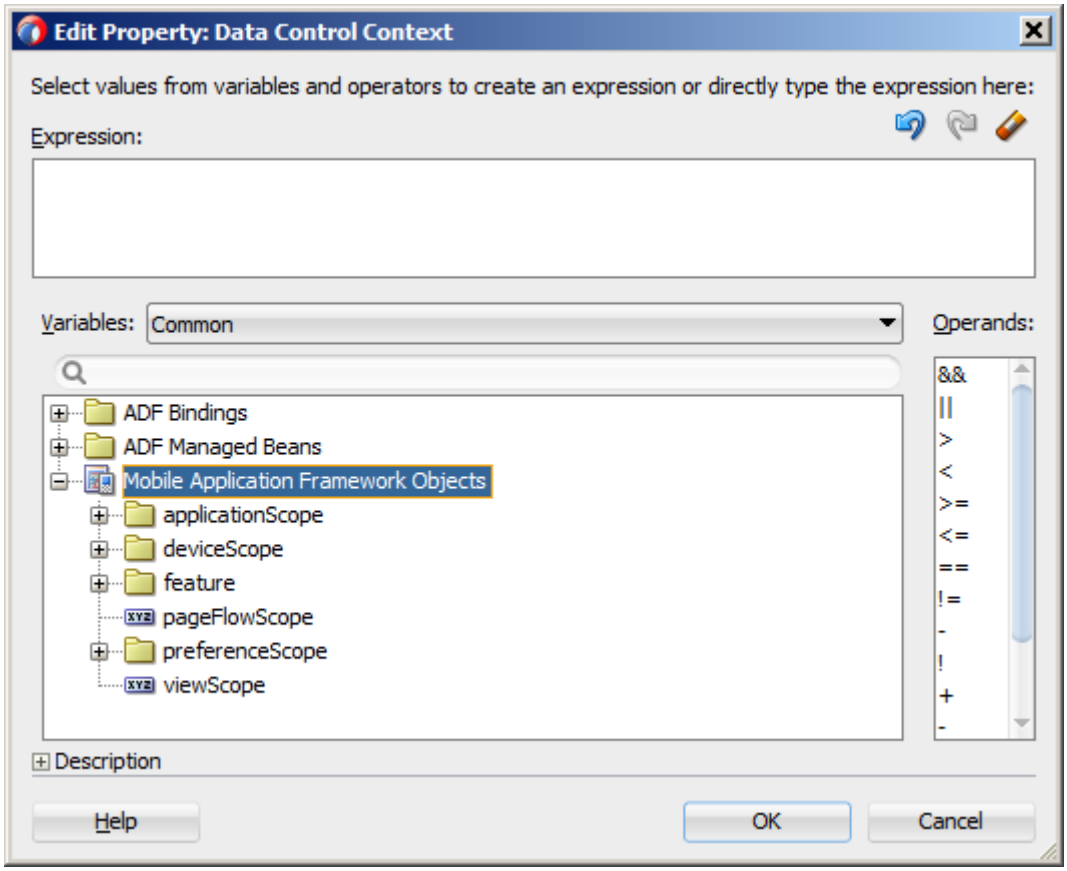
The internal object that task flows use to share their data controls or to store their own isolated data control is known as a data control context. When a task flow specifies a `data-control-context` value of `shared`, the called task flow uses the data control context of the calling task flow rather than creating its own. This allows the called task flow to share data control instances attached to the data control context. Alternatively, if a called task flow specifies a `data-control-context` value of `isolated`, a new data control context is created and a new instance of any data controls used by the task flow is attached to the newly-created data control context.

You can define a data control context for a task flow call activity as follows:

1. In the **Diagram** view, select a **Task Flow Call** activity.
2. In the **Properties** window, expand the **General** section, and then provide a value for the **Data Control Context** property:



- Select `shared` (default) from the drop-down field if the data control is to be shared with the calling task flow. For example, changes made to a called task flow are reflected when navigation back to the calling task flow occurs.
- Select `isolated` from the drop-down field if the task flow is to have its own set of data control instances.
- Define an EL expression that evaluates to either `"shared"` or `"isolated"`. You set this value in the **Edit Property: Data Control Context** dialog (see the following illustration) that you can open by clicking the Property Menu icon located to the right of the Data Control Context field in the Properties window.



Note: To select the correct value for the `data-control-context`, consider looking at your data model and decide how data should be shared across task flow boundaries.

For information on setting depth of the data control context, see How to Define the Data Control Context Depth for Task Flows.

13.2.5.5 Adding Task Flow Return Activities

A task flow return activity identifies the point in a 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 Components window, select **Components > Activities**.
3. Drag and drop a Task Flow Return activity onto the diagram.
4. In the Properties window (see Figure 13-12), expand the General section and enter an outcome in the **Name** field.

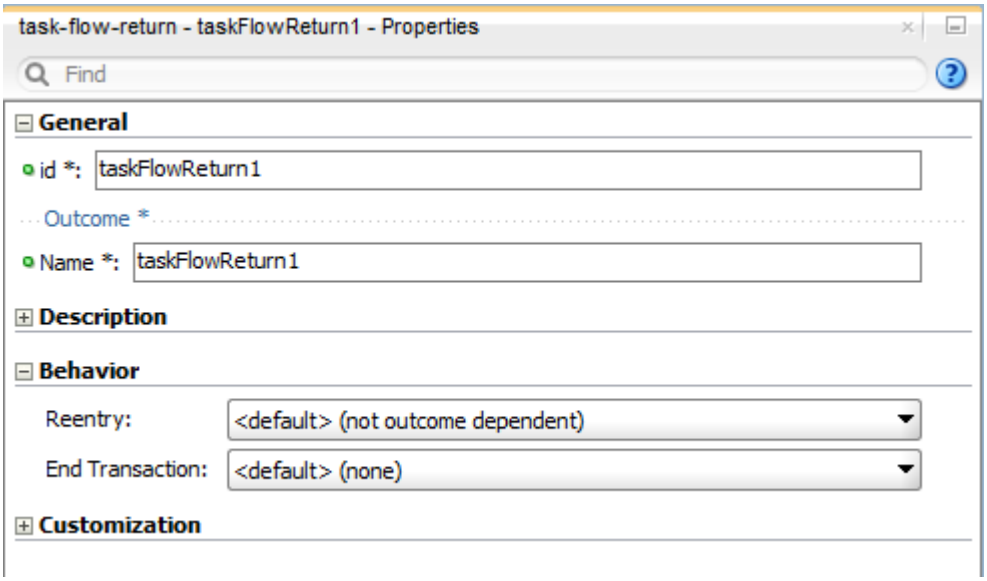
   The task flow return activity returns this outcome to the calling task flow when the current bounded task flow exits. You can specify only one outcome per task flow return activity. The calling task flow should define control flow rules to handle control flow upon return. For more information, see How to Define Control Flows.
5. 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.
6. From the End Transaction dropdown 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.

Figure 13-12 Configuring Task Flow Return Activity



13.2.5.6 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 a 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, JDeveloper 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, JDeveloper displays a context menu option for all task flow activities to go to the page definition file (see Accessing the Page Definition File). JDeveloper 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.

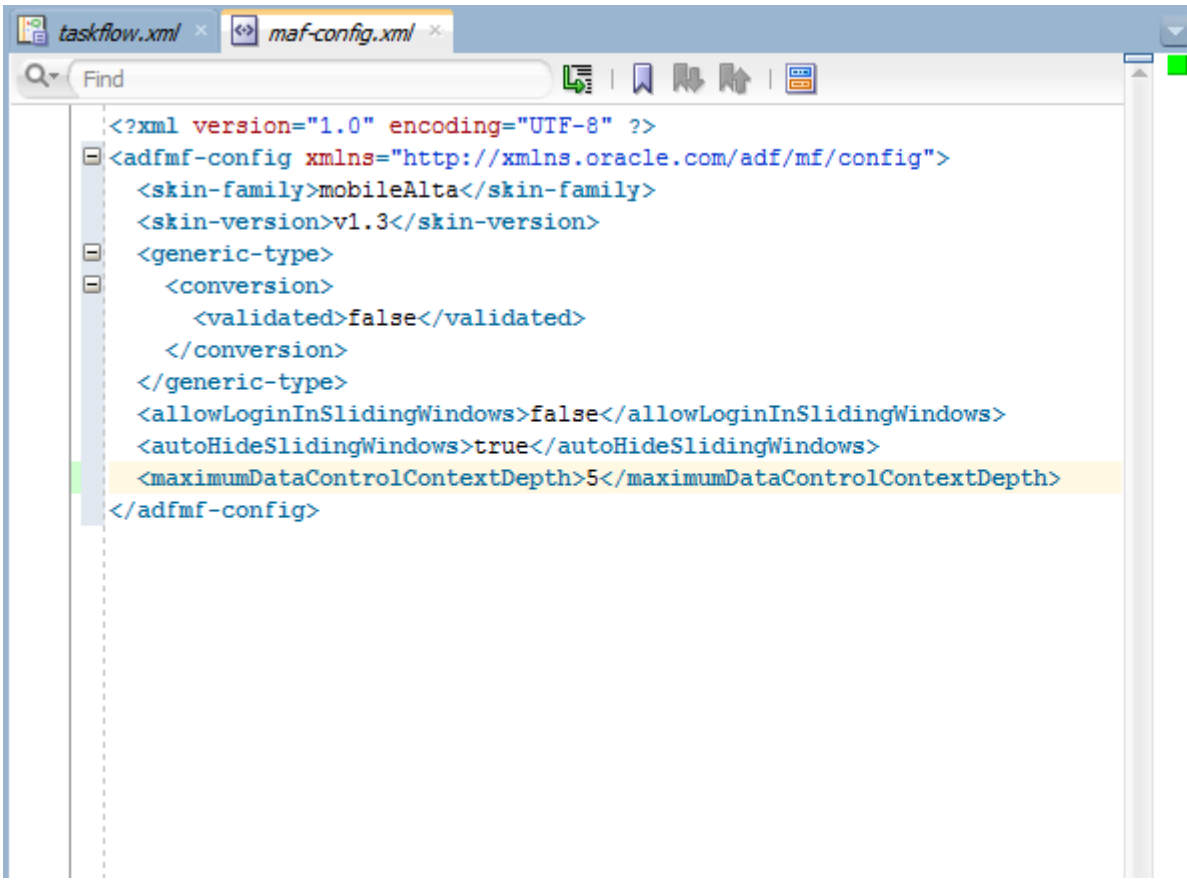
For more information about page definition files, see What You May Need to Know About Generated Drag and Drop Artifacts.

When the preceding steps are completed, JDeveloper 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. JDeveloper 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.

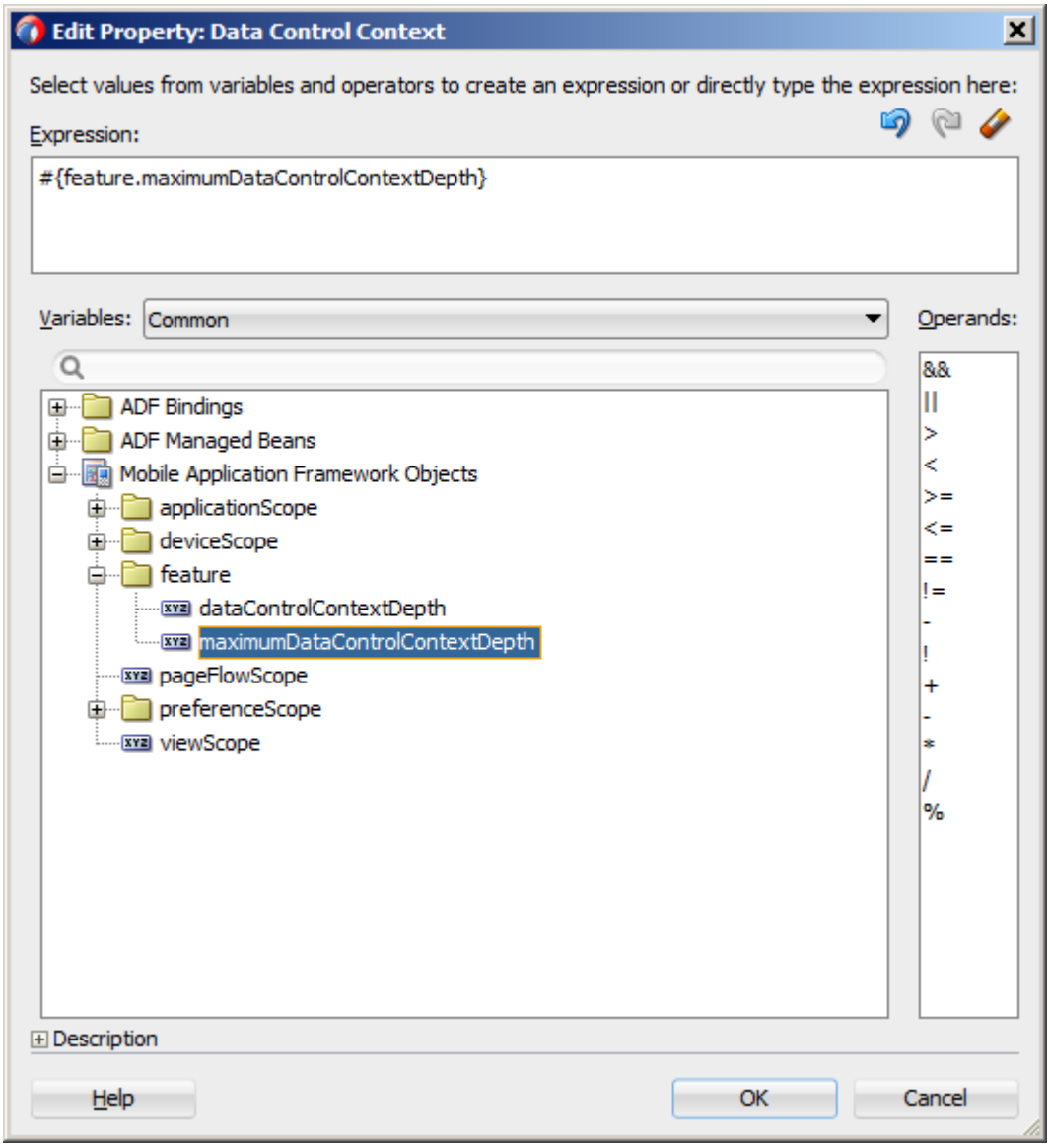
13.2.6 How to Define the Data Control Context Depth for Task Flows

Configuring the data control context depth allows you to avoid an infinite drill into a task flow by disabling certain functionality. For example, you can disable drilling into nested task flow when a particular data control context stack depth is reached.

You can modify the maximum depth of the data control context (see Specifying the Data Control Context) by setting the `maximumDataControlContextDepth` property in your application's `maf-config.xml` file, as the following illustration shows. The default value is `10`.



You can also define an EL expression of either `#{feature.dataControlContextDepth}` or `#{feature.maximumDataControlContextDepth}` using the Mobile Application Framework `feature` object, as the following illustration shows.



In addition, you can use methods of the `EmbeddedFeatureContext` class that provide access to the current data control context stack depth as well as the maximum stack depth. For more information, see *Java API Reference for Oracle Mobile Application Framework*.

For more information about the `maf-config.xml` file, see Table C-1.

For more information about EL expressions, see Creating EL Expressions.

13.2.7 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 Defining a Control Flow Case)
- Wildcard Control Flow Rule (see Adding a Wildcard Control Flow Rule)

13.2.7.1 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 **Components** window. Next, connect the **Control Flow Case** to the source activity, and then to the destination activity.

JDeveloper 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 **Components** window.
3. On the diagram, click a source activity and then a destination activity. JDeveloper adds the control flow case to the diagram. Each line that JDeveloper 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.

13.2.7.2 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 **Components** window. 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 **Components** window and drop it onto the diagram.
3. Select **Control Flow Case** in the **Components** window.
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 **Overview** editor for 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**.

13.2.7.3 What You May Need to Know About Control Flow Rule Metadata

The following example shows the general syntax of a control flow rule element in the task flow source file.

```
<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 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 the preceding example, 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.

13.2.7.4 What You May Need to Know About Control Flow Rule Evaluation

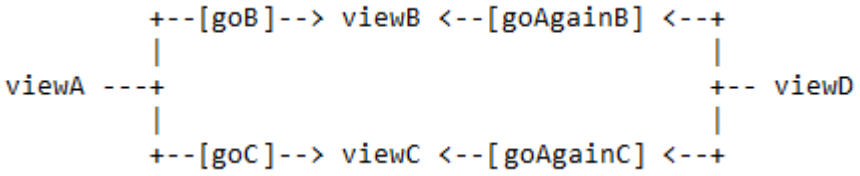
At runtime, task flows evaluate control flow rules from the most specific to the least specific match to determine the next transition between activities. Evaluation is based on the following priority:

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.

13.2.8 What You May Need to Know About MAF Support for Back Navigation

In the task flow example that Figure 13-13 shows, it is possible to take two separate paths to reach **viewD** based on the action outcome value (see How to Specify Action Outcomes Using UI Components): either from **viewA** to **viewB** to **viewD**, or from **viewA** to **viewC** to **viewD**.

Figure 13-13 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:

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

- How to Create and Reference Managed Beans
- Enabling the Back Button Navigation

13.2.9 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 How to Enable Drag Navigation.

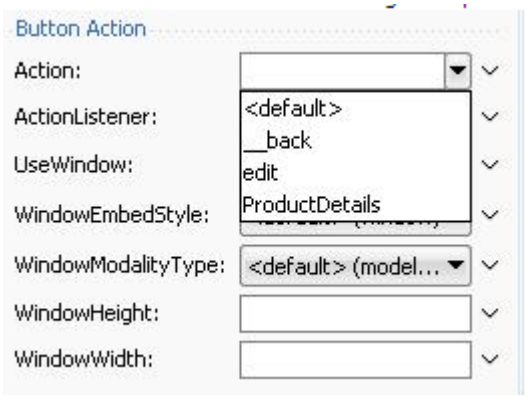
13.2.10 How to Specify Action Outcomes Using UI Components

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 How to Use Buttons)
- Command Link (see How to Use Links)
- List Item

You use the UI component's **Action** field (see Figure 13-14) to make a selection from a list of possible action outcomes defined in one or more task flow for a specific MAF AMX page.

Figure 13-14 Setting Actions



A **Back** action (`__back`) is automatically added to every list to enable navigation to the previously visited page.

Note:

A 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.

13.2.11 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 JDeveloper's usual editing mechanism to reference managed beans and create references to them for applicable fields. For more information, see Creating and Using Managed Beans.

Figure 13-15 shows the **Edit** option for an `action` property in the **Properties** window. You click this option to invoke the **Edit Property** dialog that Figure 13-16 shows.

Figure 13-15 Edit Dialog

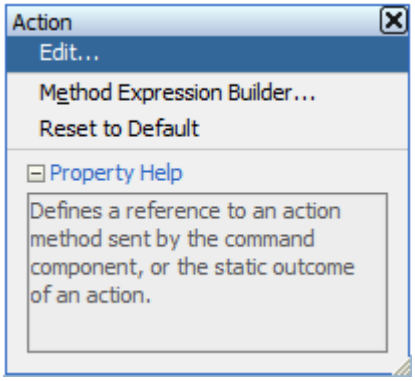


Figure 13-16 Edit Property Dialog for Action

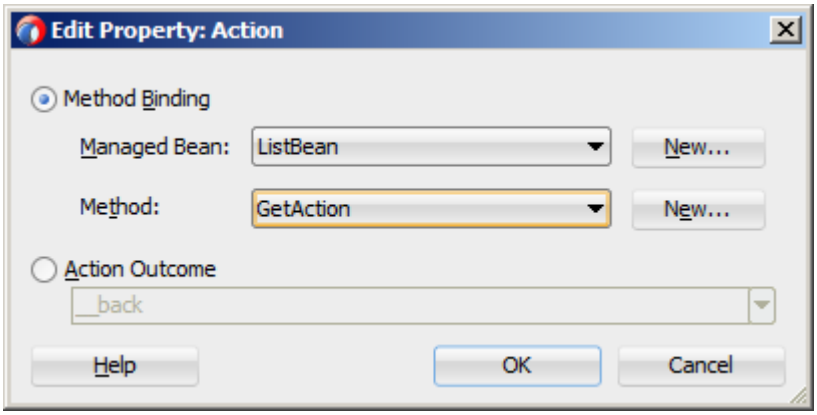


Table 13-7 lists MAF AMX attributes for which the **Edit** option in the **Properties** window is available.

Table 13-3 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`

Table 13-3 (Cont.) Editable Attributes

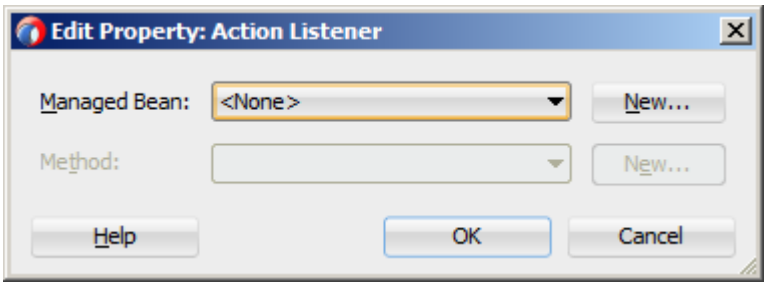
Property	Element
actionListener	amx:listItem
actionListener	amx:commandButton
actionListener	amx:commandLink
binding	amx:actionListener
mapBoundsChangeListener	dvtm:geographicMap
mapInputListener	dvtm:geographicMap
moveListener	amx:listView
rangeChangeListener	amx:listView
selectionListener	amx:listView
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:lineChart
selectionListener	dvtm:funnelChart
selectionListener	dvtm:pieChart
selectionListener	dvtm:scatterChart
valueChangeListener	amx:inputDate
valueChangeListener	amx:inputNumberSlider
valueChangeListener	amx:inputText
valueChangeListener	amx:selectBooleanCheckbox

Table 13-3 (Cont.) Editable Attributes

Property	Element
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:lineChart

Clicking **Edit** for all other properties invokes a similar dialog, but without the **Action Outcome** option, as Figure 13-17 shows.

Figure 13-17 Edit Property Dialog for Action Listener



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 13-16 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 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`:

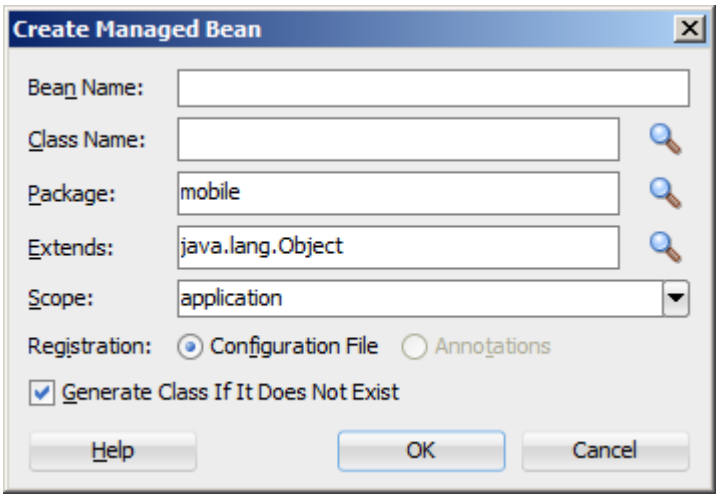
```
<amx:commandButton action="goHome"/>
```

The **Method Binding** option (see Figure 13-16) 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 13-18 shows.

Figure 13-18 Create Managed Bean Dialog



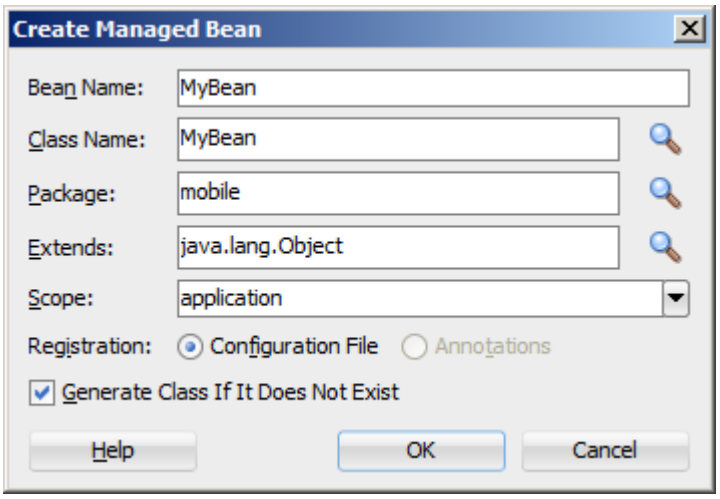
MAF supports the following scopes:

- `application`
- `view`
- `pageFlow`

When you declare a managed bean to a 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 Creating EL Expressions.

2. Provide the managed bean and class names (see Figure 13-19), and then click **OK**.

Figure 13-19 Setting Managed Bean Name and Class



The following example shows the newly created managed bean class. The task flow that this MAF AMX page is part of is updated to reference the bean.

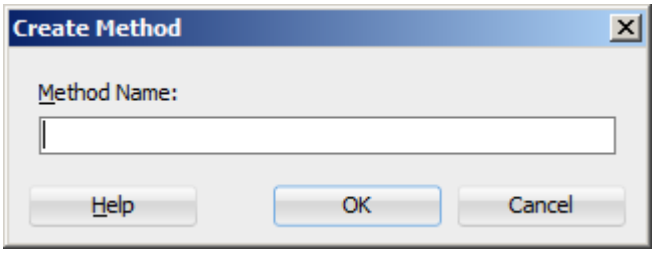
```
<managed-bean id="__3">
   <managed-bean-name>MyBean</managed-bean-name>
   <managed-bean-class>mobile.MyBean</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

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.

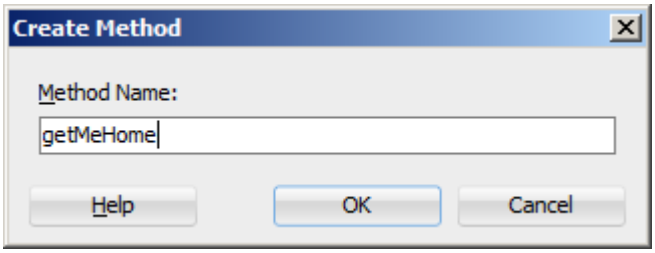
3. Click **New** next to the **Method** field (see Figure 13-16 and Figure 13-17) to open the **Create Method** dialog that Figure 13-20 shows.

Figure 13-20 Create Method Dialog



Use this dialog to provide the managed bean method name (see Figure 13-21).

Figure 13-21 Naming Managed Bean Method



Upon completion, the selected property value is updated with the appropriate EL expression, such as the following for a `commandButton`:

```
<amx:commandButton action="#{MyBean.getMeHome}"/>
```

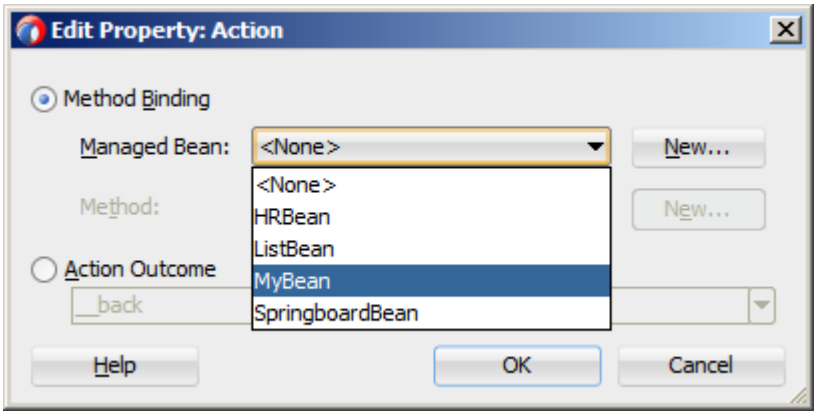
The managed bean class is also updated to contain the newly created method, as the following example shows.

```
package mobile;

public class MyBean {
   public MyBean() {
   }

   public String getMeHome() {
      // Add event code here...
      return null;
   }
}
```

To select an existing managed bean:

1. Make a selection from the **Managed Bean** list that Figure 13-22 shows.

Figure 13-22 Selecting Managed Bean



Similar to the action outcomes, the **Managed Bean** list is populated with managed beans from all task flows that this MAF AMX page is part of.

Note:

If the MAF AMX page is not part of any task flow, you can still create a managed bean.

For more information, see the following:

- About the Managed Beans Category
- APIDemo, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

13.2.12 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

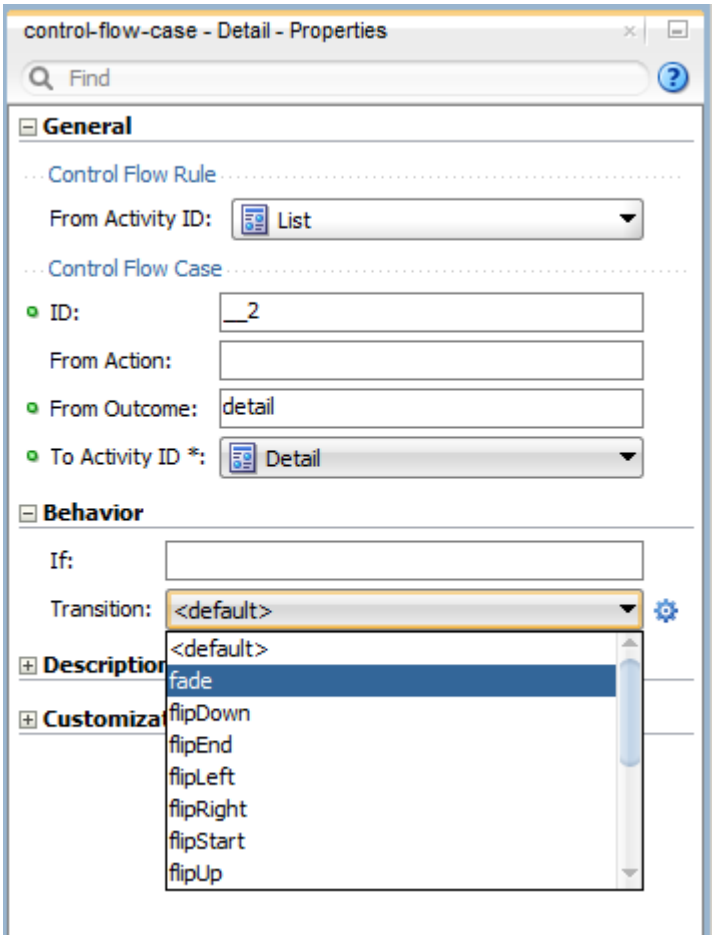
Sliding in from start and end, as well as flipping from start and end are used on the iOS platform and Android 4.2 or later platform to accommodate the right-to-left (RTL) text direction. It is generally recommended to use the start and end transition style as opposed to left and right.

You set the transition style by modifying the `transition` attribute of the `control-flow-case` (Control Flow Case component), as the following example shows.

```
<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**, as Figure 13-23 shows. The default transition style is `slideLeft`.

Figure 13-23 Setting Transition Style in Properties Window



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.

13.2.13 What You May Need to Know About Bounded and Unbounded Task Flows

Task flows provide a modular approach for defining control flow in a 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 a 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 What You May Need to Know About Task Flow Activities and Control Flows.

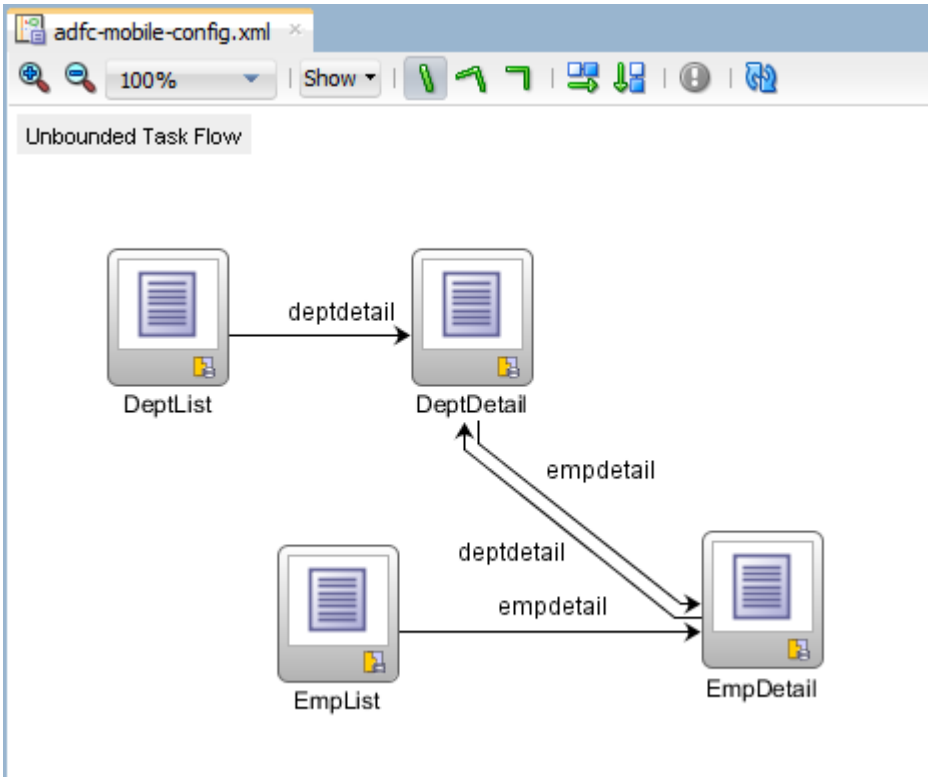
A typical MAF AMX application feature contains a combination of one unbounded task flow created at the time when the application feature is created and one or more bounded task flows. At runtime, the MAF application can call bounded task flows from activities that you added to the unbounded task flow.

13.2.13.1 Unbounded Task Flows

A 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.

Figure 13-24 displays the diagram for an unbounded task flow from a MAF AMX application feature. This task flow contains a number of view activities that are all entry points to the application feature.

Figure 13-24 Unbounded 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.

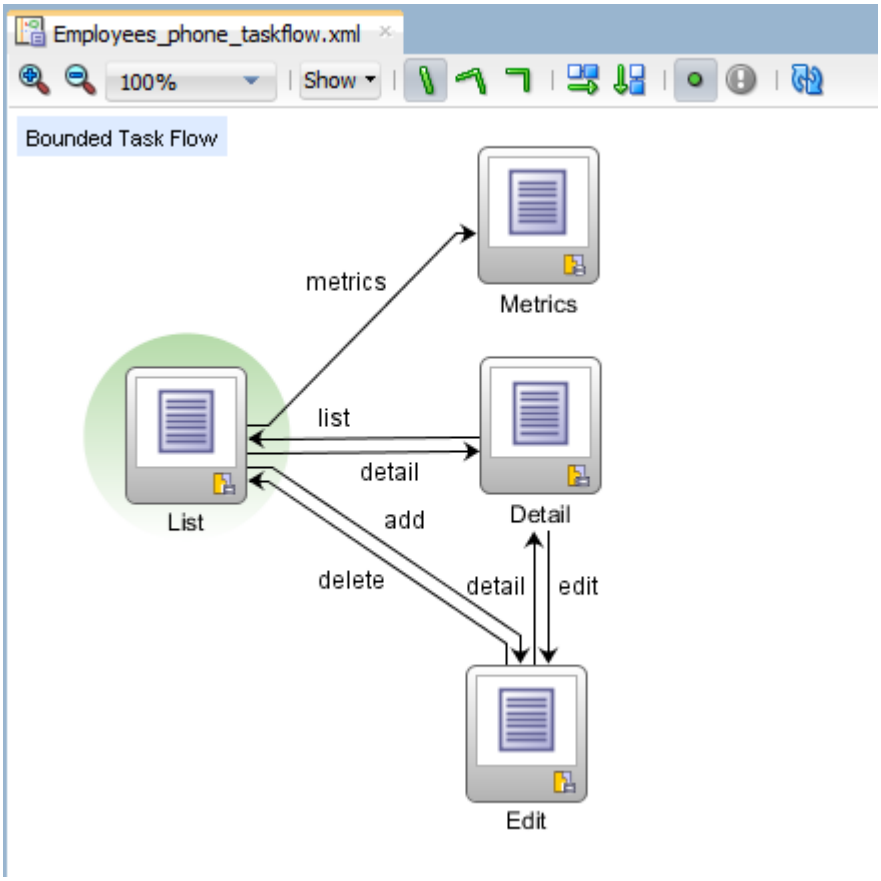
13.2.13.2 Bounded Task Flows

By default, the IDE proposes a file name for the source file of a bounded task flow (see 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 13-25 displays the diagram for a bounded task flow from a MAF AMX application feature.

Figure 13-25 Bounded 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 a 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 Passing Parameters to a Bounded Task Flow and Configuring a Return Value from a Bounded Task Flow).

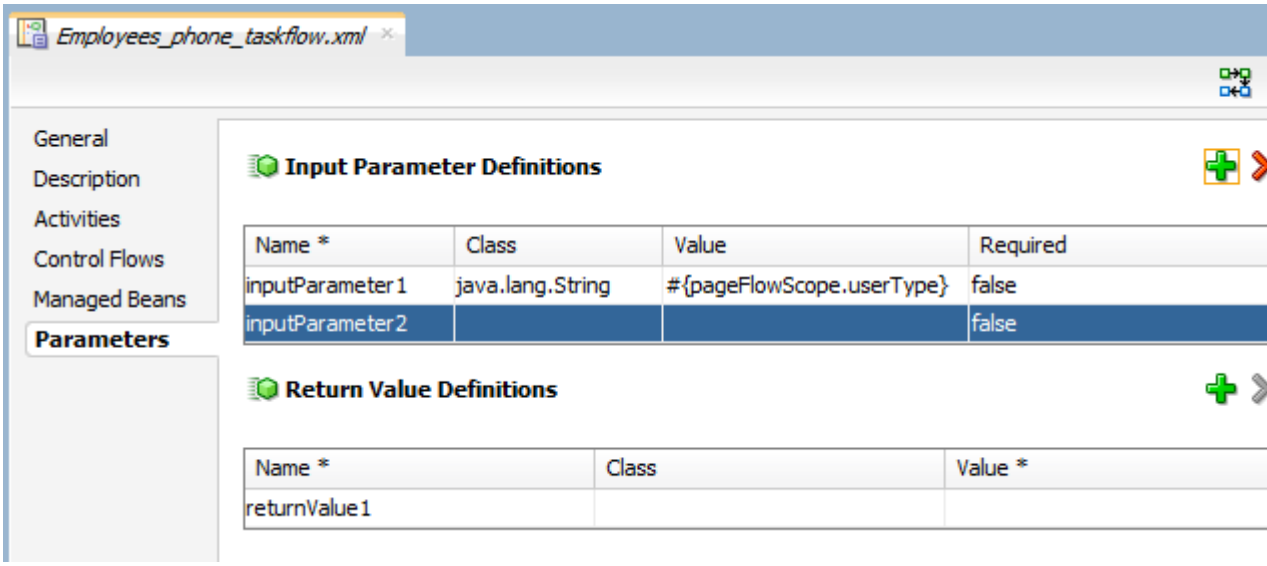
  In addition, you can share data controls between bounded task flows.
- On-demand loading of metadata: bounded task flow metadata is loaded on demand when entering a bounded task flow.

13.2.13.3 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 13-26 shows a task flow that specifies an input parameter definition to hold information about a user in a pageFlow scope.

Figure 13-26 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.

13.2.13.3.1 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 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 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 How to Create a Task Flow.
- Add a task flow call activity to the calling task flow.

Figure 13-27 shows an example where the view activity passes control to the task flow call activity.

Figure 13-27 Calling Task Flow

To pass an input parameter to a bounded task flow:

1. Open a 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:

   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 **Applications** window, 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 13-26):

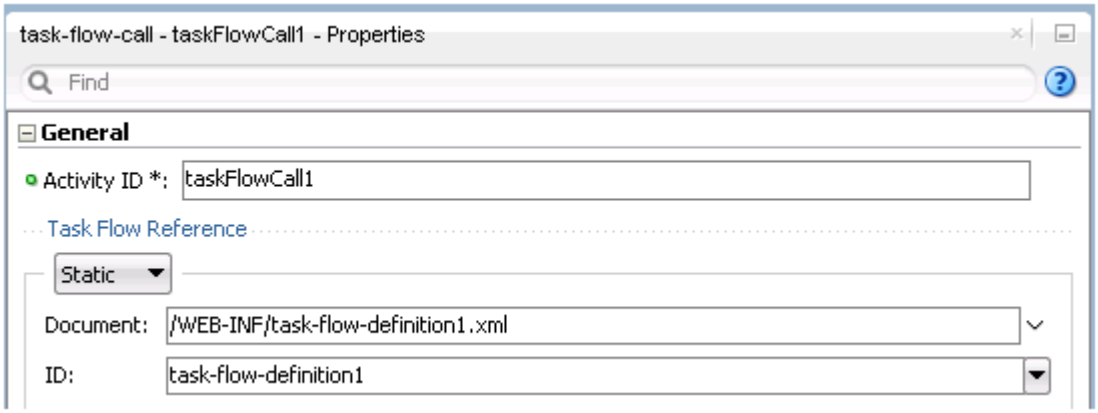
   - In the **Name** field, enter a name for the parameter (for example, `inputParm1`).
   - In the **Value** field, enter an EL expression where the parameter value is stored and referenced (for example, `#{pageFlowScope.inputValue}`), either manually or using the Expression Builder.

6. In the **Applications** window, double-click the calling task flow that contains the task flow call activity to invoke the called bounded task flow.

7. In the **Applications** window, 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 13-28, 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 13-28 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, JDeveloper writes entries to the source files for the calling task flow and called task flow based on the values that you select.

The following example shows an input parameter definition specified on a bounded task flow.

```
<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>
```

The following example shows the input parameter metadata for the task flow call activity that calls the bounded task flow shown in the preceding example.

```
<task-flow-call id="taskFlowCall1">
...
   <input-parameter>
      <name>inputParameter1</name>
      <value>#{pageFlowScope.newCustomer}</value>
      <pass-by-value/>
   </input-parameter>
...
</task-flow-call>
```

At runtime, the task flow call activity calls the bounded task flow and passes it the value specified by its value element.

13.2.13.3.2 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 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 **Applications** window, 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 13-26):
   - 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 **Applications** window, double-click the calling task flow.
4. With the task flow page open in the **Diagram** view, select **Components > Activities** from the **Components** window, 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}`).

Upon completion, JDeveloper writes entries to the source files for the calling task flows that you configured.

The following example shows a sample entry that JDeveloper writes to the source file for the calling task flow.

```
<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>
```

The following example shows a sample entry that JDeveloper writes to the source file for the called task flow.

```
<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.

13.3 Creating Views

You can create a view by adding data controls, dragging and dropping components onto a MAF AMX page.

You can start creating a MAF AMX view by doing the following:

- Getting familiar with the MAF AMX page structure (see Interpreting the MAF AMX Page Structure)
- Editing and previewing a MAF AMX page (see Using UI Editors)

- Dragging and dropping components onto a MAF AMX page (see How to Add UI Components to a MAF AMX Page)
- Adding data controls to a view (see How to Add Data Controls to a MAF AMX Page)

13.3.1 How to Work with MAF AMX Pages

A MAF AMX page is represented by an XML file whose structure consists of layered elements defining the page's presentation and functionality.

13.3.1.1 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 Providing Data Visualization), UI elements are declared under the `<amx>` namespace.

For more information, see What Happens When You Create a MAF AMX Page.

13.3.1.2 Creating MAF AMX Pages

MAF AMX files are contained in the **View Controller** project of the MAF application. You create these files using the **Create MAF AMX Page** dialog.

MAF offers two alternative ways of creating a MAF AMX page:

- From the New Gallery
- From an existing task flow

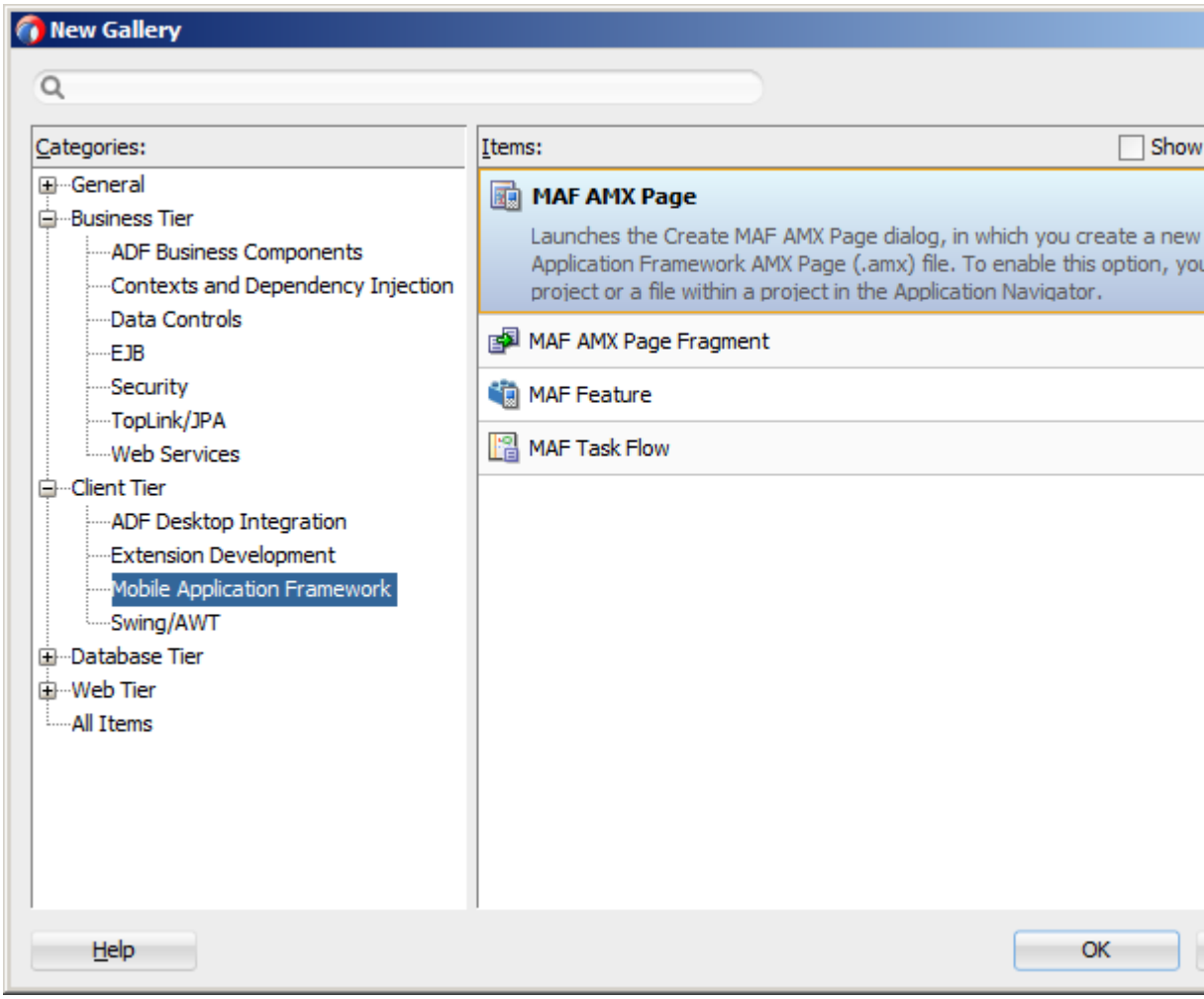
Before you begin:

To create a MAF AMX page, the MAF application must include a View Controller project file (see Getting Started with MAF Application Development).

To create a MAF AMX page from the **New Gallery**:

1. From the top-level menu in JDeveloper, click **File**, and then select **New > From Gallery**.
2. In the **New Gallery**, expand the **Client Tier** node, select **Mobile Application Framework,** and then **MAF AMX Page** (see Figure 13-29). Click **OK**.

Figure 13-29 Creating MAF AMX Page



3. In the **Create MAF AMX Page** dialog, enter a name and, if needed, a location for your new file, as Figure 13-30 shows.

4. Optionally, you may select which facets your new MAF AMX page will include as a part of the page layout:

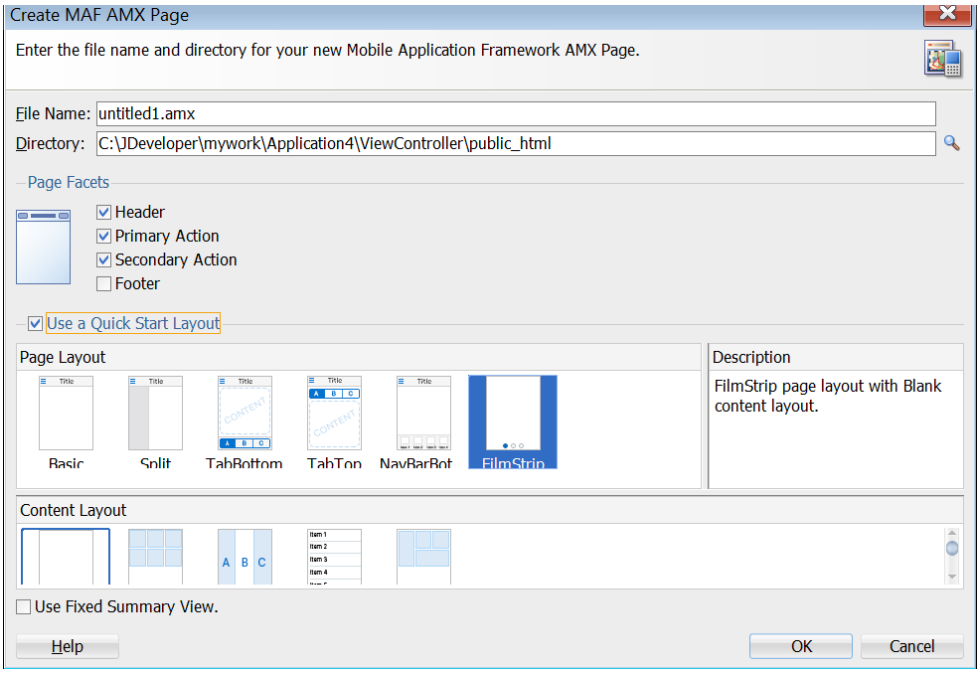
   - Header
   - Primary
   - Secondary
   - Footer

   For more information, see What Happens When You Create a MAF AMX Page and How to Use a Facet Component.

Note:

When you select or deselect a facet, the image representing the page changes dynamically to reflect the changing appearance of the page.

Figure 13-30 Create MAF AMX Page Dialog



Note:

MAF persists your facet selection and applies it to each subsequent invocation of the **Create MAF AMX Page** dialog.

5. Select to choose a page layout for your new page. The **Quick Start Layout** use the predefined components for a quick and easy way to correctly build the layout. Choose the type of page layout that you want to use (Basic, Split, and so on). The list of options in the **Content Layout** is filtered to display an appropriate option based on **Page Layout** selection.

6. Click **OK** on the **Create MAF AMX Page** dialog.

To create a MAF AMX page from a **View** component of the task flow:

1. Open a task flow file in the diagrammer (see Figure 13-7, How to Create a Task Flow and What You May Need to Know About the MAF Task Flow Diagrammer)

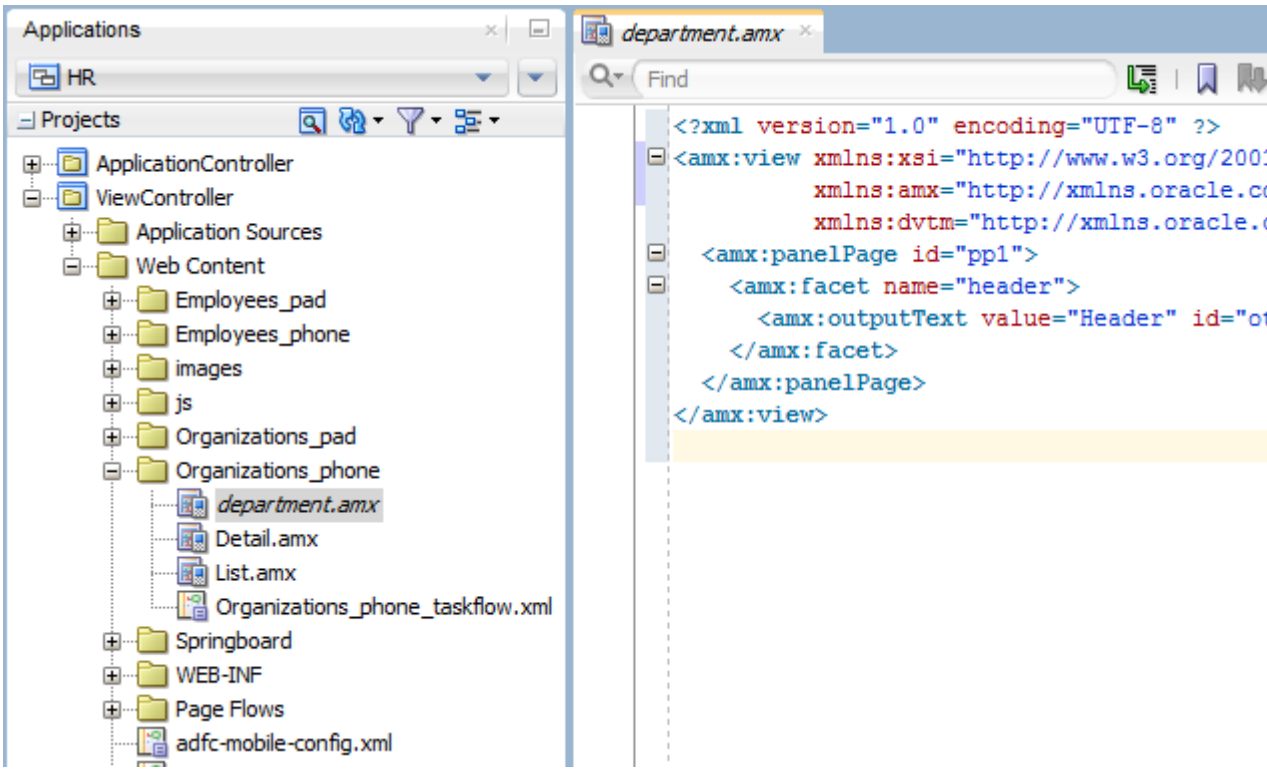
2. Double-click a **View** component of the task flow to open the **Create MAF AMX Page** dialog, and then enter a name and, if needed, a location for your new file. Click **OK**.

13.3.1.3 What Happens When You Create a MAF AMX Page

When you use the **Create MAF AMX Page** dialog to create a MAF AMX page, JDeveloper creates the physical file and adds it to the `Web Content` directory of the View Controller project.

In the **Applications** window that Figure 13-31 shows, the `Web Content` node contains a newly created MAF AMX file called `department.amx`.

Figure 13-31 MAF AMX File in Applications Window



JDeveloper 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 13-31.

Figure 13-33 shows how the **Preview** pane and the generated MAF AMX code would look like if you selected all facet types listed in the **Page Facet** section of the **Create MAF AMX Page** dialog when creating the page (see Figure 13-32).

Figure 13-32 Creating MAF AMX Page with All Facets

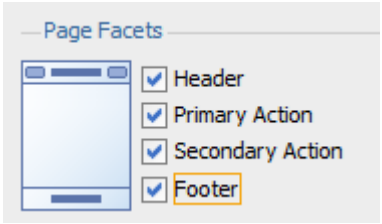
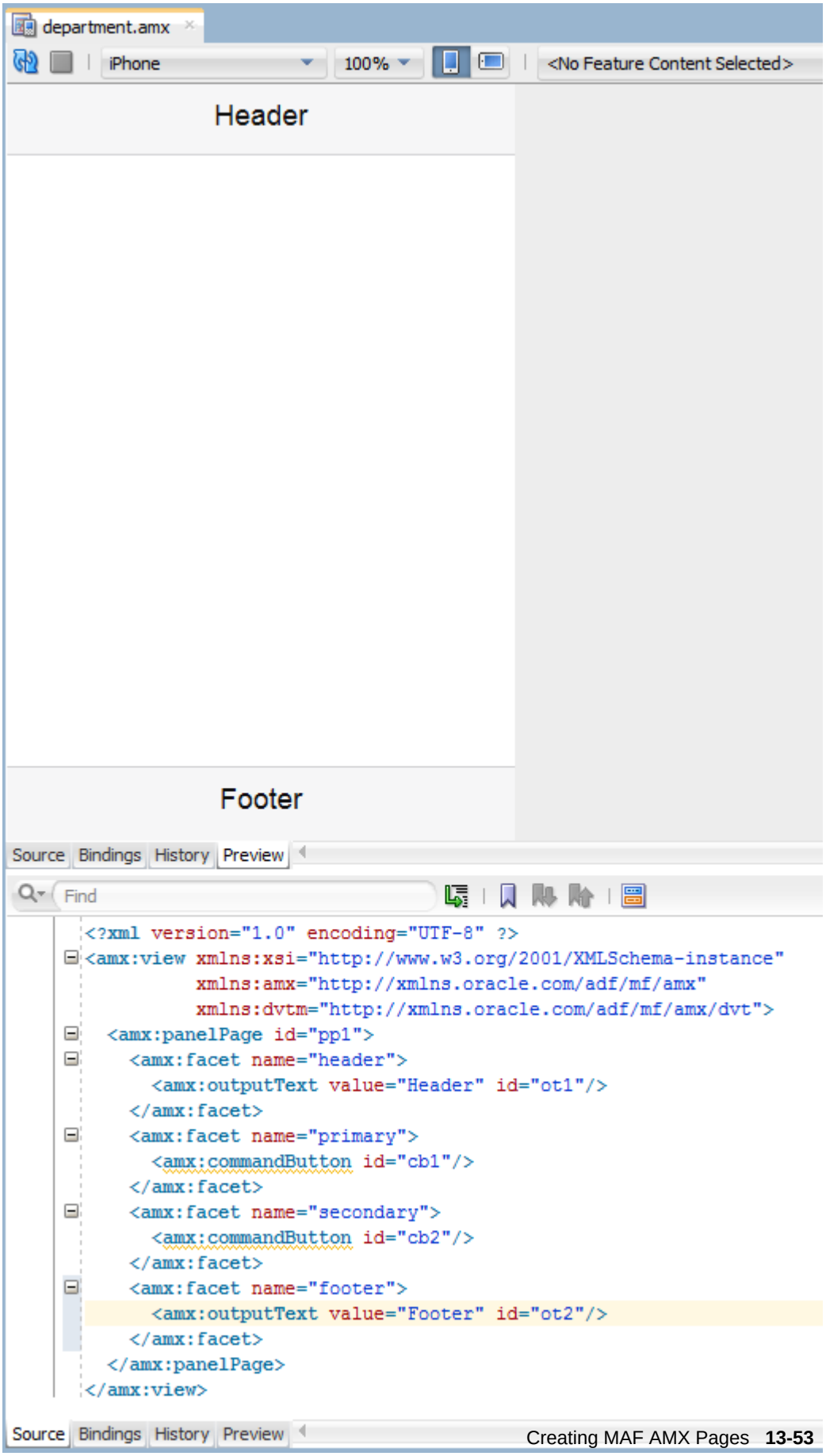


Figure 13-33 MAF AMX Page With All Facets



In the page created with all the facets selected (see Figure 13-32 and Figure 13-33), 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 JDeveloper 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 13-34 shows the **Page Facet** section of the **Create MAF AMX Page** dialog without any facets selected and Figure 13-35 shows the **Preview** pane with the generated MAF AMX code.

Figure 13-34 Creating MAF AMX Page Without Selected Facets

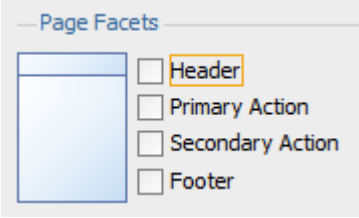
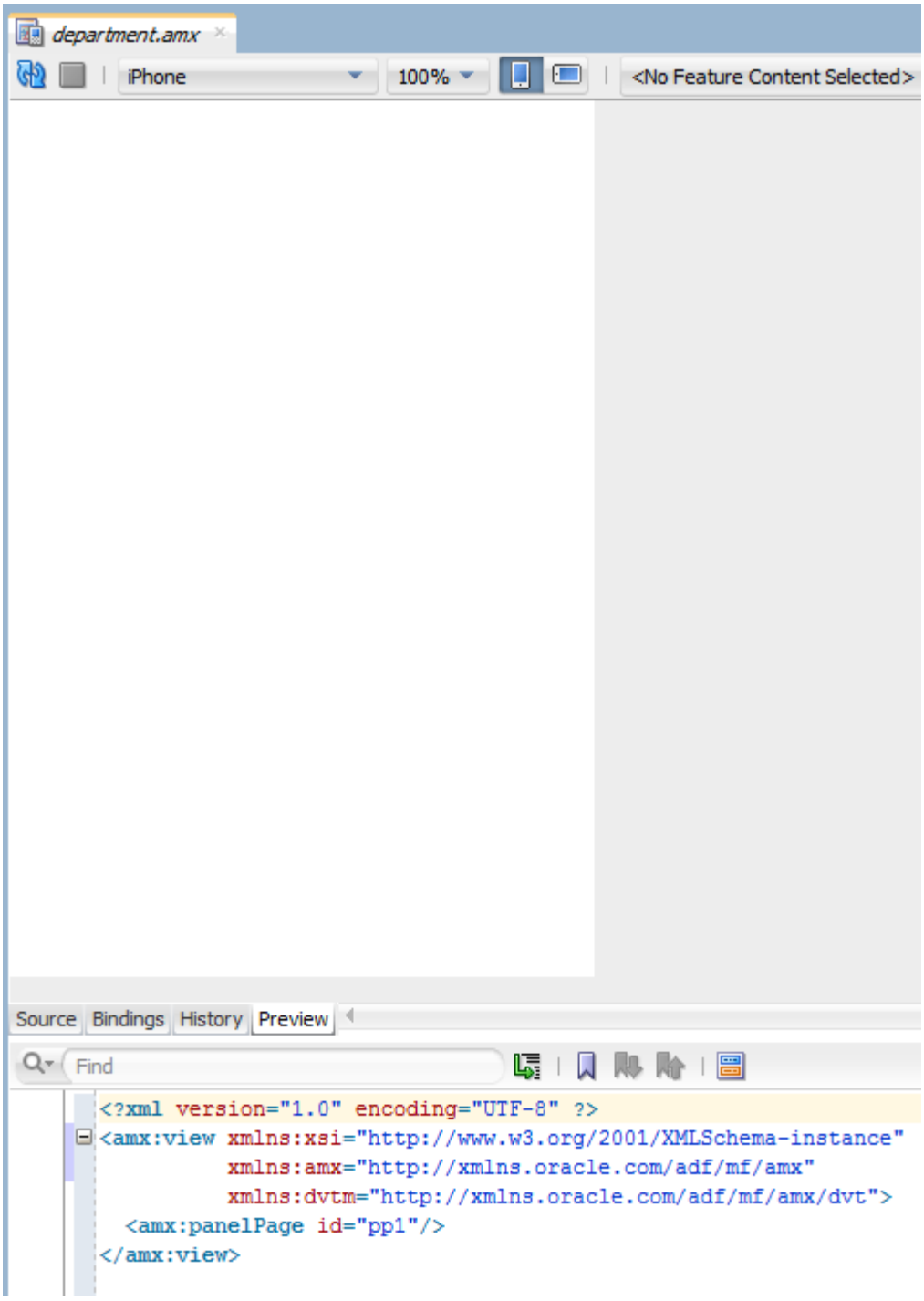


Figure 13-35 MAF AMX Page Without Facets

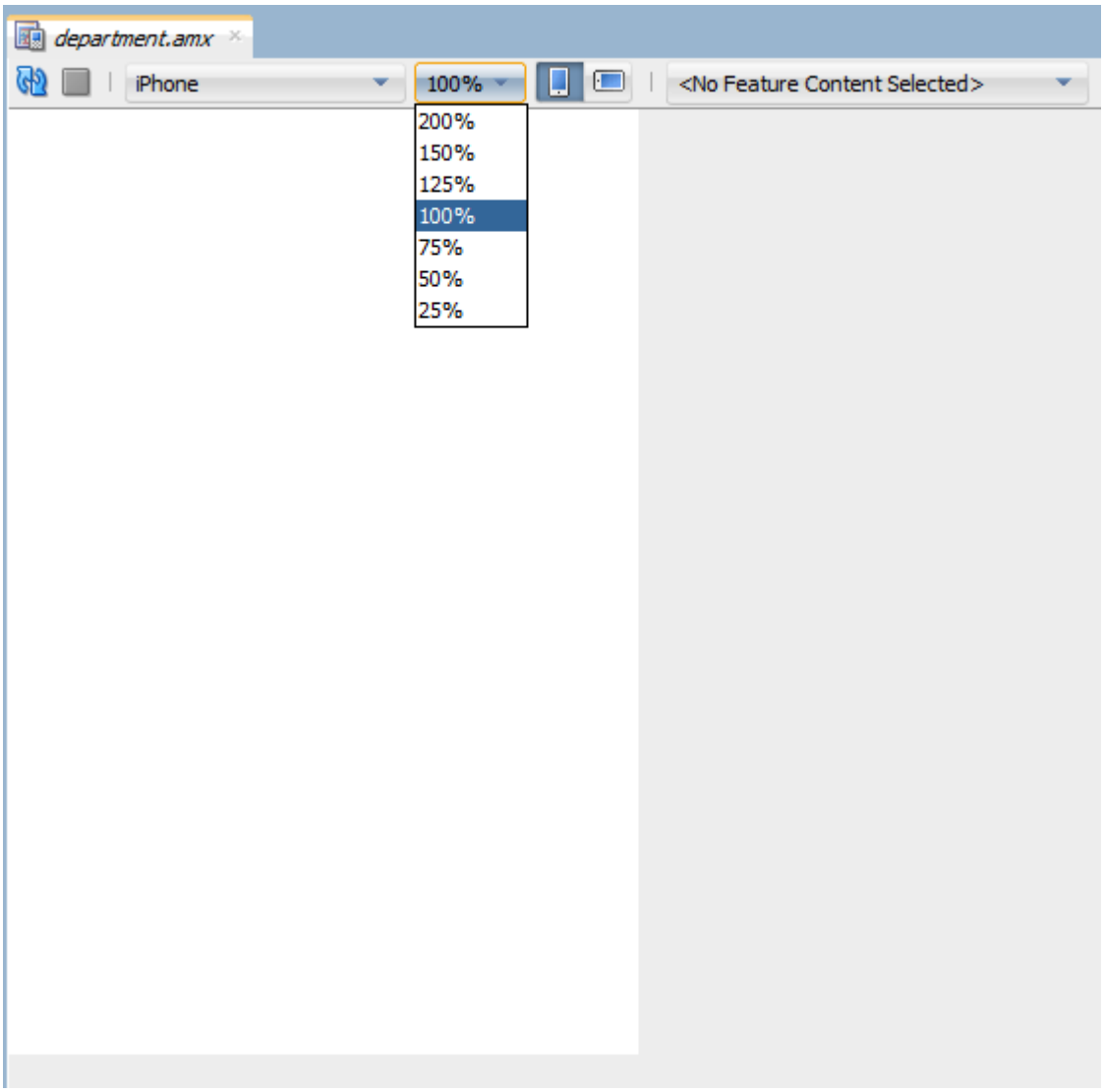


13.3.1.4 Using UI Editors

When the page is first displayed in JDeveloper, it is displayed in the **Source** editor. To view the page in a WYSIWYG environment, use the **Preview** pane (accessed by clicking the **Preview** tab).

Figure 13-36 shows the Preview tab selected for a newly created MAF AMX page called `department.amx`. This page is blank because it has not yet been populated with MAF AMX UI components or data controls.

Figure 13-36 The Preview Pane for Newly Created Page



Using the **Preview** pane's tool bar that Figure 13-36 shows, you can do the following:

- Refresh the display of the MAF AMX page by clicking **Refresh Page**.
- Stop loading of the page by clicking **Stop Loading Page**.
- Modify the form factor for the page by selecting a different form factor from the drop-down list. For more information on form factors, see the "Configuring the Development Environment for Form Factors" section in *Installing Oracle Mobile Application Framework*.
- Modify the scaling of the display by selecting a different percentage value from the drop-down list. Since mobile device displays can be of various sizes and densities, the **Preview** pane allows you to see the effect of scaling on your MAF AMX pages.

Note:

Scaling is available for both Portrait and Landscape mode.

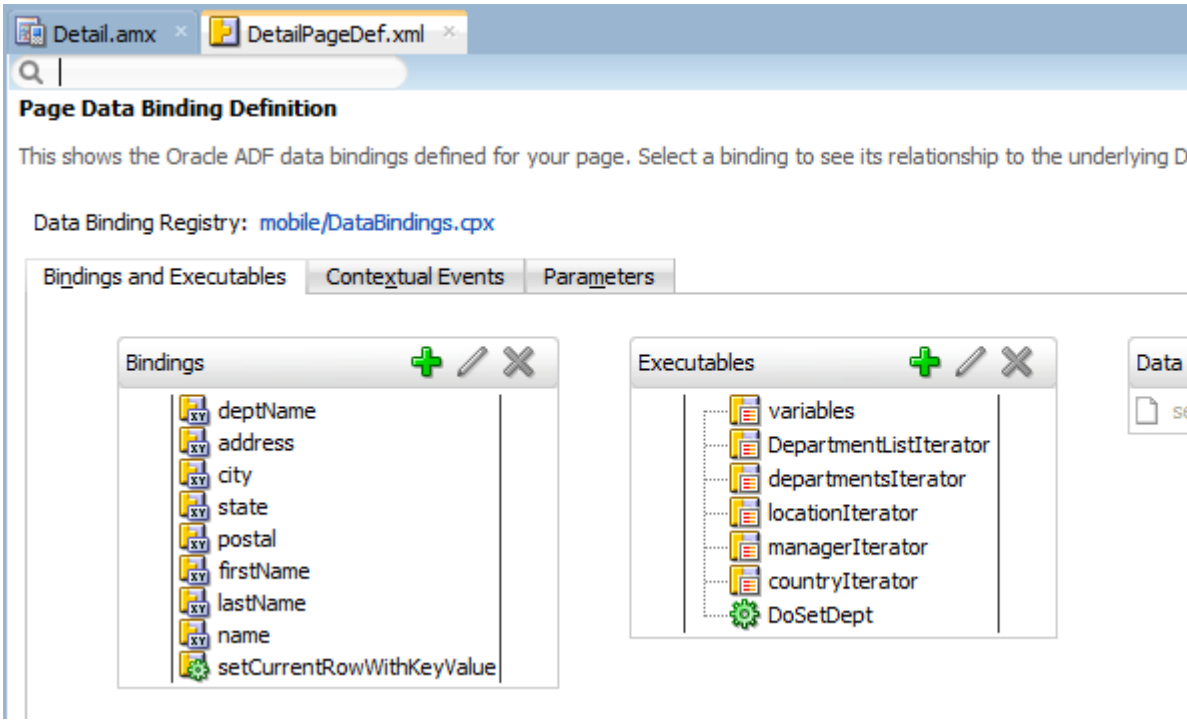
- Change orientation for the display to portrait and landscape by selecting **Show Portrait Orientation** or **Show Landscape Orientation** respectively.
- Select the feature content for your MAF AMX page form the dropdown list of available application features. By default, <No Feature Content Selected> is displayed.

To view the source for the page in the Source editor, click the **Source** tab that Figure 13-31 shows. The **Structure** window, located in the lower left-hand corner of JDeveloper (shown in Figure 13-31 and Figure 13-36), provides a hierarchical view of the page. For more information, see Using the Preview.

13.3.1.5 Accessing the Page Definition File

MAF AMX supports JDeveloper's **Go to Page Definition** functionality that enables you to navigate to the MAF AMX page definition (see Figure 13-37 and 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.

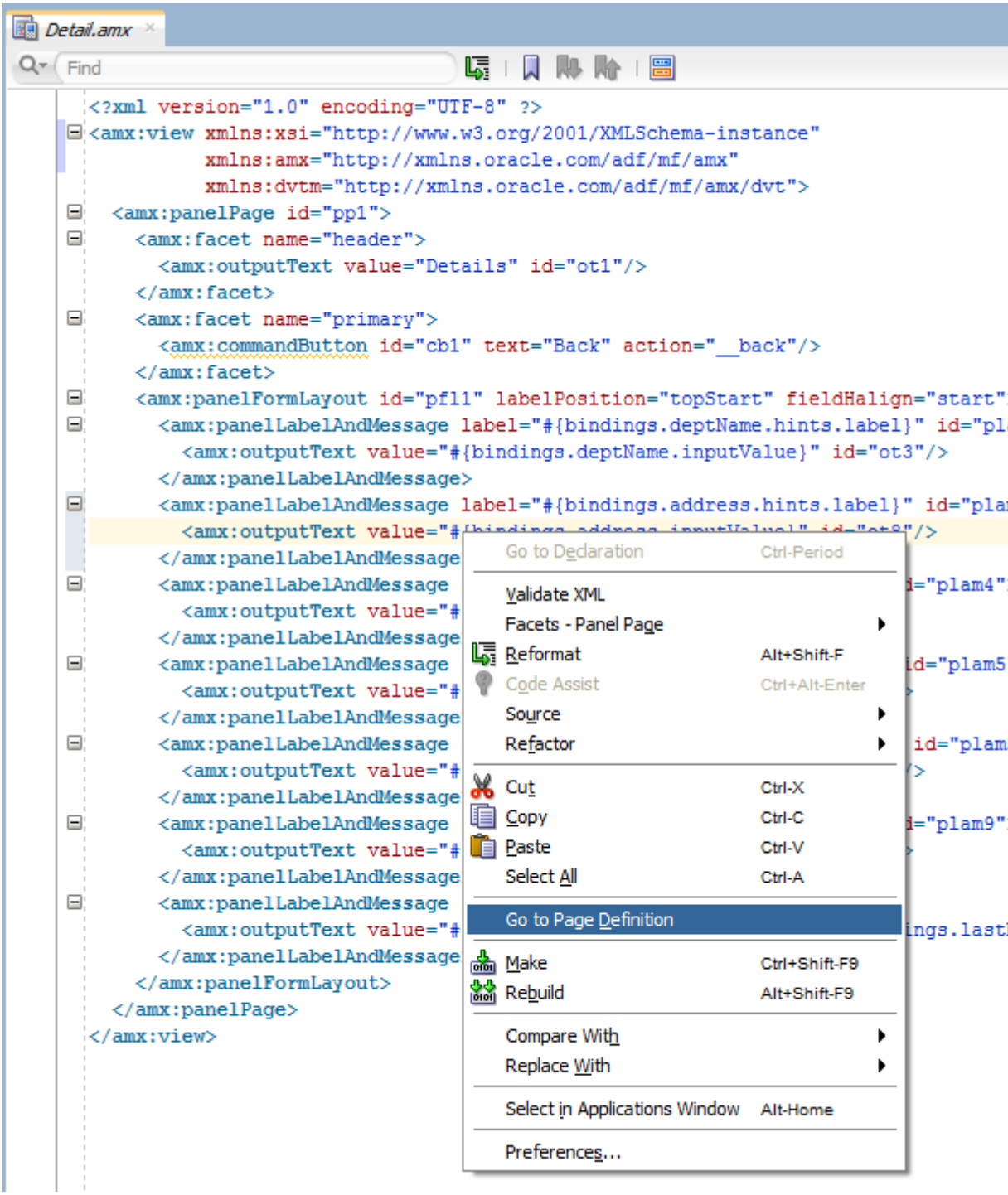
Figure 13-37 Page Definition File Accessed Through Go To Page Definition



You can invoke the context menu that contains the **Go to Page Definition** option from the following:

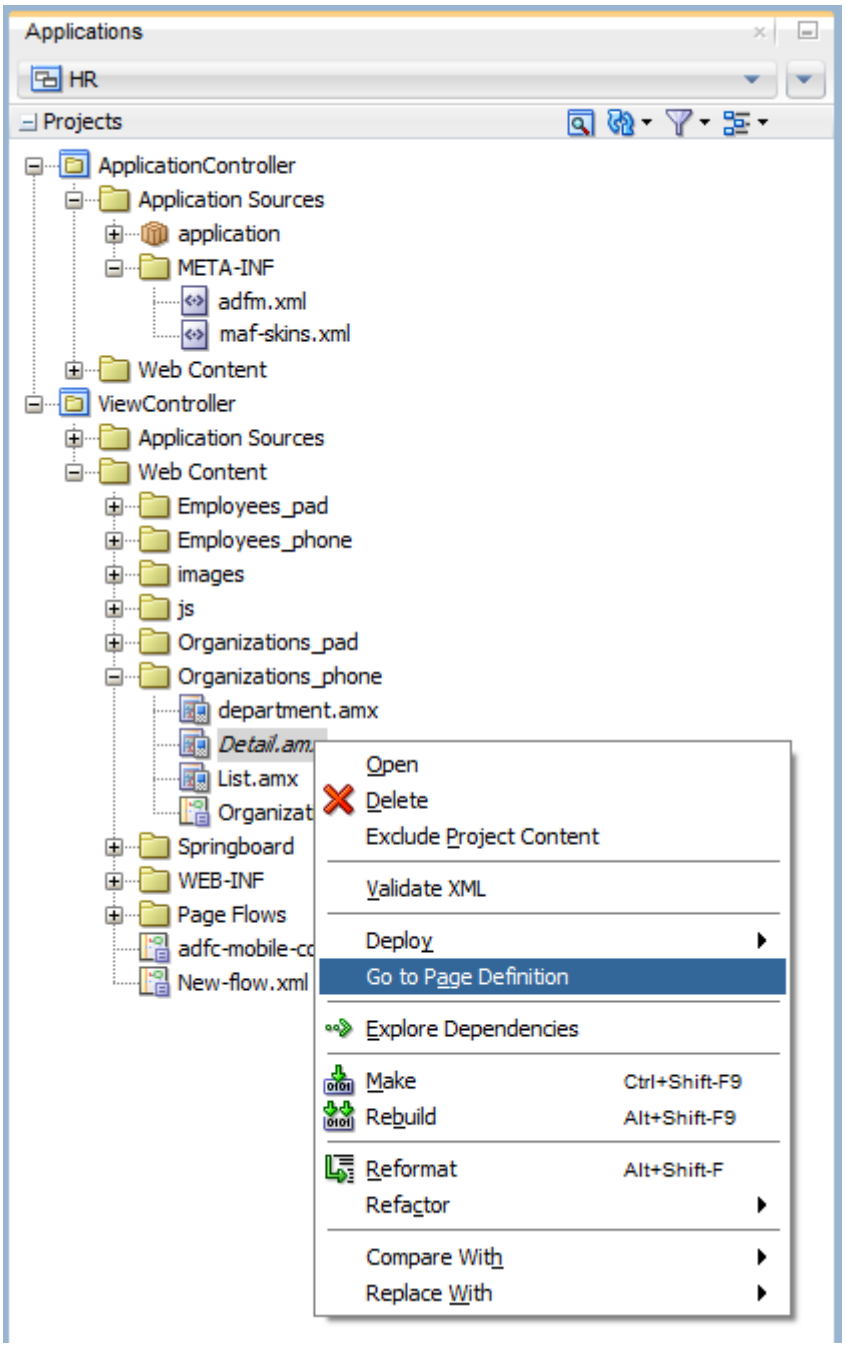
- **Source** editor, as Figure 13-38 shows.

Figure 13-38 Go to Page Definition from Source Editor



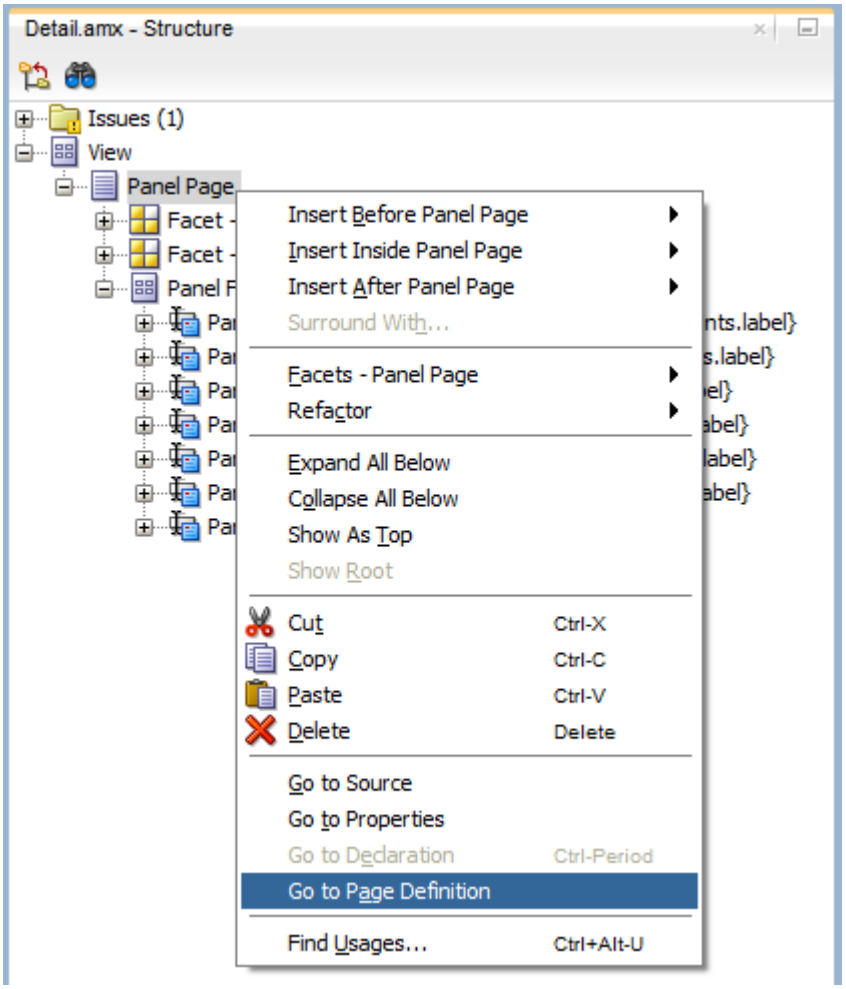
- **Applications** window, as Figure 13-39 shows.

Figure 13-39 Go to Page Definition from Applications Window



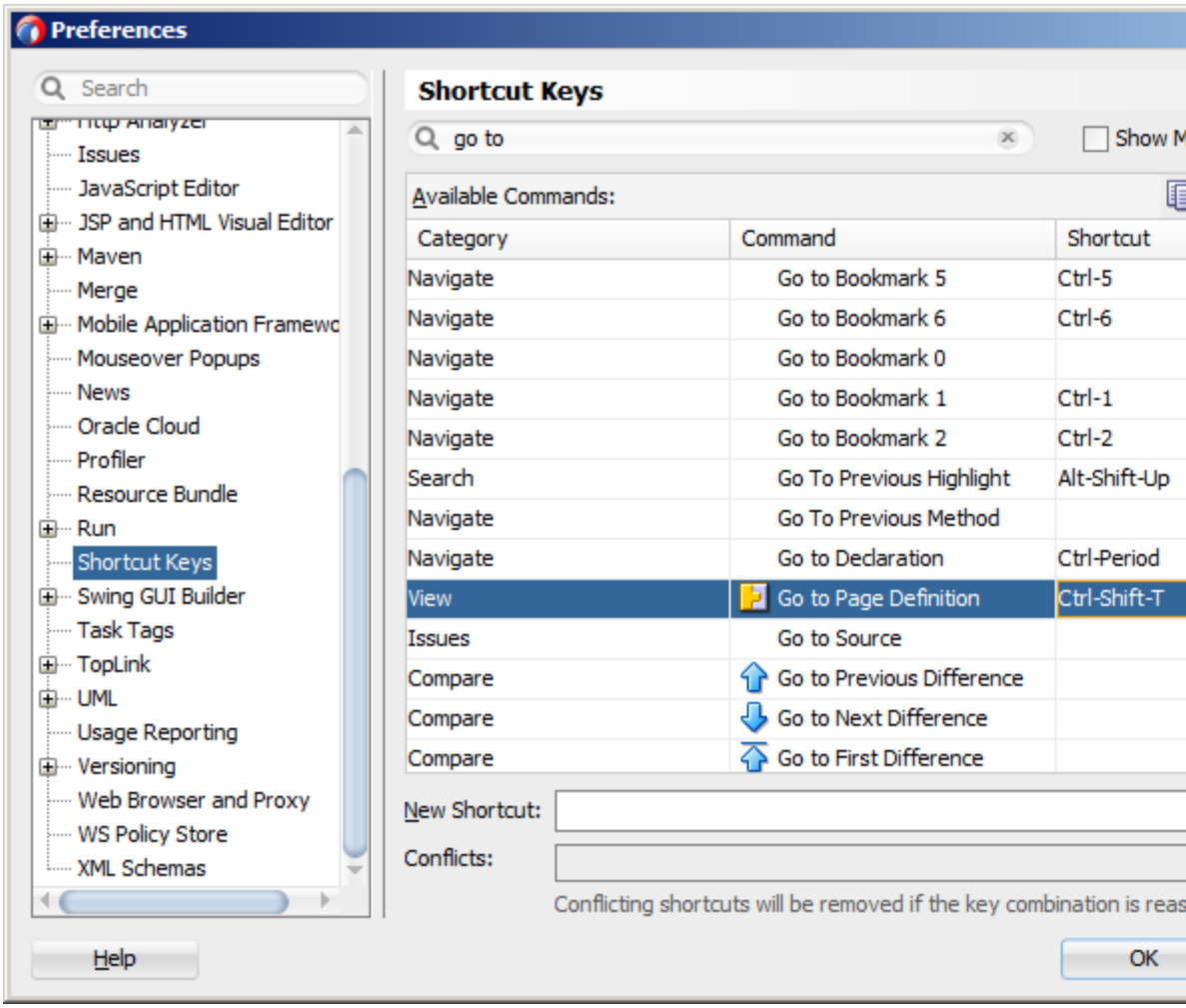
- **Structure** window, as Figure 13-40 shows.

Figure 13-40 Go to Page Definition from Structure Pane



In addition, you can open the **Page Definition** file using the **Go to Page Definition** shortcut key defined under **Tools > Preferences** on the main menu, as Figure 13-41 shows.

Figure 13-41 Opening Page Definition from Preferences



13.3.1.6 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 a 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 a MAF AMX fragment file (`.amxf`) onto a MAF AMX page or another fragment file to create a reference to the fragment and to define its attributes (see Configuring the Fragment Content). The fragment file resides inside your project and you can drop it from the **Applications** window.

Before you begin:

Ensure that the MAF application includes a View Controller project.

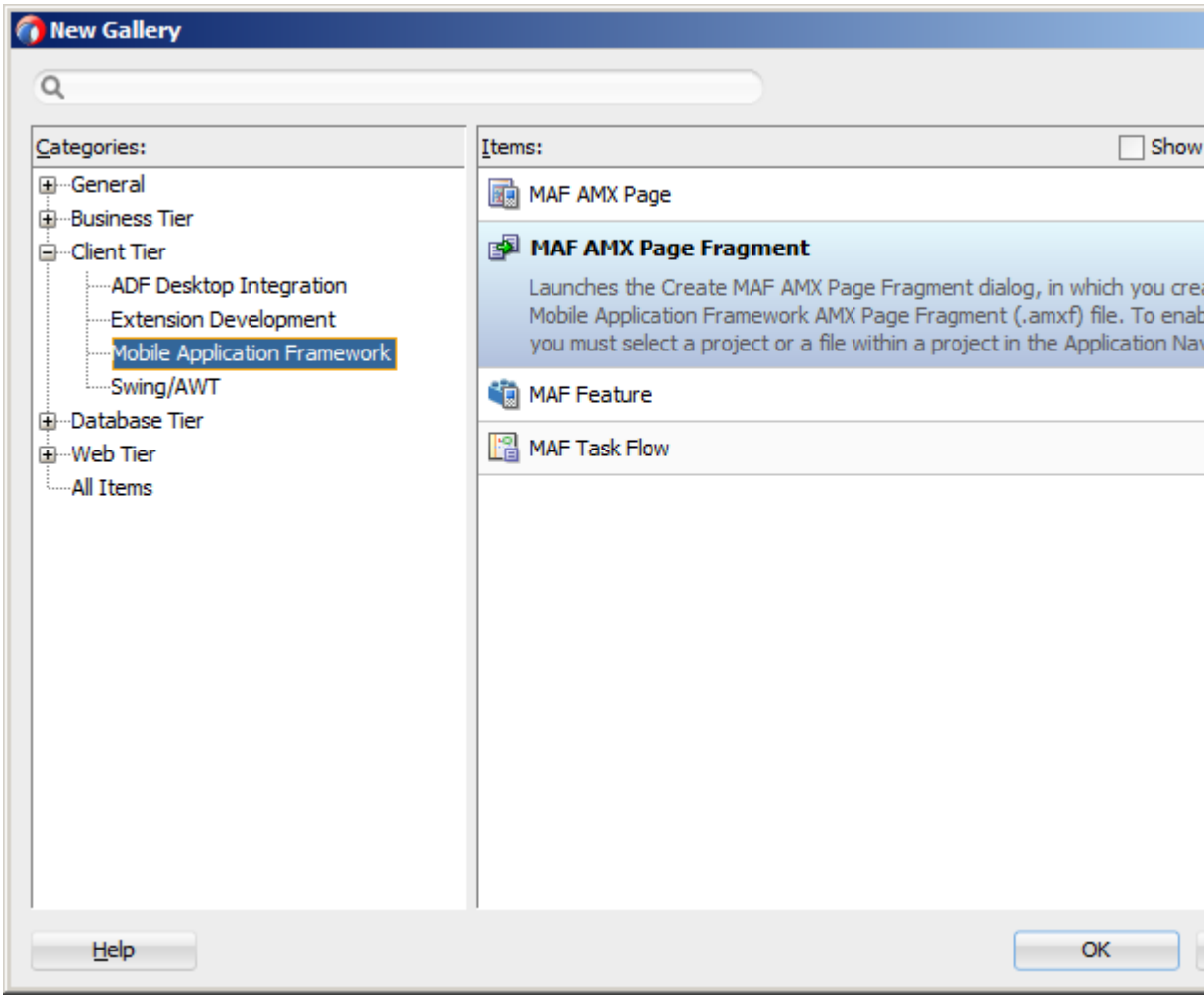
If the View Controller project does not contain a MAF AMX page or MAF AMX page task flow from which to create a page, you can invoke the **Create MAF AMX Page** dialog by double-clicking a view icon in a task flow diagram or by selecting **Client Tier > Mobile Application Framework > MAF AMX Page** from the **New Gallery** (see Creating MAF AMX Pages).

To create a Fragment from the New Gallery:

1. From the top-level menu in JDeveloper, click **File**, and then select **New > From Gallery**.

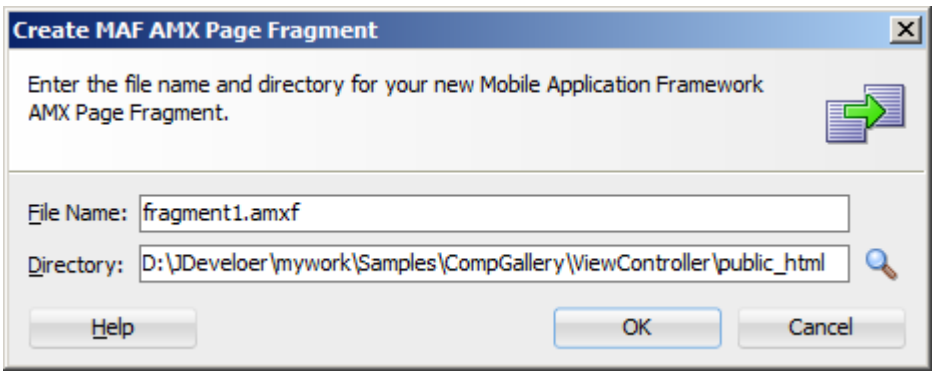
2. In the **New Gallery**, expand the **Client Tier** node, select **Mobile Application Framework,** and then **MAF AMX Page Fragment** (see Figure 13-42). Click **OK**.

Figure 13-42 Creating New Fragment



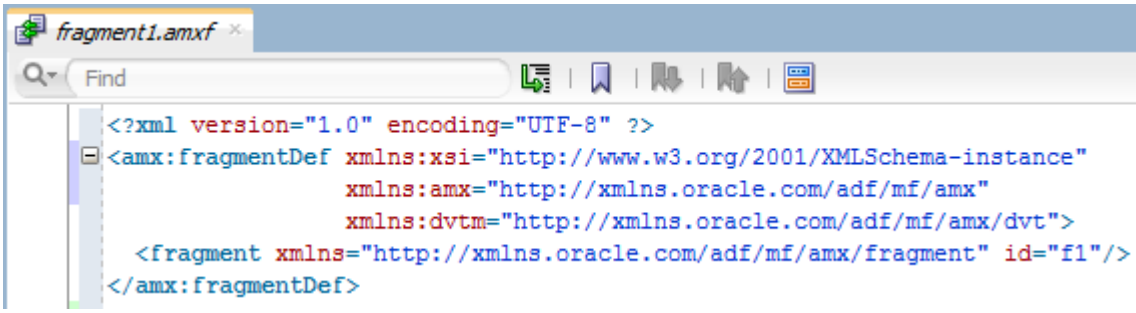
3. Complete the **Create MAF AMX Page Fragment** dialog by entering the file name and location of the new fragment, as Figure 13-43 shows. Click **OK**.

Figure 13-43 Create MAF AMX Page Fragment Dialog



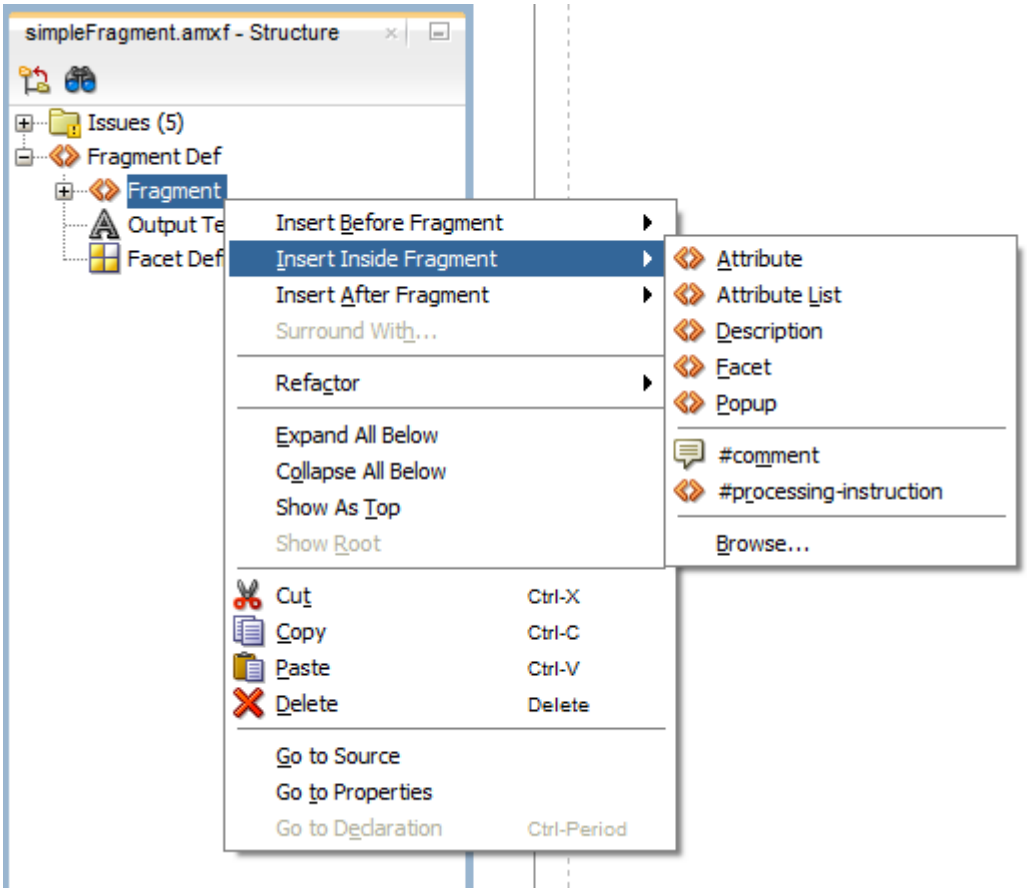
Upon completion of the dialog, a newly created file opens in the **Source** editor of JDeveloper (see Figure 13-44).

Figure 13-44 Fragment File



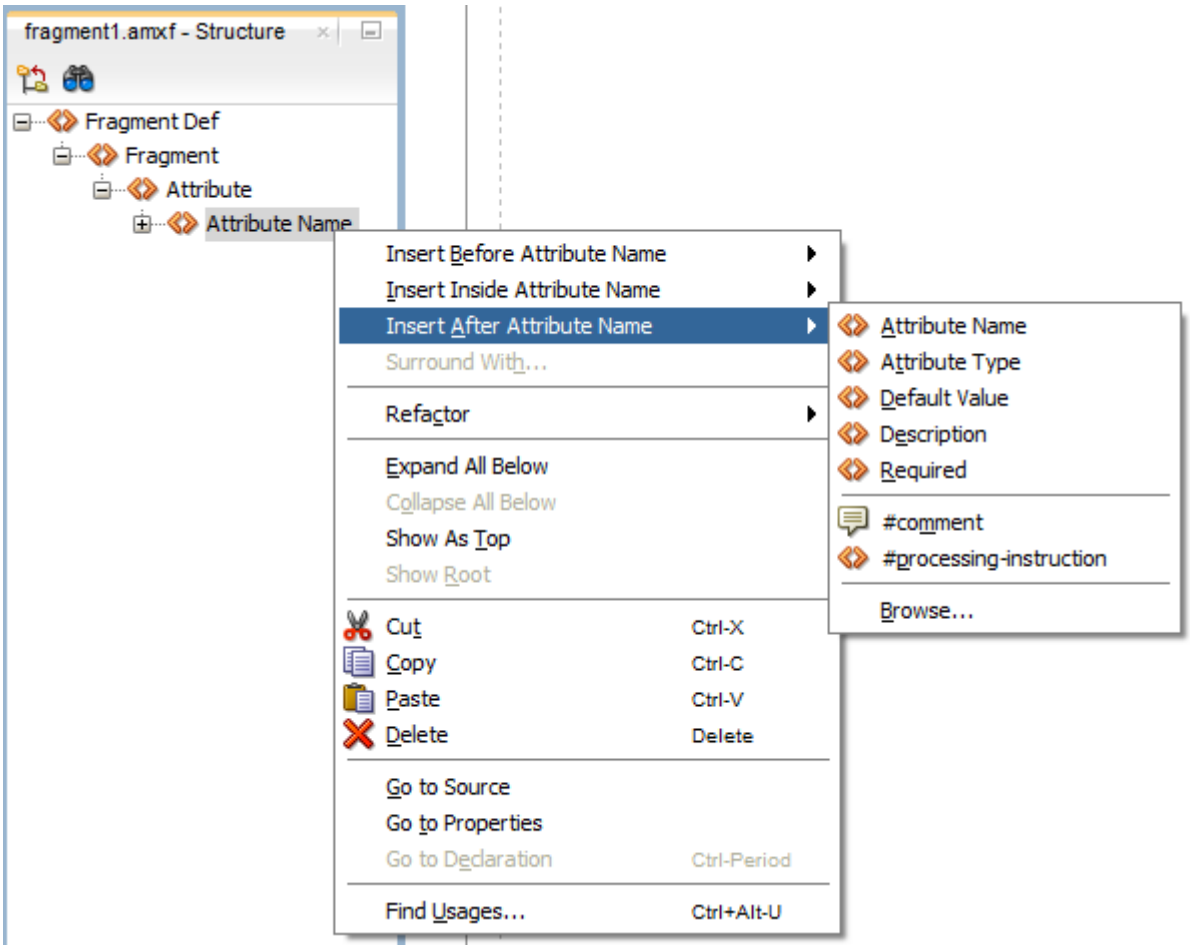
4. Right-click **Fragment** in the **Structure** window and select **Insert Inside Fragment**. Choose elements with which to populate the new fragment (see Figure 13-45): **Attribute**, **Attribute List**, **Description**, **Facet**, or **Popup**.

Figure 13-45 Populating Fragment



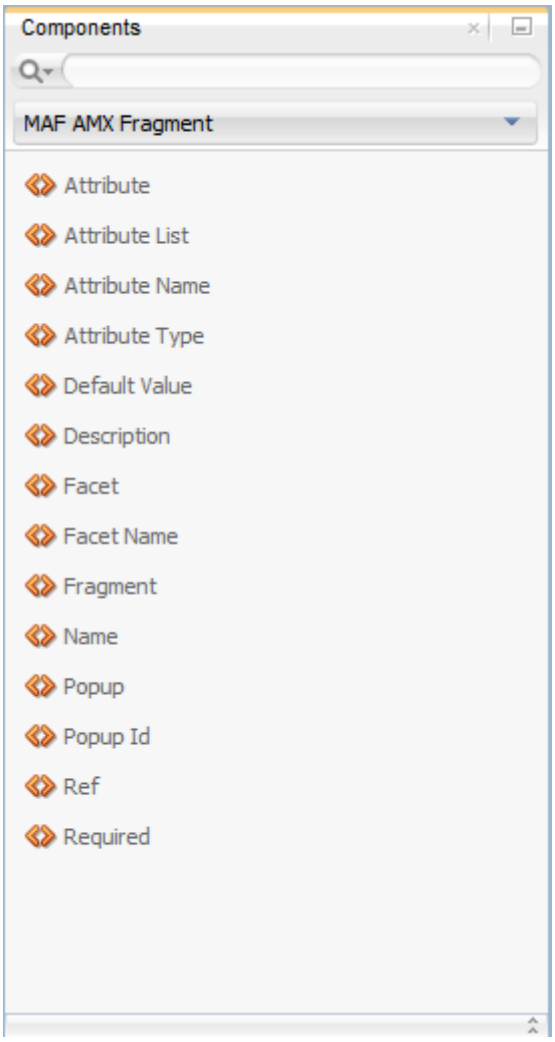
5. Proceed by defining the Fragment's **Attribute** and other children by right-clicking that child in the **Structure** view and selecting appropriate values (see Figure 13-46).

Figure 13-46 Defining Fragment's Attribute



You can also define the Fragment by dragging and dropping its elements onto the MAF AMX fragment file by selecting **MAF AMX Fragment** in the **Components** window (see Figure 13-47).

Figure 13-47 Dragging and Dropping Fragment Elements



The following example shows a MAF AMX fragment file called `fragment1.amxf`.

```
<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 How to Use the Fragment Component) and set its `src` attribute to the fragment file of your choice. The following example shows a `fragment` element added to a 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 `fragmetDef`. 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.

```
<?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 receives all the information through its attributes. In addition to defining individual attributes, you can define a set of attributes to be passed to the Fragment as a list which could be iterated through in the Fragment definition. For more information, see Passing List of Attributes with Metadata to a Fragment.

Note:

EL expressions used within the Fragment file (`.amxf`) are not validated.

The Fragment supports the following:

- Embedded popups (see 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 fragment. These sample applications are located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

13.3.1.6.1 Configuring the Fragment Content

When you drag and drop a MAF AMX fragment file (`.amxf`) onto a MAF AMX page or another fragment file, the **Configure Fragment Content** dialog (see Figure 13-48) appears. This dialog displays and allows you to specify all Fragment attributes that are defined as direct children of the Fragment.

Note:

Facets, Popup components, Attribute Lists and their artifacts are not available through the **Configure Fragment Content** dialog.

Figure 13-48 Configure Fragment Content Dialog

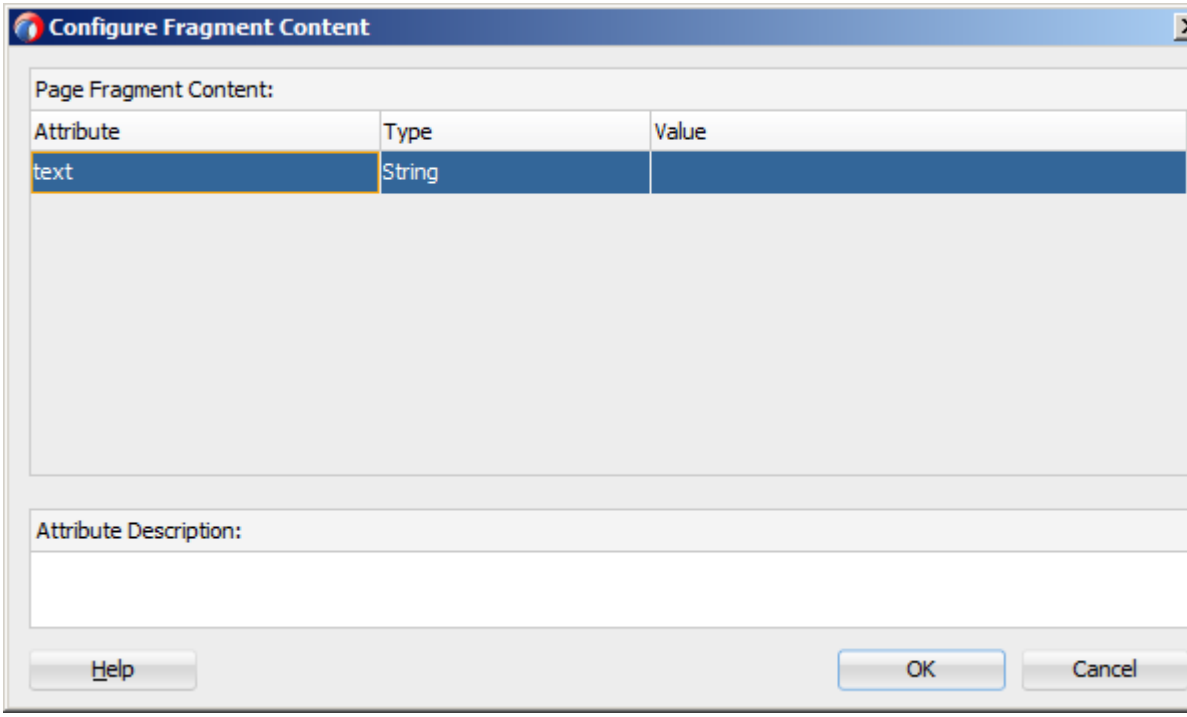
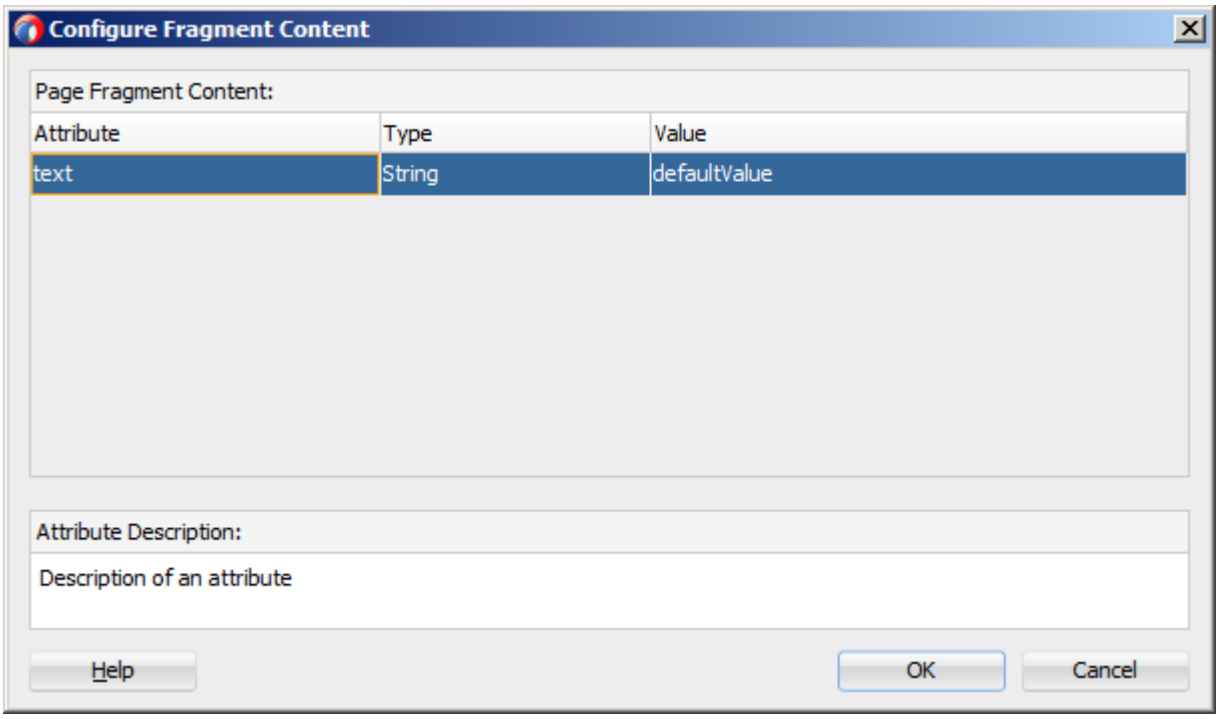


Figure 13-49 demonstrates the **Configure Fragment Content** dialog that appears when you drag and drop the MAF AMX fragment file whose contents is shown in the following example onto a MAF AMX file.

```
<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>
```

Figure 13-49 Configure Fragment Content Dialog with Values



When completing the dialog, consider the following:

- If you are configuring an attribute defined as required in the Fragment, an asterisks (*) is displayed at the end of the attribute name.
- The **OK** button of the dialog is disabled until all of the required attribute values have been defined.
- If the Fragment attribute's default value is specified and at the same time the `required` property is defined and set to `true`, this attribute is not treated as required (the default value takes precedence). In this case, the following occurs:
  - An audit warning is displayed in the Fragment.
  - The **Configure Fragment Content** dialog does not add the asterisk to the attribute's name and does not treat this attribute as required.
- The **Type** column displays the value of the `attribute-type` element from the Fragment. It is used as a description of the attribute type (as opposed to the Java class).

Note:

Even though the `attribute-type` is a required element in the Fragment, it might appear unspecified in the Fragment being dropped if you failed to define its value. In this case the dialog displays String as a default value.

- The **Value** column allows you to specify the value to pass to the Fragment's attribute. You can enter the value by typing it or clicking on the ellipsis (…) to invoke the EL Builder and specify an EL expression. If the `default` element is present for the given attribute in the Fragment, this default value is specified in the **Value** column for the attribute. You can override the default value.
- If the `description` element is present in the Fragment for this attribute, the bottom portion of the dialog displays the **Attribute Description** field when you switch between rows of different attributes. If the description element is not defined, the **Attribute Description** field is blank.
- You cannot add, remove, or reorder attributes using the **Configure Fragment Content** dialog.

13.3.1.6.2 Passing List of Attributes with Metadata to a Fragment

When defining the Fragment attributes, MAF allows you to do the following:

- Pass in dynamic attributes.
- Have metadata associated with each attribute (see Passing List of Attributes with Metadata to a Fragment).
- Loop over attributes in the Fragment definition.
- Nest dynamic attributes in an attribute.
- Pass dynamic attributes from one Fragment to an embedded Fragment.

Table 13-4 lists direct and indirect child elements of the MAF AMX `fragment` that enable you to pass lists of attributes.

Table 13-4 Attribute-Related Child Elements of the Fragment

Child Attribute Name	Description
`attributeList`	Defines an attribute list to pass to a Fragment. Can be a direct child of the MAF AMX `fragment` or `attributeSet` element. There can be any number of the child `attributeList` elements defined for a parent element. The `attributeList` element may be referenced by another `attributeList` through its `ref` attribute. An attribute list may be passed from one Fragment to another by reference, in which case both attribute lists must have the same metadata.

Table 13-4 (Cont.) Attribute-Related Child Elements of the Fragment

Child Attribute Name	Description
`attributeSet`	Can be specified as a child of the MAF AMX `attributeList`. Defines one set of attributes to be used during an iteration of the MAF AMX `attributeListIterator`. May be thought of as one item in an array. There can be any number of the child `attributeSet` elements defined for a parent `attributeList` element.
`attributeListIterator`	Consumes a MAF AMX `attributeList`. Behaves similarly to the MAF AMX Iterator in terms of stamping, but exposes attributes differently, with its `name` attribute tying the iterator to the `attributeList`. When one `attributeListIterator` is nested inside another, the `name` must point to the `attributeList` which is a child of the `attributeSet` currently being processed by the `attributeListIterator`. During the iteration, the defined attribute names are exposed as EL variables. For attributes that are not provided by the caller and in cases when the attribute has no default value, the value `adf.mf.api.OptionalFragmentArgument` is used as an EL variable. You may test for this condition by using the empty EL keyword (for example, `rendered="#{not (empty myAttribute)}"`).

For information on attributes and their values, see *Tag Reference for Oracle Mobile Application Framework*.

The following example demonstrates the basic case of passing an Attribute List to a MAF AMX Fragment.

```
<amx:fragment src="something.amxf">
   <amx:attributeList name="attributeToPass" ref="nameOfAnOuterAttributeList" />
</amx:fragment>
```

The following example shows the `fragment` element with the child `attributeList` defined in the MAF AMX file.

```
<amx:fragment src="summaryView.amxf">
   <amx:attributeList name="attrs">
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.firstName}"/>
         <amx:attribute name="displayType" value="string" />
      </amx:attributeSet>
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.lastName}"/>
         <amx:attribute name="displayType" value="string" />
      </amx:attributeSet>
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.homePhone}"/>
         <amx:attribute name="displayType" value="phone" />
      </amx:attributeSet>
   </amx:attributeList>
</amx:fragment>
```

The following example shows nested `attributeList` elements defined within the `fragment` element in the MAF AMX file.

```
<amx:fragment src="summaryView.amxf">
   <amx:attributeList name="attrs">
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.firstName}"/>
         <amx:attribute name="displayType" value="string" />
      </amx:attributeSet>
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.lastName}"/>
         <amx:attribute name="displayType" value="string" />
      </amx:attributeSet>
      <amx:attributeSet>
         <amx:attribute name="attribute" value="#{bindings.homePhone}"/>
         <amx:attribute name="displayType" value="phone" />
      </amx:attributeSet>
      <amx:attributeSet>
         <amx:attributeList name="subAttributes">
            <amx:attributeSet>
               <amx:attribute name="attribute"
                              value="#{bindings.spouseFirstName}"/>
               <amx:attribute name="displayType"
                              value="string" />
            </amx:attributeSet>
            <amx:attributeSet>
               <amx:attribute name="attribute"
                              value="#{bindings.spouseLastName}"/>
               <amx:attribute name="displayType"
                              value="string" />
            </amx:attributeSet>
         </amx:attributeList>
         <amx:attribute name="label" value="Spouse"/>
      </amx:attributeSet>
   </amx:attributeList>
</amx:fragment>
```

The following example shows how a Fragment with defined Attribute List components is used within the Fragment Definition. The `amxf:attribute-list` tag defines the metadata for an Attribute List. This tag must be declared as a child of the `amxf:fragment` or another `amxf:attribute-list` tag and its valid child tags are `amxf:name`, `amxf:description`, `amxf:attribute-list`, and `amxf:attribute`. The `name` child is required and must be unique within the current XML node. Even though the name could be identical to the `amxf:attribute` that is declared on the same level, such naming practice is not recommended.

```
<amx:fragmentDef
   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">
      <attribute-list>
         <name>attributes</name>
         <attribute>
            <attribute-name>attribute</attribute-name>
         </attribute>
         <attribute>
            <attribute-name>displayType</attribute-name>
         </attribute>
         <attribute>
            <attribute-name>label</attribute-name>
         </attribute>
         <attribute-list>
            <name>subAttributes</name>
            <attribute>
```

```
            <attribute-name>attribute</attribute-name>
          </attribute>
          <attribute>
            <attribute-name>displayType</attribute-name>
          </attribute>
        </attribute-list>
      </attribute-list>
    </fragment>
    ...
    <amx:attributeListIterator name="attributes">
      <amx:panelLabelAndMessage label="#{attribute.hints.label}"
                                id="plam1"
                                rendered="#{not (empty attribute)}">
        <amx:outputText value="#{attribute.inputValue}" id="ot1"/>
      </amx:panelLabelAndMessage>
      <amx:outputText value="#{label}"
                      id="ot2"
                      rendered="#{not (empty label)}"/>
      <amx:attributeListIterator name="subAttributes"
                                 rendered="#{not (empty subAttributes)}">
        <amx:panelLabelAndMessage label="#{attribute.hints.label}"
                                  id="plam2"
                                  rendered="#{not (empty attribute)}">
          <amx:outputText value="#{attribute.inputValue}" id="ot3"/>
        </amx:panelLabelAndMessage>
      </amx:attributeListIterator>
    </amx:attributeListIterator>
    ...
</amx:fragmentDef>
```

The `attribute-list`, `attribute-set`, and `attribute` tags could be used in the `fragment` node to define the following:

- Attribute List components that are allowed.
- The information necessary to validate the page that is calling the Fragment.

13.3.2 How to Add UI Components to a MAF AMX Page

After you create a MAF AMX page, you can start adding MAF AMX UI components to your page.

You can use the **Components** window to drag and drop MAF AMX components and MAF AMX data visualization components onto the page. JDeveloper then adds the necessary declarative page code and sets certain values for component attributes.

The **Components** window displays MAF AMX components by categories (see Figure 13-50):

- General Controls
- Text and Selection
- Data Views
- Layout, with the following subcategories:
  - Interactive Containers and Headers
  - Secondary Windows

  - Core Structure
- Operations, with the following subcategories:
  - Behavior
  - Listeners
  - Validators and Converters

For information on adding and using specific components, see Creating and Using UI Components.

The **Components** window also displays MAF AMX data visualization components by categories (see Figure 13-50):

- 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 Controller project, which may or may not contain a MAF AMX page or MAF AMX page task flow from which to create a page.

As described in 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 **Client Tier > Mobile Application Framework > MAF AMX Page** from the **New Gallery**.

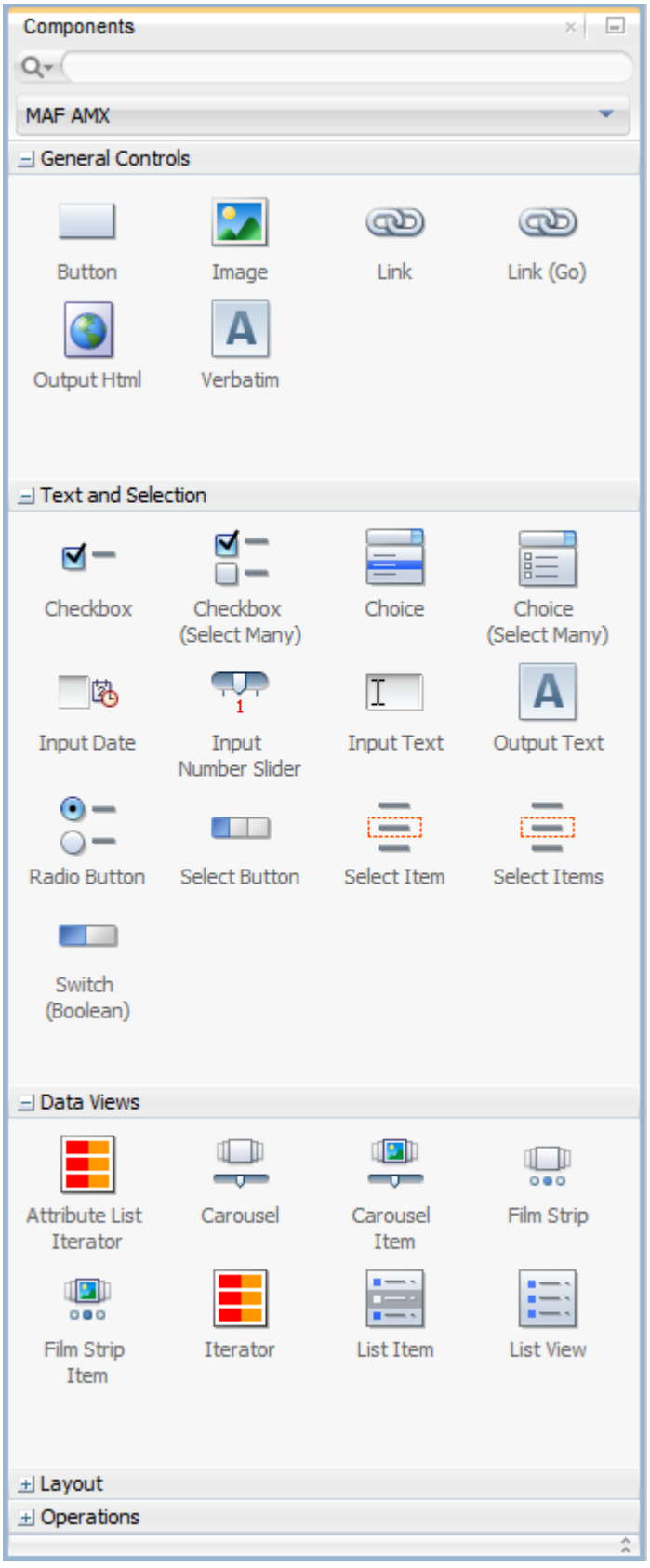
To add a UI component to a page:

1. Open a **MAF AMX page** in the **Source** editor (default).
2. In the **Components** window, use the menu to choose **MAF AMX**, as Figure 13-50 shows.

Tip:

If the **Components** window is not displayed, choose **Window > Components** from the main JDeveloper menu. By default, the Components is displayed in the upper right-hand corner of JDeveloper.

Figure 13-50 MAF AMX Components Window



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:

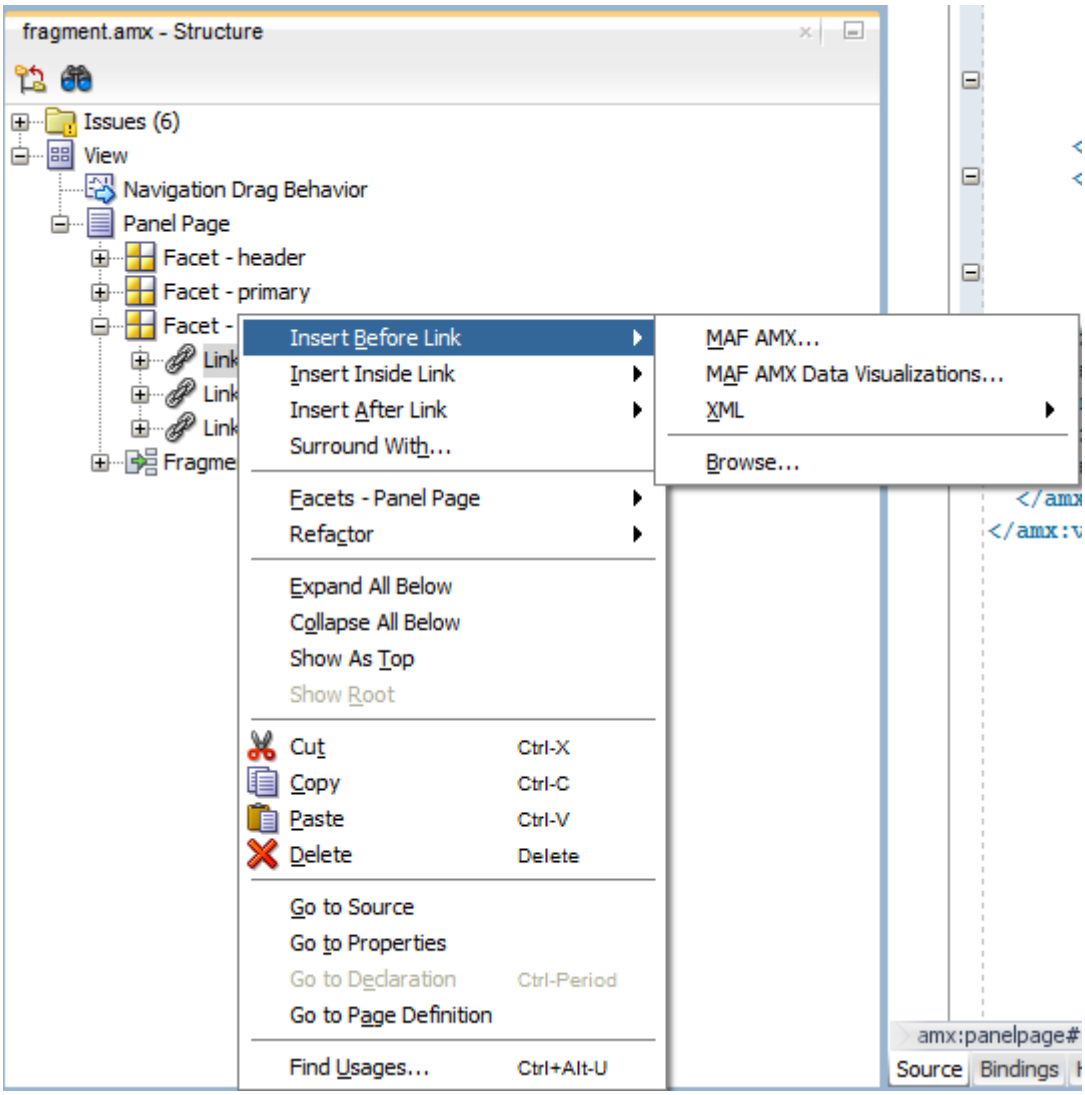
   When building a MAF AMX page, you can only drop UI components into UI containers such as, for example, a Panel Group Layout.

   JDeveloper redraws the page in the **Preview** pane with the newly added component.

Alternatively, you can add UI components and data visualization components from the **Structure** window as follows:

1. On the **Structure** window, select an existing component that you want to use as a starting point for inserting another component.
2. Right-click the selected component and choose one of the options: **Insert Before <component>**, **Insert Inside <component>**, or **Insert After <component>**, as Figure 13-51 shows.

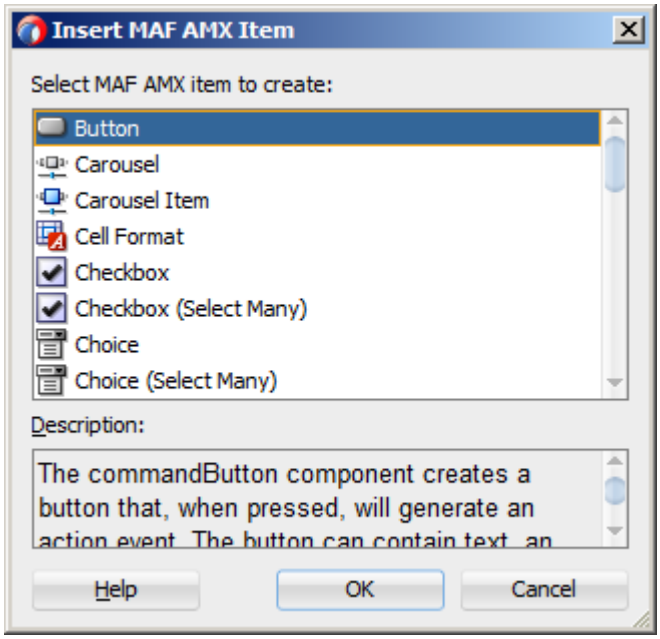
Figure 13-51 Inserting Components from Structure Window



3. From the context menu, select either **MAF AMX** or **MAF AMX Data Visualizations**:

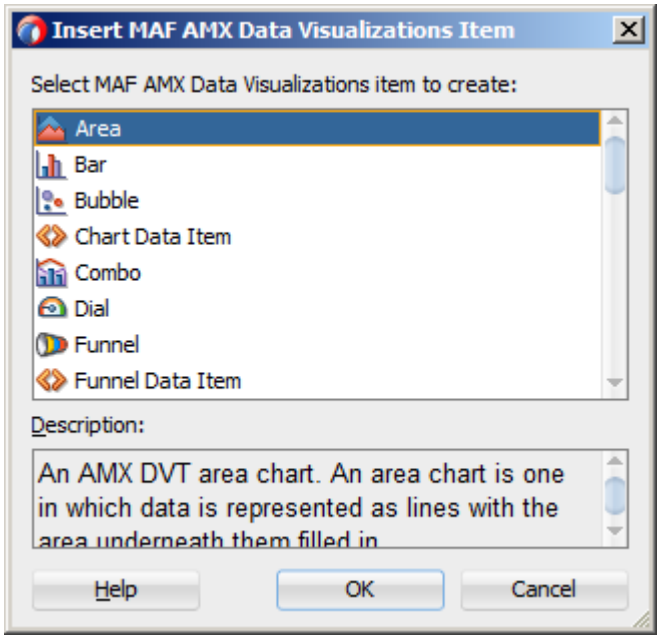
   - If you select **MAF AMX**, the **Insert MAF AMX Item** dialog opens allowing you to choose the UI component to add to the page, as Figure 13-52 shows.

Figure 13-52 Inserting MAF AMX Component



- If you select **MAF AMX Data Visualizations**, the **Insert MAF AMX Data Visualizations Item** dialog opens allowing you to choose the data visualization component to add to the page, as Figure 13-53 shows.

Figure 13-53 Inserting MAF AMX Data Visualization Component



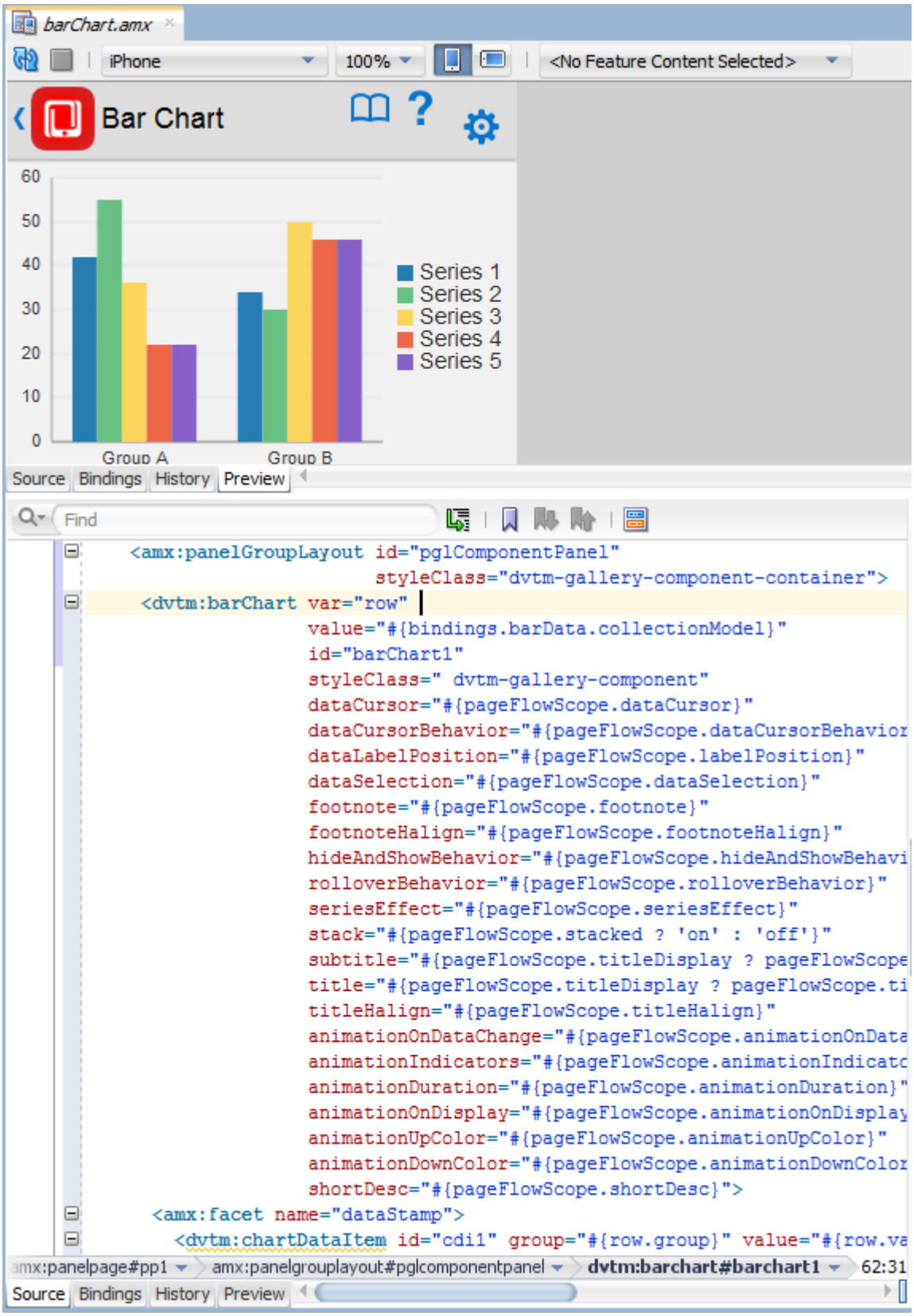
JDeveloper redraws the page in the **Preview** pane with the newly added component.

13.3.2.1 Using the Preview

JDeveloper's **Preview** provides WYSIWYG support for both the iOS and Android platforms when you build the user interface using MAF AMX files. As illustrated in Figure 13-54, splitting a view while adding the MAF AMX components to the MAF

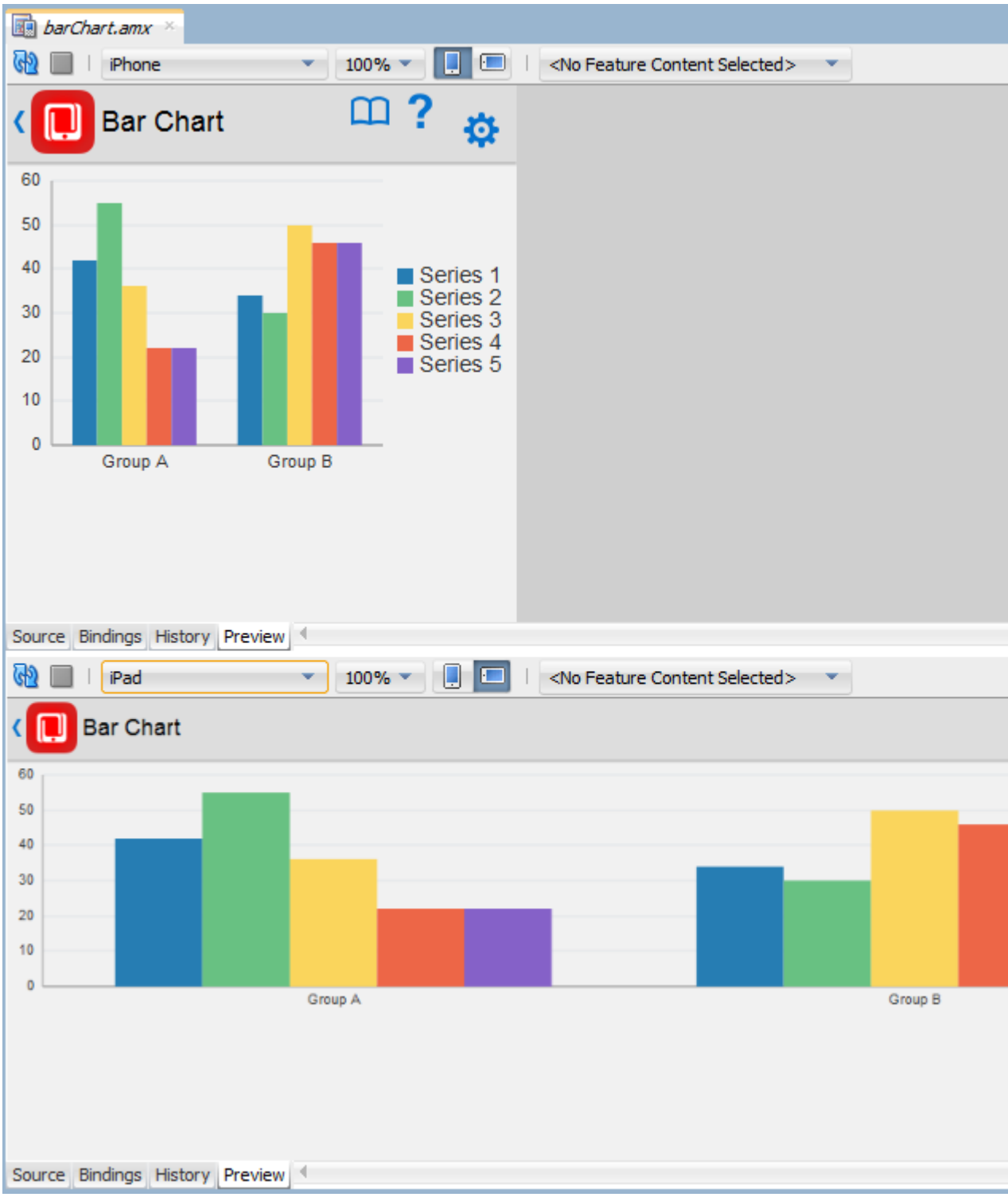
AMX file enables you to see both the code view through the **Source** editor and a UI view through the **Preview** pane. As a result, you can modify the source and get instant feedback in terms of the look and feel of that application on both the iOS and Android platforms.

Figure 13-54 Splitting Design and Source Views



In addition to being able to see the design and source views simultaneously, you can also open and work with multiple design views at the same time, as well as set each one to a different platform and screen size. By opening a combination of design views for different devices, you can develop applications simultaneously for different platforms and form factors using different orientation. Figure 13-55 shows a split screen with iPhone on the top and iPad with 75% scaling on the bottom. You can split the **Preview** pane using the default split functionality of JDeveloper.

Figure 13-55 Multiple Design Views



Note:

A MAF AMX page is rendered even for an invalid MAF AMX file. Errors are indicated by the error icon on a component. By moving the mouse over the error icon, you can view the error details.

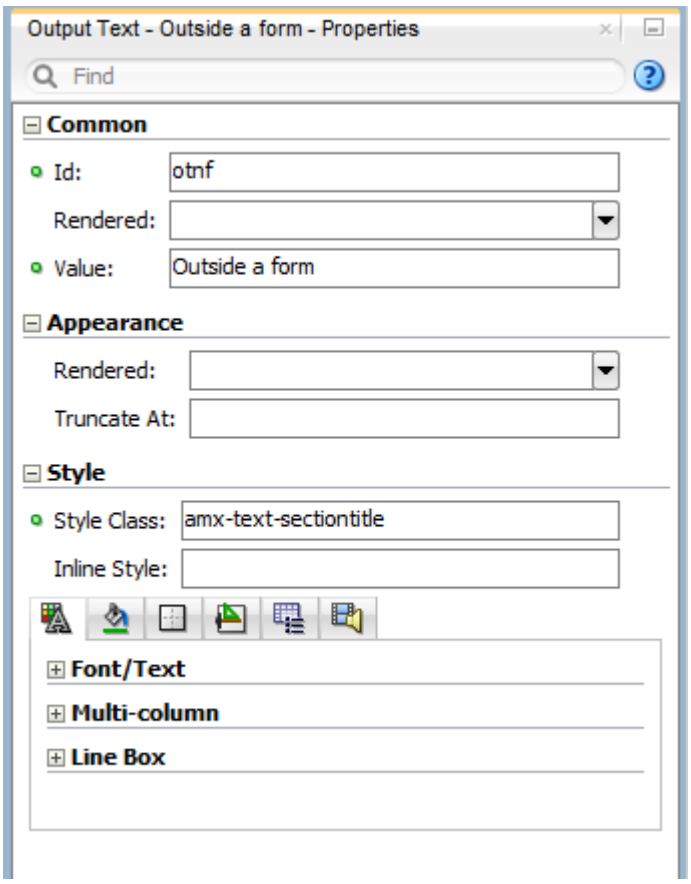
13.3.2.2 Configuring UI Components

Once you drop UI components onto a page, you can use the Properties window (displayed by default at the bottom right of JDeveloper) to set attribute values for each component.

> **Tip:**
>
> If the Properties window is not displayed, choose **Window > Properties** from JDeveloper's main menu.

Figure 13-56 shows the Properties window displaying the attributes for an Output Text component.

Figure 13-56 The Properties Window



To set component attributes:

1. Select the component for which you want to set attributes. You can select the component in the Structure window or you can select its tag directly in the Source editor.
2. In the Properties window, expand the section that contains the attribute you wish to set.

Tip:

Some attributes are displayed in more than one section. Entering or changing the value in one section will also change it in any other sections. You can search for an attribute by entering the attribute name in the search field at the top of the Properties window.

3. In the Properties window, either enter values directly into the fields, or if the field contains a list, use that list to select a value. You can also use the list to the right of the field, which launches a popup containing tools you can use to set the value. These tools are either specific property editors (opened by choosing **Edit**) or the Expression Builder, which you can use to create EL expressions for the value (opened by choosing **Expression Builder** or Method Expression Builder where applicable). For more information about using the Expression Builder, see How to Create an EL Expression.

When you use the Properties window to set or change attribute values, JDeveloper automatically changes the page source for the attribute to match the entered value.

Tip:

You can always change attribute values by directly editing the page in the Source editor. To view the page in the Source editor, click the **Source** tab at the bottom of the page.

13.3.2.3 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 **Components** window, 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.

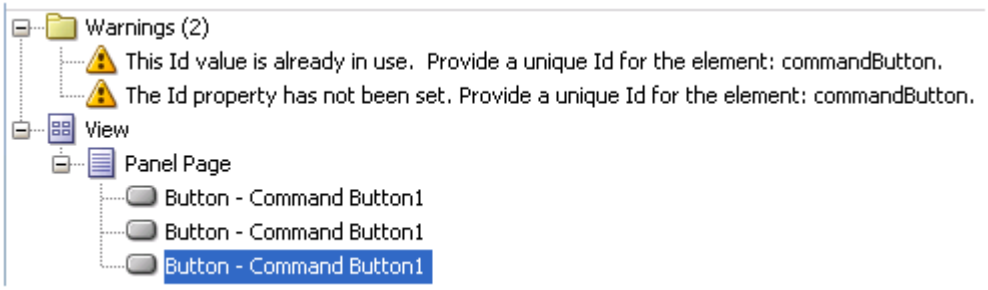
MAF provides an identifier audit utility that does the following:

- Checks the presence and uniqueness of identifiers in a 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 13-57 and Figure 13-58 show the identifier error reporting in the Source editor and Structure pane respectively.

Figure 13-57 Element Identifier Audit in Source Editor

Figure 13-58 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 How to Use a Popup Component).

Figure 13-59 and Figure 13-60 show the Show Popup Behavior's **Popup Id** and **Align Id** attributes error reporting in the **Source** editor respectively.

Figure 13-59 Popup Id Attribute Audit in Source Editor

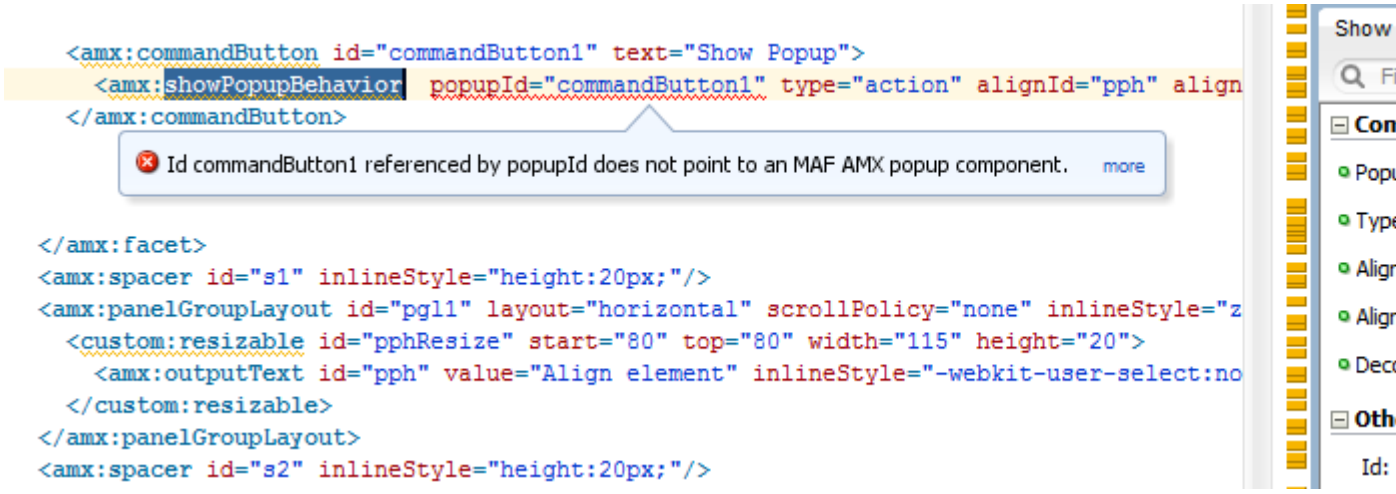
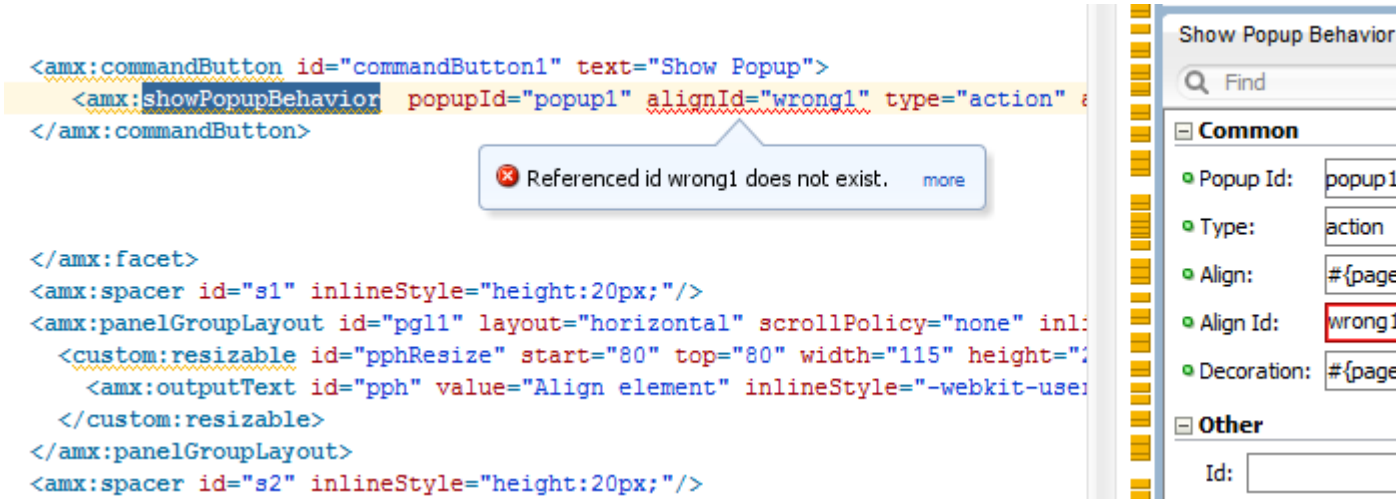


Figure 13-60 Align Id Attribute Audit in Source Editor



For more information, see "Auditing and Monitoring Java Projects " in *Developing Applications with Oracle JDeveloper*.

13.3.3 How to Add Data Controls to a MAF AMX Page

After you create a MAF AMX page, you can start adding data controls to your page.

You can create databound UI components in a 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, JDeveloper invokes a context menu of default UI components available for the item that you dropped. When you select the desired UI component, JDeveloper inserts it into a MAF AMX page. In addition, JDeveloper creates the binding information in the associated page definition file. If such file does not exist, then JDeveloper 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 a MAF AMX page.

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 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 JDeveloper, see Creating Databound UI Components from the Data Controls Panel.

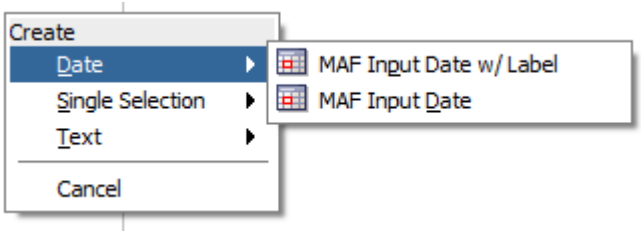
13.3.3.1 Dragging and Dropping Attributes

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:

13.3.3.1.1 Date

This category provides options for creating MAF Input Date and MAF Input Date with Label controls. Figure 13-61 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 a MAF AMX page.

Figure 13-61 Context Menu for Date Controls



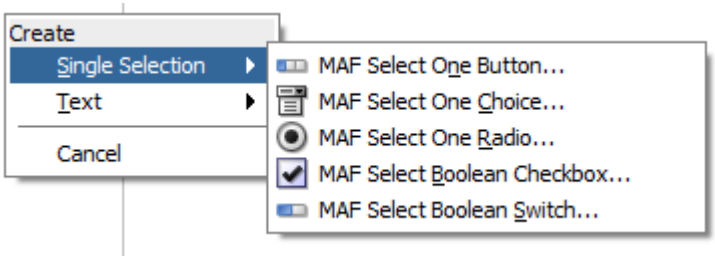
13.3.3.1.2 Single Selection

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

Figure 13-62 shows the context menu for adding selection controls that appears when you drag an attribute from the Data Controls window onto the **Source** editor or **Structure** window of a MAF AMX page.

Figure 13-62 Context Menu for Selection Controls

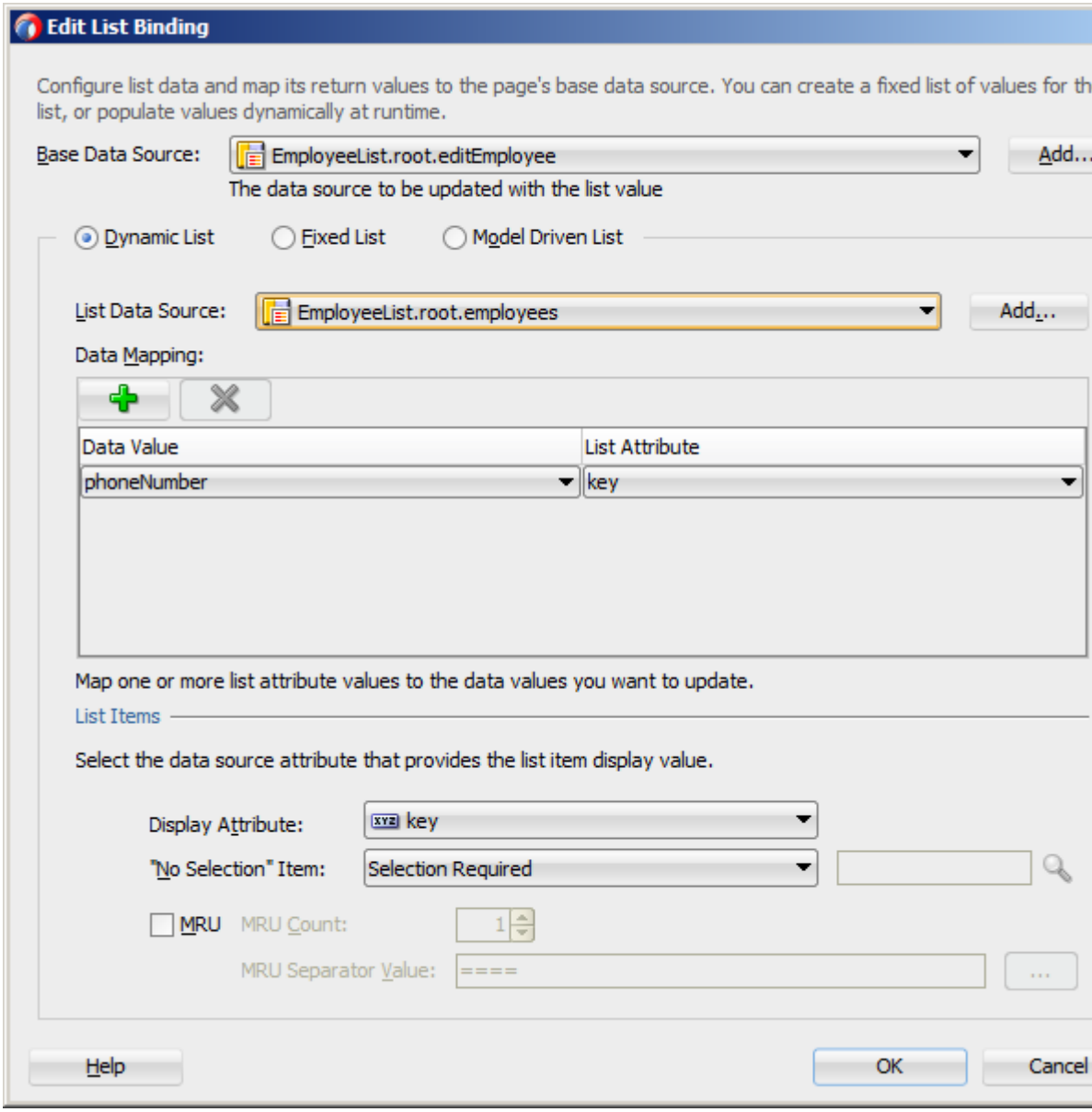


If you are working with an existing MAF AMX page and you select **MAF Select One Button** or **MAF Select One Choice** option, an appropriate version of the **Edit List Binding** dialog is displayed (see Figure 13-63). If you drop a control onto a completely new MAF AMX page, the **Edit Action Binding** dialog opens instead. After you click **OK**, the **Edit List Binding** dialog opens.

Note:

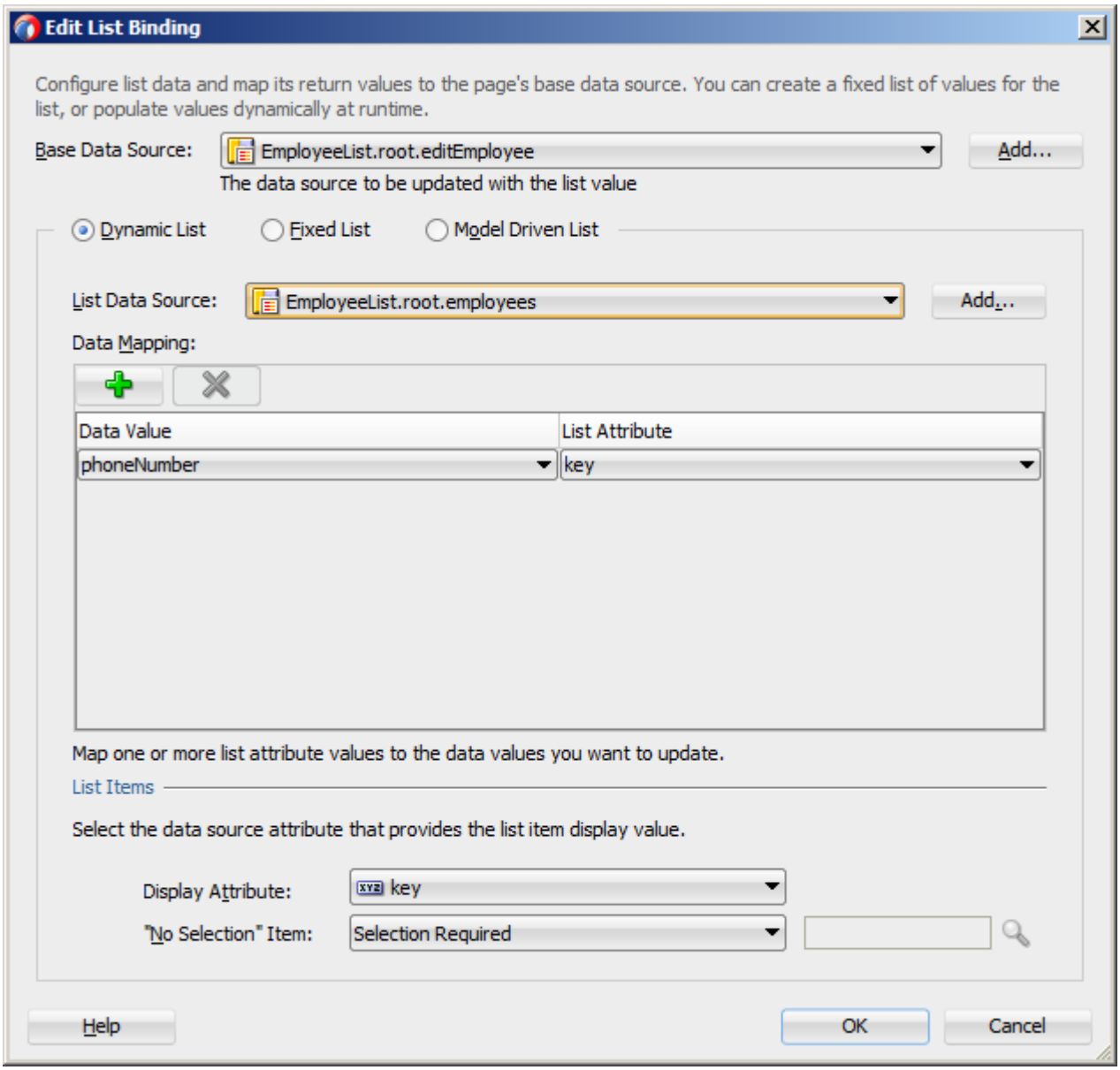
The **Edit List Binding** or **Edit Boolean Binding** dialog appears every time you drop any data control attributes as any of the single selection or boolean selection components, respectively.

Figure 13-63 Edit List Binding Dialog for Select One Button and Choice Controls



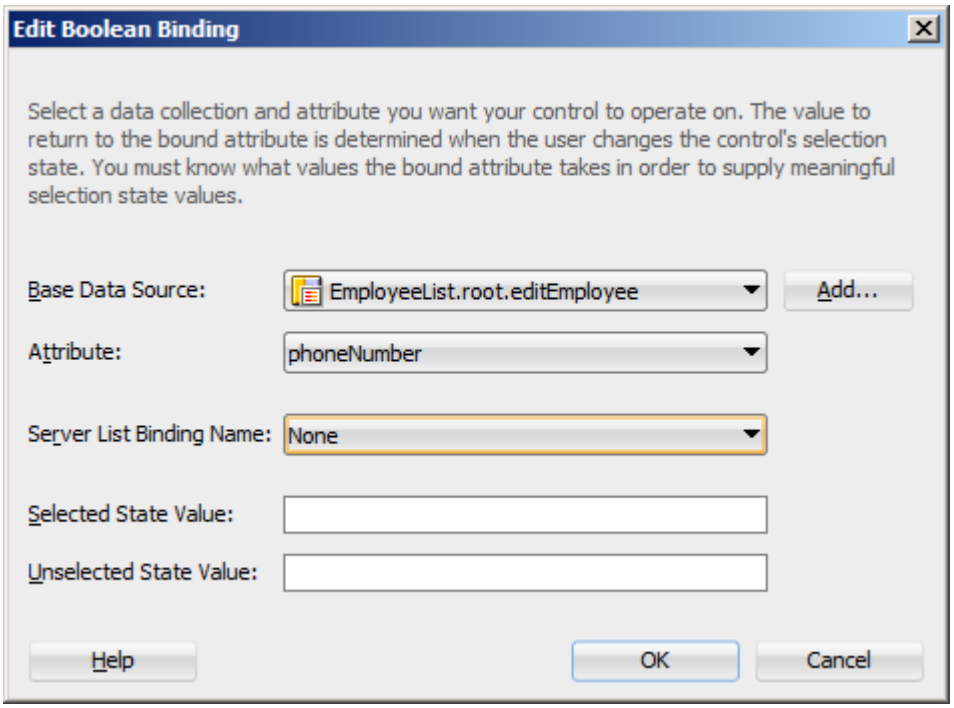
If you select **MAF Select One Radio** option, another version of the **Edit List Binding** dialog is displayed, as shown in Figure 13-64.

Figure 13-64 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 13-65.

Figure 13-65 Edit List Binding Dialog for Select Boolean Checkbox and Switch Controls



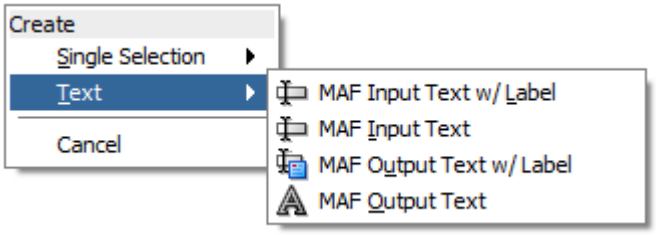
13.3.3.1.3 Text

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 13-66 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 a MAF AMX page.

Figure 13-66 Context Menu for Text Controls



13.3.3.2 Dragging and Dropping Operations

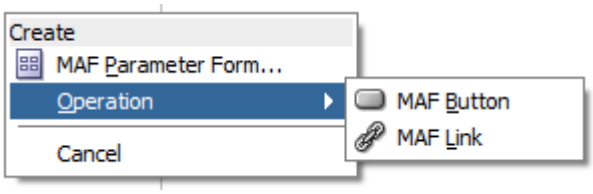
In addition to attributes, you can drag and drop operations and custom methods. Depending on the type of operation or method, different data binding menu options are provided, as follows:

13.3.3.2.1 Operation

This category is for data control operations. It provides the following options (see Figure 13-67):

- MAF Button
- MAF Link
- MAF Parameter Form

Figure 13-67 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. JDeveloper creates a binding similar to the following for the generated List Item:

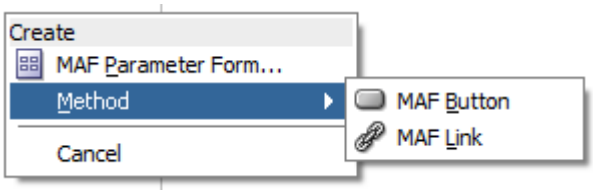
```
<amx:listItem actionListener="#{bindings.getLocation.execute}"/>
```

13.3.3.2.2 Method

This category is for custom methods. It provides the following options (see Figure 13-68):

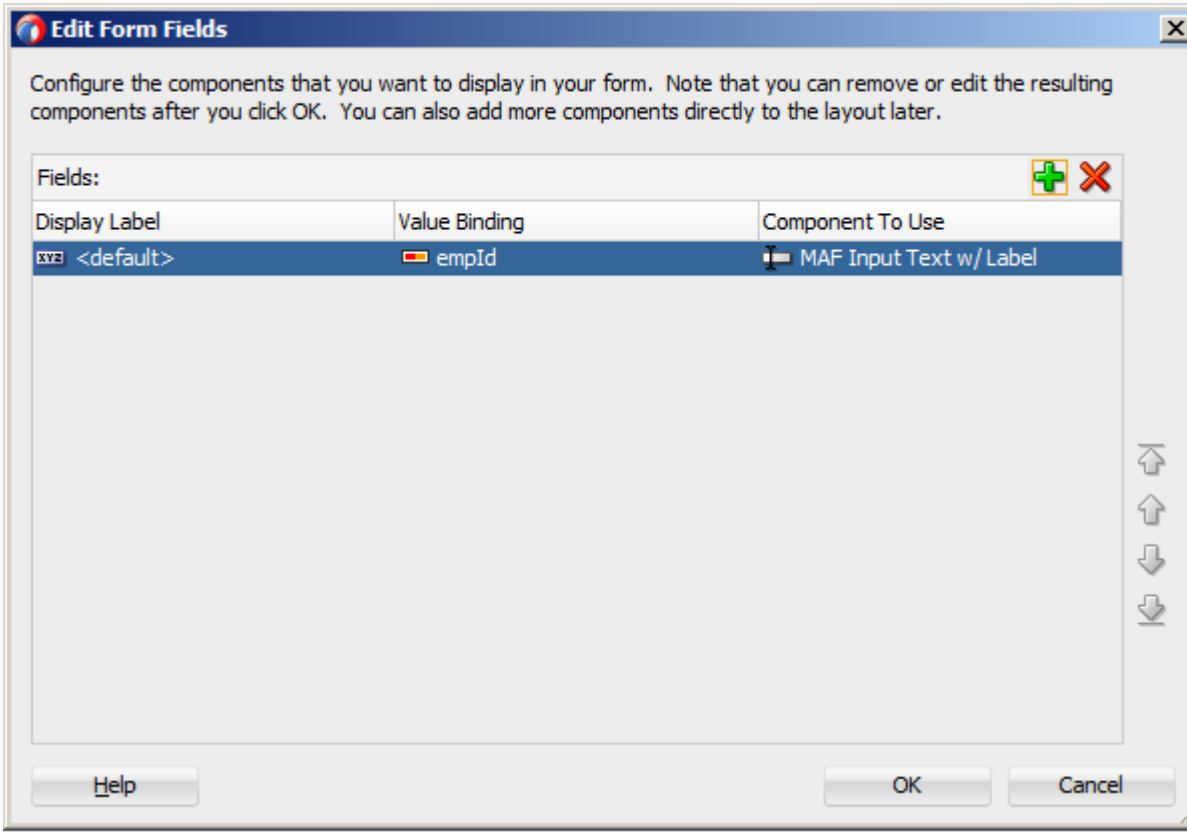
- MAF Button
- MAF Link
- MAF Parameter Form

Figure 13-68 Context Menu for Methods



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 13-69).

Figure 13-69 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 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

13.3.3.3 Dragging and Dropping Collections

You can drag and drop collections onto a MAF AMX page. Depending on the type of collection, different data binding menu options are provided, as follows:

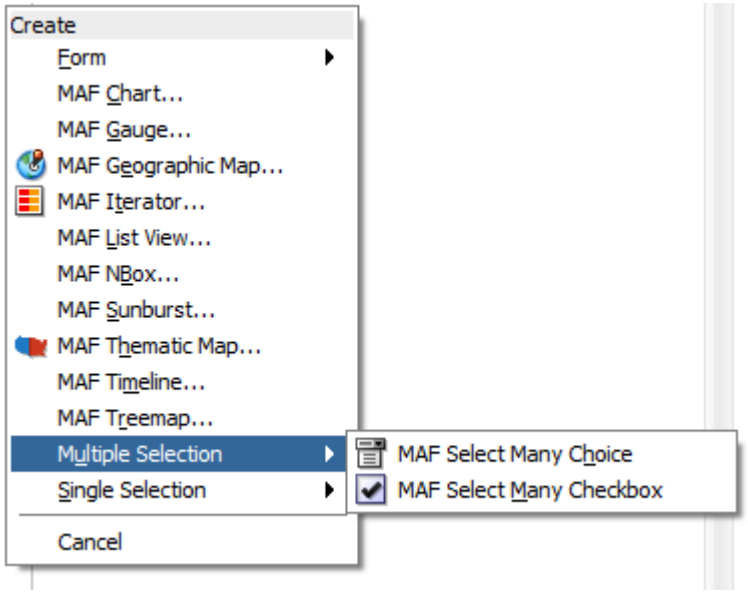
- Multiple Selection
- Form
- Iterator
- List View

13.3.3.3.1 Multiple Selection

This category provides options for creating multiple selection controls. The following controls can be created under this category (see Figure 13-70):

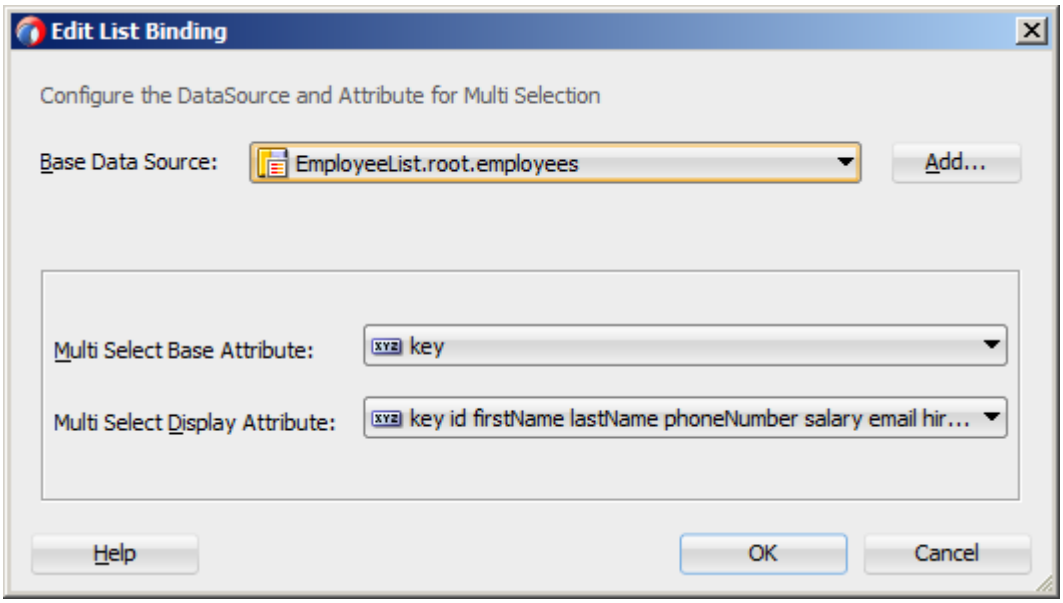
- MAF Select Many Checkbox
- MAF Select Many Choice

Figure 13-70 Context Menu for Multiple Selection Controls



If you select either **MAF Select Many Choice** or **MAF Select Many Checkbox** as the type of the control you want to create, the **Edit List Binding** dialog is displayed (see Figure 13-71).

Figure 13-71 Edit List Binding Dialog for Multiple Selection Controls



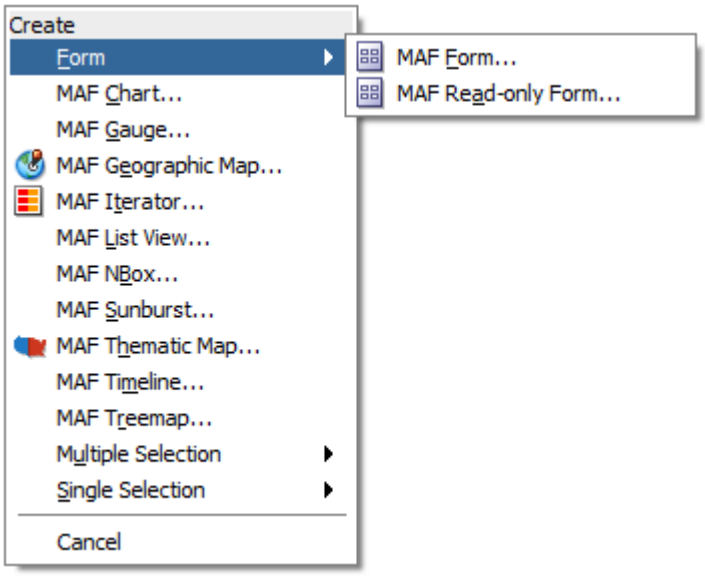
13.3.3.3.2 Form

This category is used to create the MAF AMX Panel Form controls.

The following controls can be created under the **Form** category (see Figure 13-72):

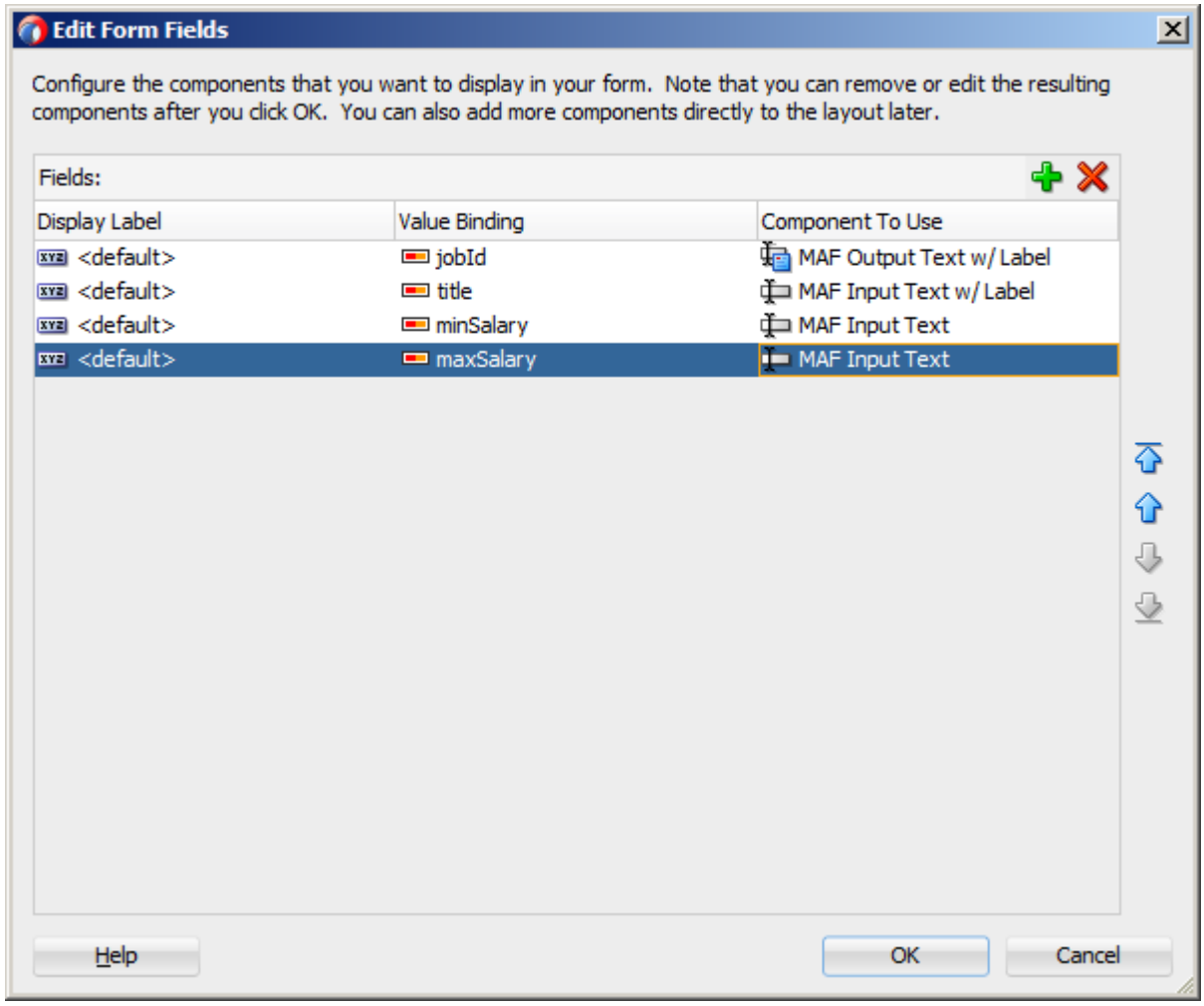
- MAF Form
- MAF Read-only Form

Figure 13-72 Context Menu for Form Controls



If you select **MAF Form** as the type of the form you want to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the form, as well as the respective controls for each of the fields (see Figure 13-73).

Figure 13-73 Edit Form Fields Dialog for MAF Form



The following data bindings are generated after you select the appropriate options in the **Edit Form Fields** dialog:

```
<amx:panelFormLayout id="pfl1">
   <amx:panelLabelAndMessage id="plam1" label="#{bindings.jobId.hints.label}">
      <amx:outputText id="ot1" value="#{bindings.jobId.inputValue}" />
   </amx:panelLabelAndMessage>
   <amx:inputText id="it4"
                  value="#{bindings.title.inputValue}"
                  label="#{bindings.title.hints.label}" />
   <amx:inputText id="it5"
                  value="#{bindings.minSalary.inputValue}"
                  simple="true" />
   <amx:inputText id="it3"
                  value="#{bindings.maxSalary.inputValue}"
                  simple="true" />
</amx:panelFormLayout>
```

For more information about generated bindings, see What You May Need to Know About Generated Bindings.

The following are supported controls for MAF Form:

- MAF Input Date

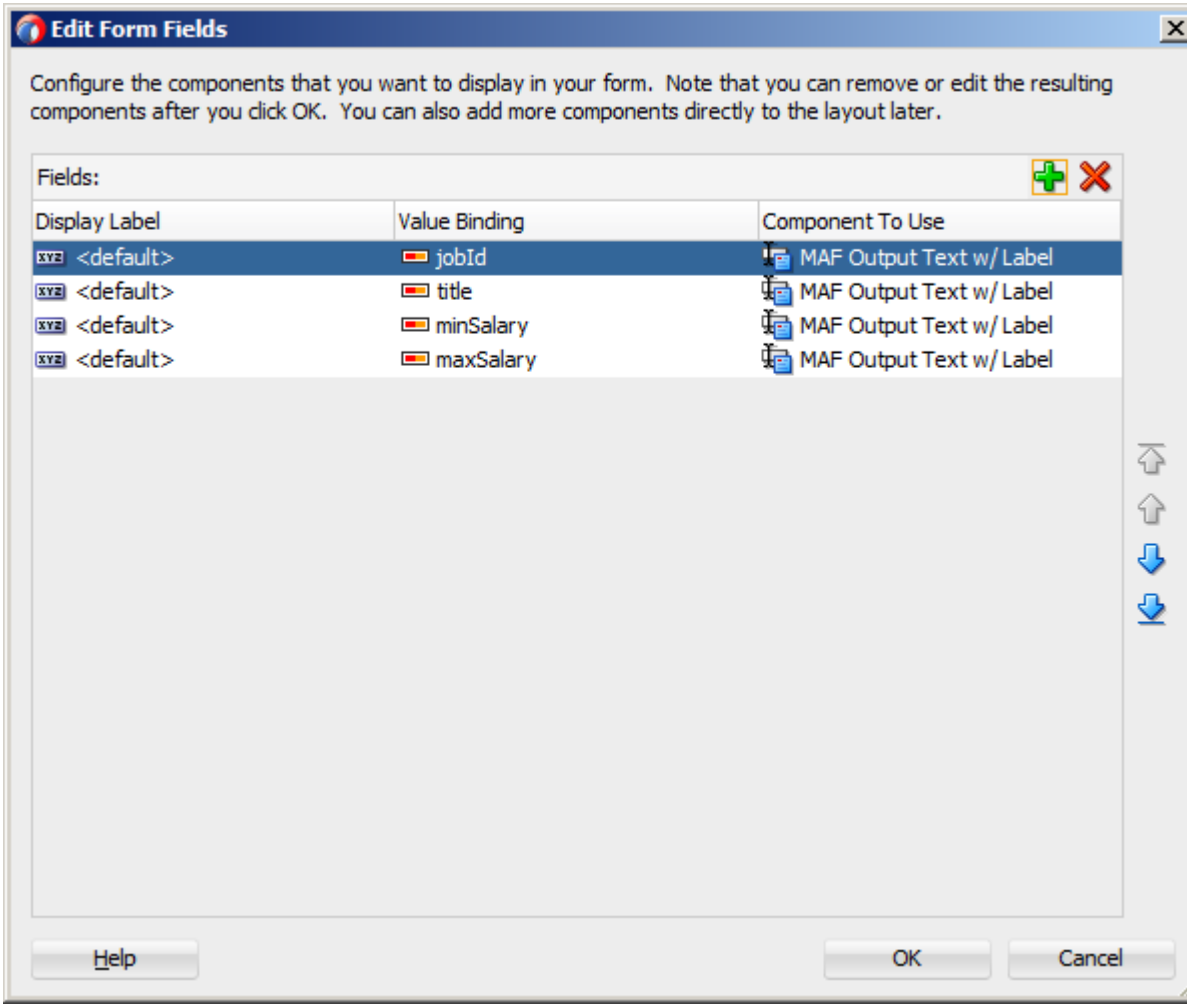
- MAF Input Date with Label
- MAF Input Text
- MAF Input Text with Label
- MAF Output Text with Label

Note:

Since MAF Output Text is not a valid Panel Form Layout child element as defined by the MAF schema, it is not supported.

If you select **MAF Read-only Form** as the type of the form you want to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the form, as well as the respective controls for each of the fields (see Figure 13-74).

Figure 13-74 Edit Form Fields Dialog for MAF Read-only Form



The following data bindings are generated after you select the appropriate options in the **Edit Form Fields** dialog:

```
<amx:panelFormLayout id="pfl1">
   <amx:panelLabelAndMessage id="plam4"
```

```
                           label="#{bindings.jobId.hints.label}">
      <amx:outputText id="ot4" value="#{bindings.jobId.inputValue}" />
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plam1"
                           label="#{bindings.title.hints.label}">
      <amx:outputText id="ot1" value="#{bindings.title.inputValue}" />
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plam3"
                           label="#{bindings.minSalary.hints.label}">
      <amx:outputText id="ot3" value="#{bindings.minSalary.inputValue}" />
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage  id="plam2"
                            label="#{bindings.maxSalary.hints.label}">
      <amx:outputText id="ot2" value="#{bindings.maxSalary.inputValue}" />
   </amx:panelLabelAndMessage>
</amx:panelFormLayout>
```

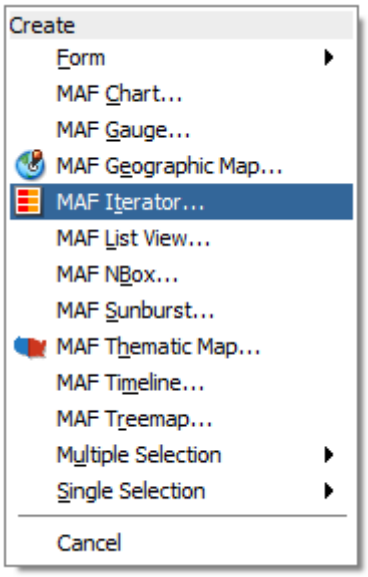
For more information about generated bindings, see What You May Need to Know About Generated Bindings.

The MAF Read-only Form only supports the MAF Output Text with Label control.

13.3.3.3.3 Iterator

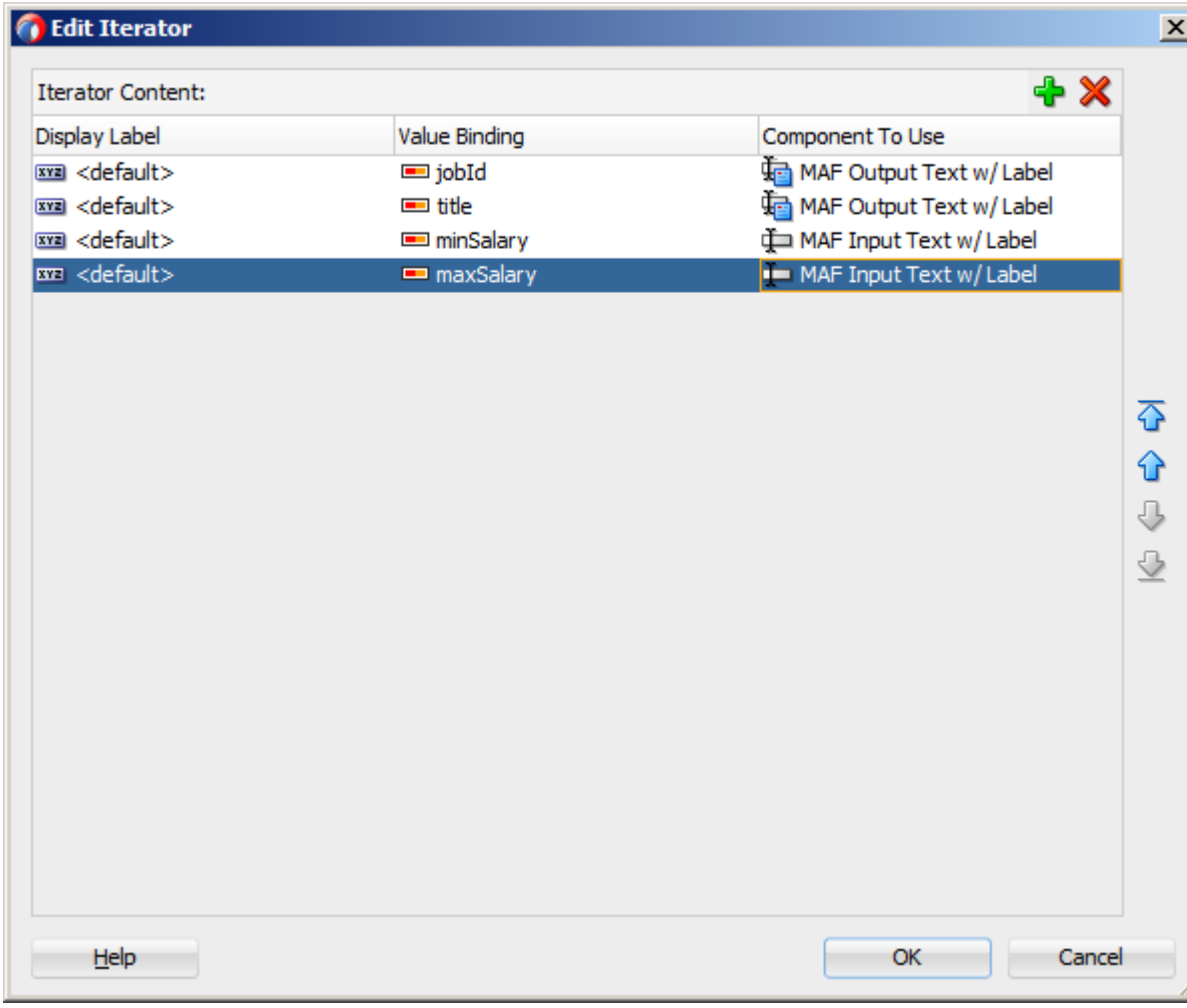
This provides an option for creating the MAF AMX Iterator with child components (see Figure 13-75).

Figure 13-75 Context Menu for Iterator Control



If you select **MAF Iterator** as the type of the control to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the iterator, as well as the respective controls for each of the fields, with **MAF Output Text w/Label** being a default selection (see Figure 13-76).

Figure 13-76 Edit Dialog for MAF Iterator



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 What You May Need to Know About Generated Bindings.

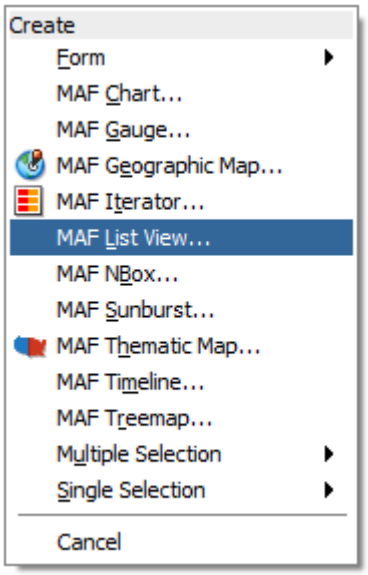
The following are supported controls for **MAF Iterator**:

- MAF Input Text
- MAF Input Text with Label
- MAF Output Text
- MAF Output Text with Label

13.3.3.3.4 List View

This provides an option for creating the **MAF AMX List View** with child components (see Figure 13-77).

Figure 13-77 Context Menu for List View Control



If you select **MAF List View** as the type of the control to create, the **ListView Gallery** opens that allows you to choose a specific layout for the **List View**, as Figure 13-78 shows.

Figure 13-78 ListView Gallery

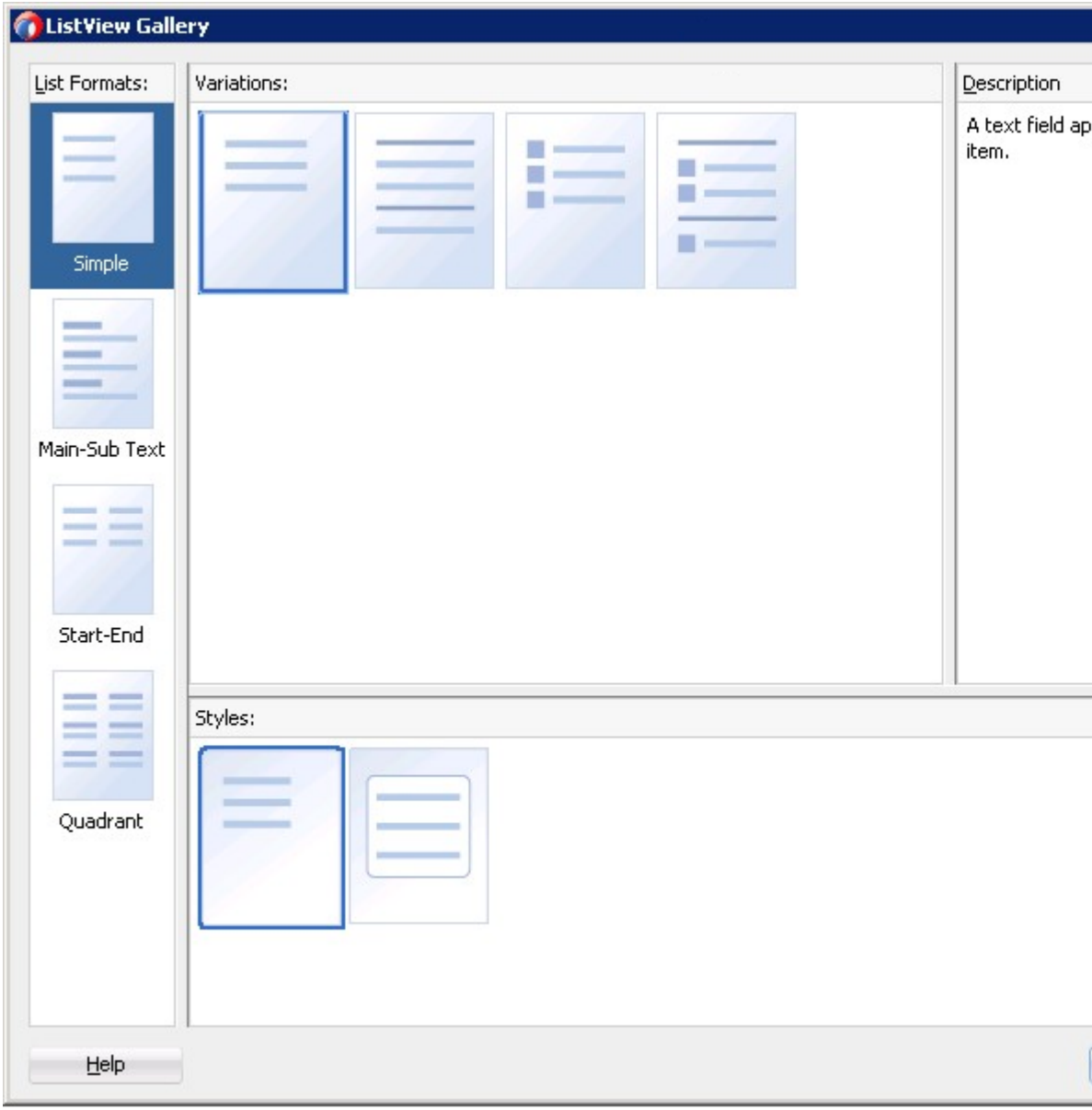


Table 13-5 lists the supported **List Formats** that are displayed in the **ListView Gallery**.

Table 13-5 List Formats

Format	Element Row Values
Simple	• Text
Main-Sub Text	• Main Text • Subordinate Text
Start-End	• Start Text • End Text

Table 13-5 (Cont.) List Formats

Format	Element Row Values
Quadrant	• Upper Start Text • Upper End Text • Lower Start Text • Lower End Text

The **Variations** presented in the **ListView Gallery** (see Figure 13-78) 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 (see Figure 13-79).
- 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** (see Figure 13-78) 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 (see Figure 13-79). 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"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport"
              id="listView2">
   <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 will include a brief description of the main style, followed by the details of the selected variation. Four list formats with four variations on each provide sixteen unique descriptions detailed in Table 13-6.

Table 13-6 List View Formats and Variations

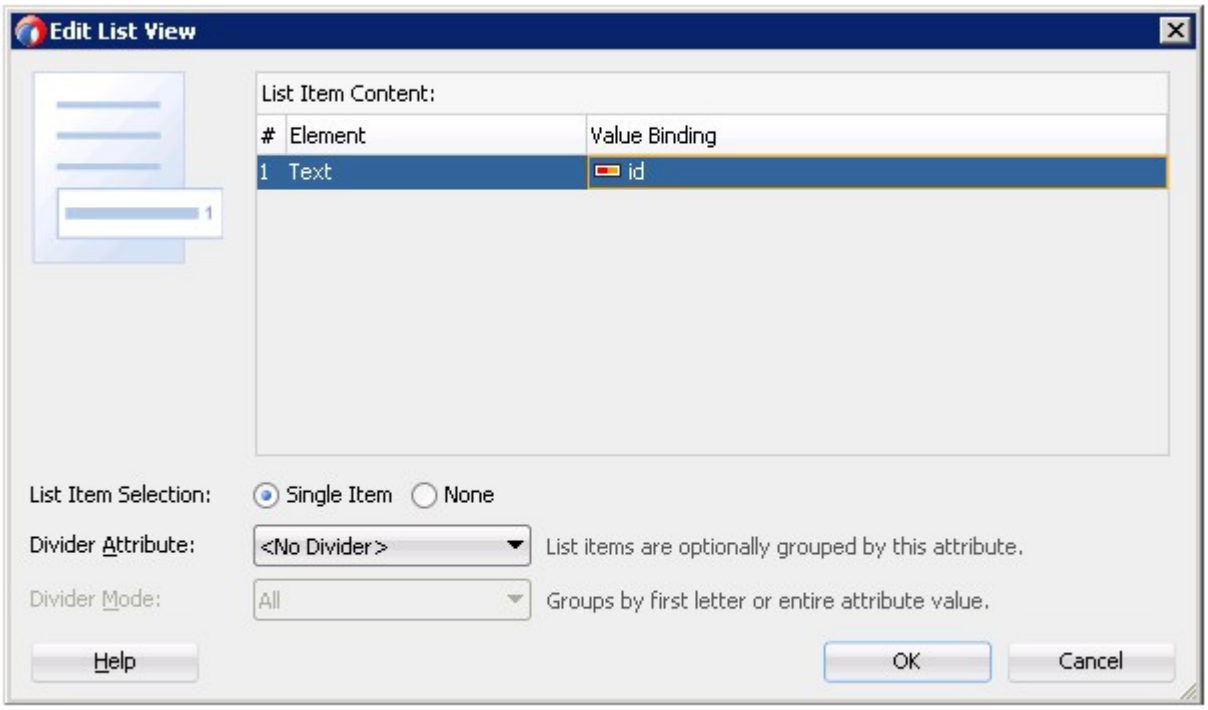
List Format	Variation	Description
Simple	Basic	A text field appears at the start side of the list item."
Simple	Dividers	A text field appears at the start side of the list item, with items grouped by dividers."
Simple	Images	A text field appears at the start side of the list item, following a leading image.

Table 13-6 (Cont.) List View Formats and Variations

List Format	Variation	Description
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 make your selection from the **ListView Gallery** and click **OK**, the **Edit List View** wizard is invoked that lets you create the contents of a **List Item** by mapping binding attributes to the elements of the selected **List View** format, as Figure 13-79 shows.

Figure 13-79 Edit Dialog for MAF AMX List View



When completing the dialog that Figure 13-79 shows, consider the following:

- The image at the start reflects the main content elements from the selected **List View** format and provides a mapping from the schematic representation to the named elements in the underlying table.
- The read-only cells in the **Element** column derive from the selected **List View** format.
- The editable cells in the **Value Binding** column are based on the data control node that was dropped.
- The **List Item** is generated as either an **Output Text** or **Image** component, depending on whichever is appropriate for the particular element.
- Since the number of elements (rows) is predetermined by the selected **List View** format, rows cannot be added or removed.
- The order of elements cannot be modified.
- The **List Item Selection** indicates the selection mode, which can be either a single item selection (default) or no selection. The `showLinkIcon` attribute of the List View is updated based on the selection mode: if the selection mode is set to **None**, `showLinkIcon` attribute is set to `false`; otherwise `showLinkIcon` attribute is not modified (for example, defaulted to `true`).

  The following attributes of the `listView` enable the functioning of the selection mode:

  – `selectionListener`: defines a method reference to a selection listener.

  – `selectedRowKeys`: indicates the selection state for this component.

If the **Single Item** option is chosen, the **Edit List View** dialog automatically sets these attributes as follows:

– `selectionListener` is set to `"bindings.treebinding.collectionModel.makeCurrent"`

– `selectedRowKeys` is set to `"bindings.treebinding.collectionModel.selectedRow"`

The `selectionListener` attribute has the **Edit** option available from the **Properties** window which allows you to create a managed bean class, as well as an appropriate managed bean method, similar to any other listener attributes (see Figure 13-80 and Figure 13-81).

Figure 13-80 Editing Selection Listener Attribute

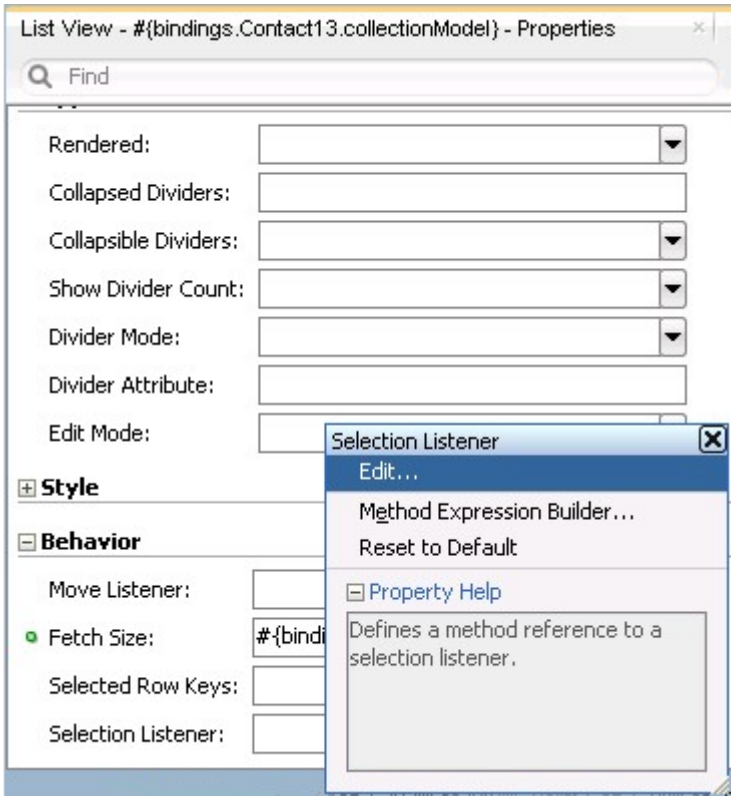


Figure 13-81 Edit Selection Listener Dialog

The following example shows the selection-related attributes of the `listView` element in the MAF AMX file. This declaration is generated when **Single Item** is chosen in the **Edit List View** dialog (see Figure 13-79.

```
<amx:listView id="listView1"
              var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport"
              selectionListener=
                  "#{bindings.employees.collectionModel.makeCurrent}"
              selectedRowKeys=
                  "#{bindings.employees.collectionModel.selectedRow}">
   <amx:listItem id="listItem1">
      ...
   </amx:listItem>
</amx:listView>
```

If **None** was chosen as the **List Item Selection** option in the **Edit List View** dialog, then the `selectionListener` and `selectedRowKeys` attributes are not set as they do not have default values and do not appear in the MAF AMX file. At the same time, the List Item's `showLinkIcon` attribute is set to `false`. The following example demonstrates omitting selection attributes in a List View.

```
<amx:listView id="listView1"
              var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport">
   <amx:listItem id="listItem1" showLinkIcon="false">
      ...
   </amx:listItem>
</amx:listView>
```

The **List View** selection state is preserved when navigation occurs to or from a MAF AMX page.

Note:

The selected row is respected if there is the same iterator ID across MAF AMX pages. For example, if you drag an `Employees` collection onto a page as a **List View** with `employeesIterator` as its iterator and then add a `Details` page, the selected row will only be respected if the `Details` page's employees iterator has its ID set to `employeesIterator`.

- The default value of the **Divider Attribute** field is No Divider, in which case the **Divider Mode** field is disabled. If you select value other than the default, then you need to specify Divider Mode parameters, as Figure 13-82 and Figure 13-83 show.

Figure 13-82 Specifying Divider Attribute

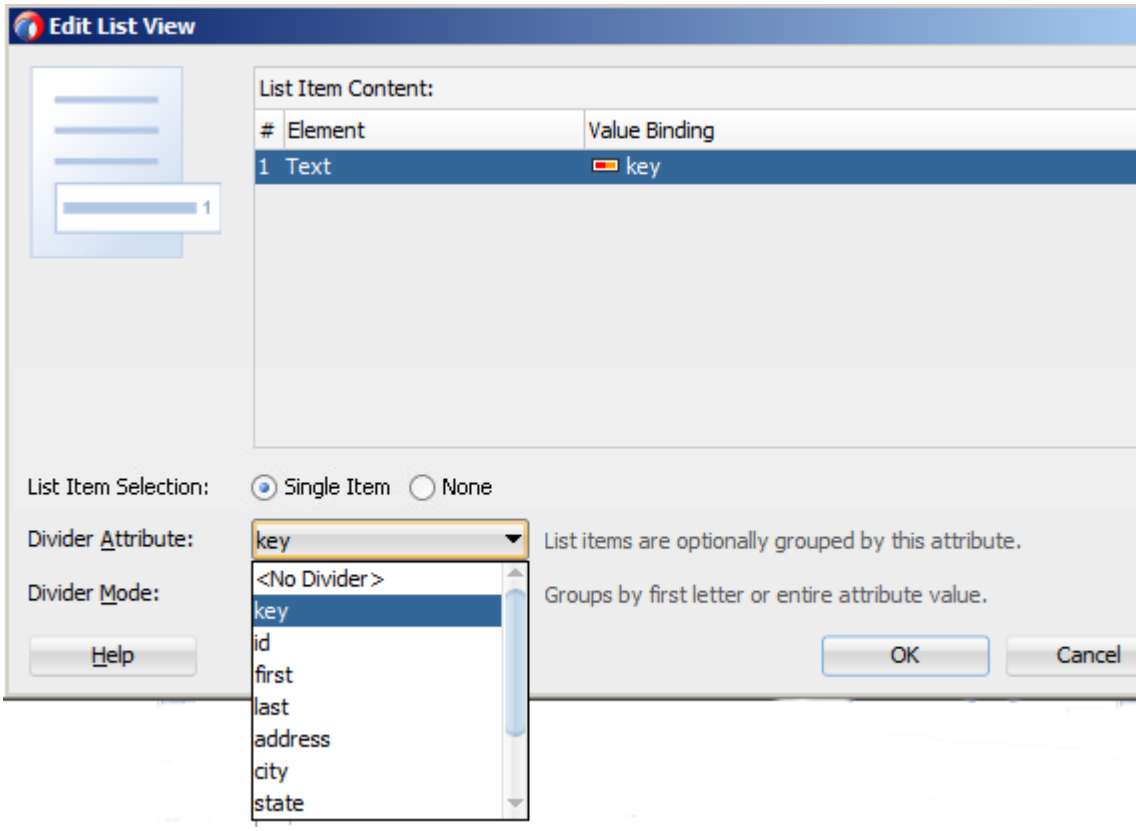
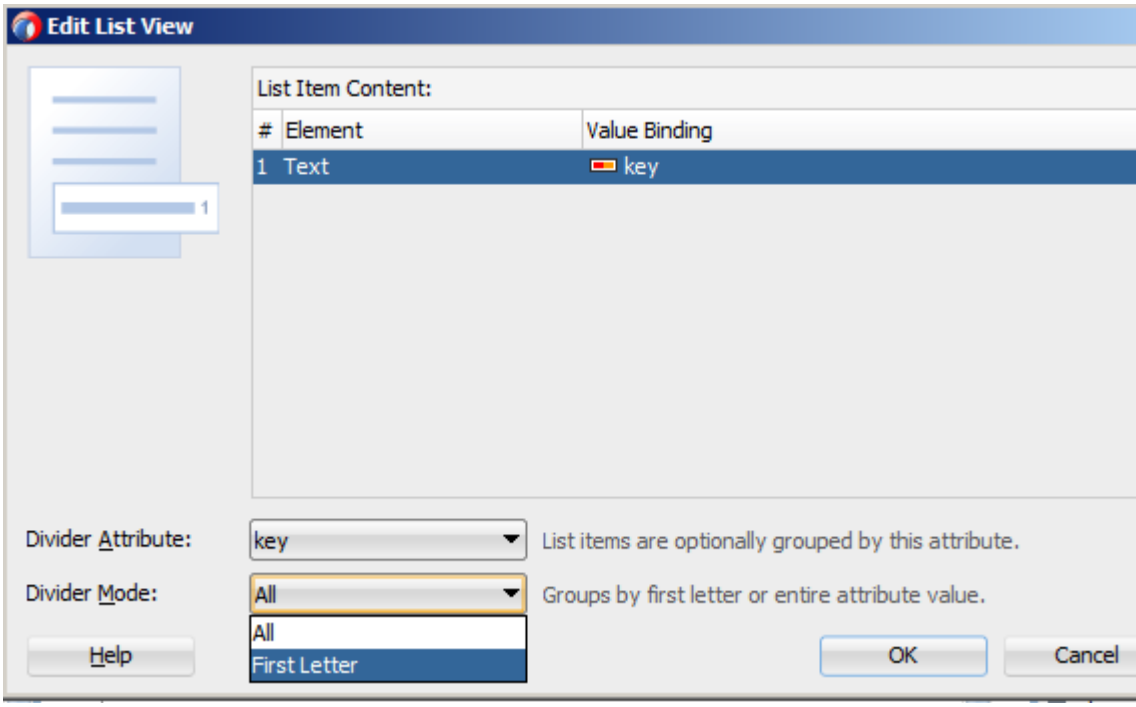


Figure 13-83 Specifying Divider Mode



For more information on **List View** dividers, see How to Use List View and List Item Components.

The following MAF AMX markup and data bindings are generated after you select the appropriate options in the **Edit List View** dialog:

```
<amx:listView id="listView1"
              var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              showMoreStrategy="autoScroll"
              bufferStrategy="viewport"
              dividerAttribute="key"
              dividerMode="firstLetter"
              selectionListener=
                "#{bindings.employees.collectionModel.makeCurrent}"
              selectedRowKeys=
                "#{bindings.employees.collectionModel.selectedRow}">
   <amx:listItem id="listItem1" >
      <amx:outputText value="#{row.key}"
                      styleClass="adfmf-listItem-subtext"
                      id="outputText1"/>
   </amx:listItem>
</amx:listView>
```

For more information about generated bindings, see What You May Need to Know About Generated Bindings.

The following are supported controls for MAF List View:

– MAF Output Text

– MAF Image

13.3.3.4 What You May Need to Know About Generated Bindings

Table 13-7 shows sample bindings that are added to a MAF AMX page when components are dropped.

Table 13-7 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>`

Table 13-7 (Cont.) Sample Data Bindings

Component	Data Bindings

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 13-7 (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>`

13.3.3.5 What You May Need to Know About Generated Drag and Drop Artifacts

The first drag and drop event generates the following directories and files:

- \Application\ViewController\adfmsrc\
  - \mobile
    - \pageDefs
      - **view1PageDef.xml**
      - **view2PageDef.xml**
      - **...**
    - **DataBindings.cpx**

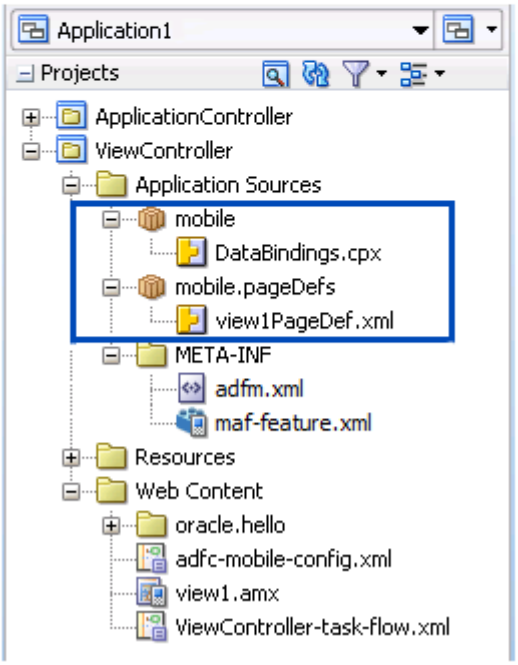


Figure 13-84 shows a sample `DataBindings.cpx` file generated upon drag and drop.

Figure 13-84 DataBindings.cpx File in Source View

```
<?xml version="1.0" encoding="UTF-8" ?>
<Application xmlns="http://xmlns.oracle.com/adfm/application"
             version="12.1.1.60.73" id="DataBindings"
             SeparateXMLFiles="false" Package="mobile" ClientType="Generic">
  <pageMap>
    <page path="/view1.amx" usageId="mobile_view1PageDef"/>
  </pageMap>
  <pageDefinitionUsages>
    <page id="mobile_view1PageDef" path="mobile.pageDefs.view1PageDef"/>
  </pageDefinitionUsages>
  <dataControlUsages>
    <dc id="DeviceDataControl" path="model.DeviceDataControl"/>
  </dataControlUsages>
</Application>
```

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. A 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.

JDeveloper 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, JDeveloper adds an entry for the first page or task flow activity. Each subsequent time you use the Data Controls window, JDeveloper adds an entry to the `DataBindings.cpx` for that page or activity, if one does not already exist.

Once JDeveloper creates a `DataBindings.cpx` file, you can open it in the **Source** view (see Figure 13-84) or the **Overview** 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 **Overview** 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 JDeveloper 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 13-85 shows a sample `PageDef` file generated upon drag and drop.

Figure 13-85 PageDef File

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" version="12.1.1.60.7
                Package="mobile.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <methodIterator Binds="GetContacts.result" DataControl="DeviceDataControl" Ra
                    BeanClass="oracle.adfmf.model.datacontrols.device.Contact" id
  </executables>
  <bindings>
    <methodAction id="GetContacts" RequiresUpdateModel="true" Action="invokeMetho
                  IsViewObjectMethod="false" DataControl="DeviceDataControl"
                  InstanceName="data.DeviceDataControl.dataProvider"
                  ReturnName="data.DeviceDataControl.methodResults.
                              GetContacts_DeviceDataControl_dataProvider_GetConta
    <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, JDeveloper 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, JDeveloper 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 Accessing the Page Definition File. When you open a page definition file in the **Overview** editor, you can view and configure bindings, contextual events, and parameters for a 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:

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, JDeveloper adds a number to the default binding object names to keep them unique.

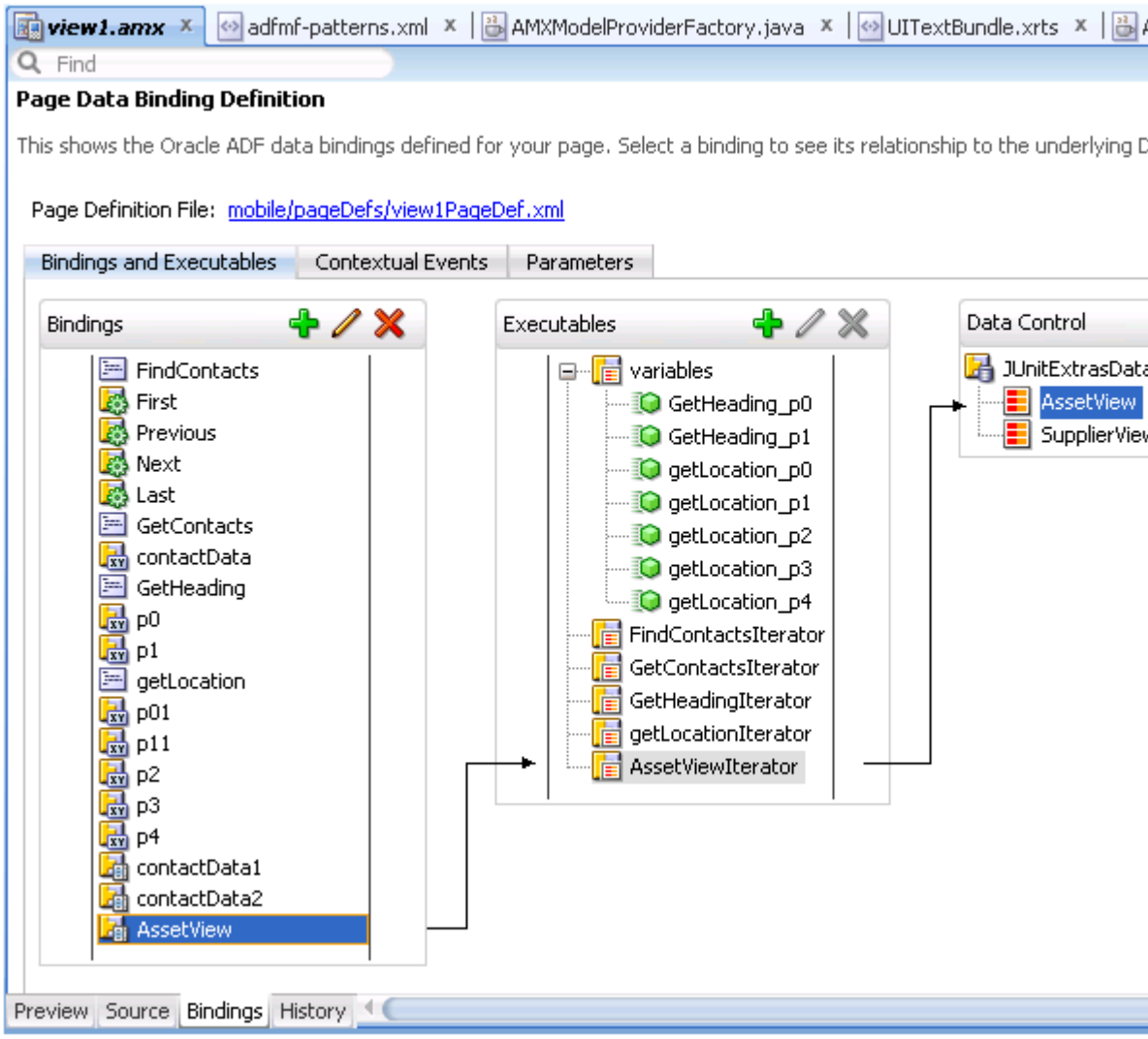
- Contextual Events: you can create contextual events to which artifacts in a MAF AMX application feature can subscribe.
- Parameters: parameter binding objects declare the parameters that the page evaluates at the beginning of a request. You can define the value of a parameter in the page definition file using static values or EL expressions that assign a static value.

When you click an item in the **Overview** 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.

13.3.3.6 Using the MAF AMX Editor Bindings Tab

JDeveloper's Bindings tab (see Figure 13-86) 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 13-86 Bindings Tab



13.3.3.7 What You May Need to Know About Removal of Unused Bindings

When you delete or cut a 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 13-87 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 13-87 Deleting Bound Components from Page

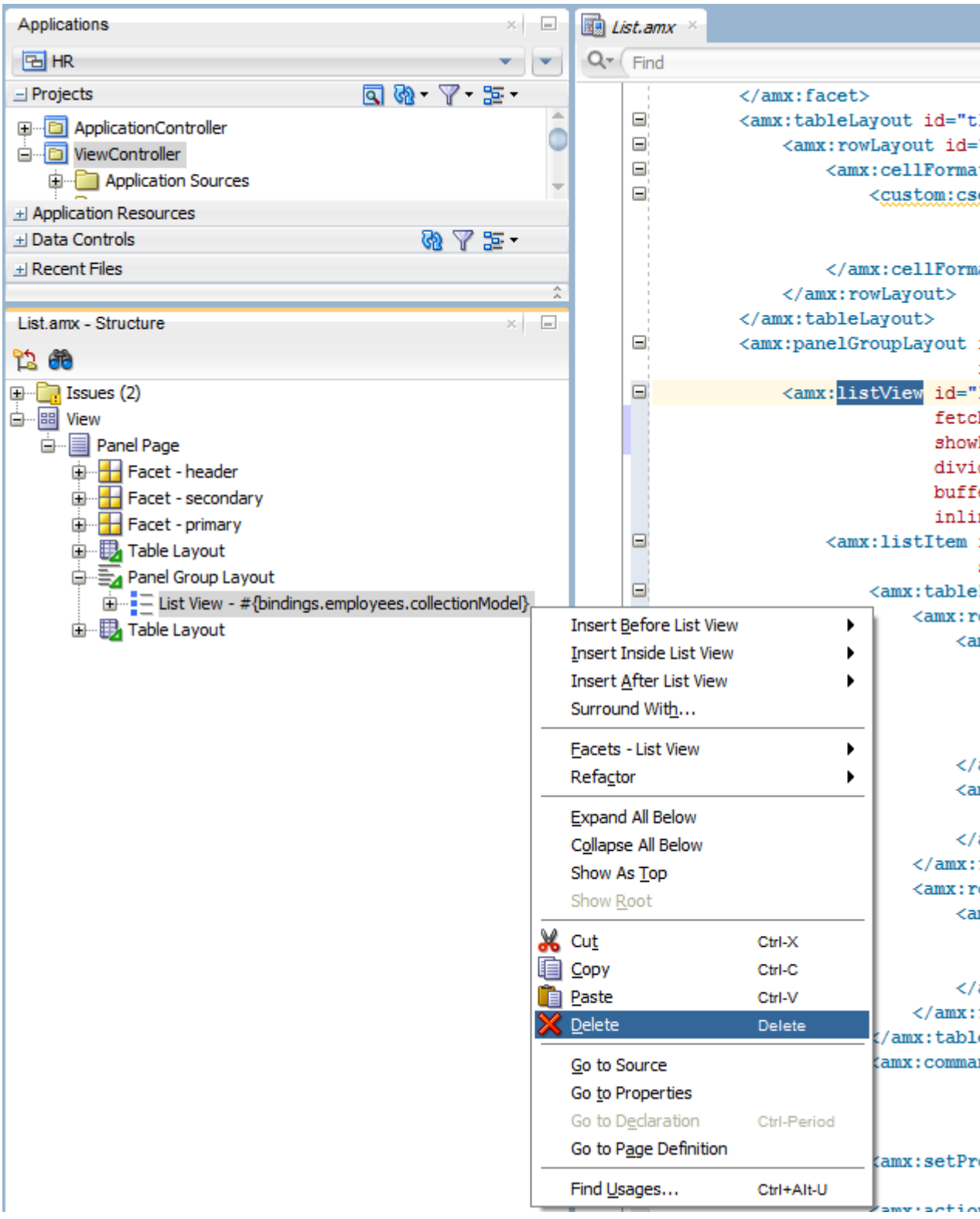
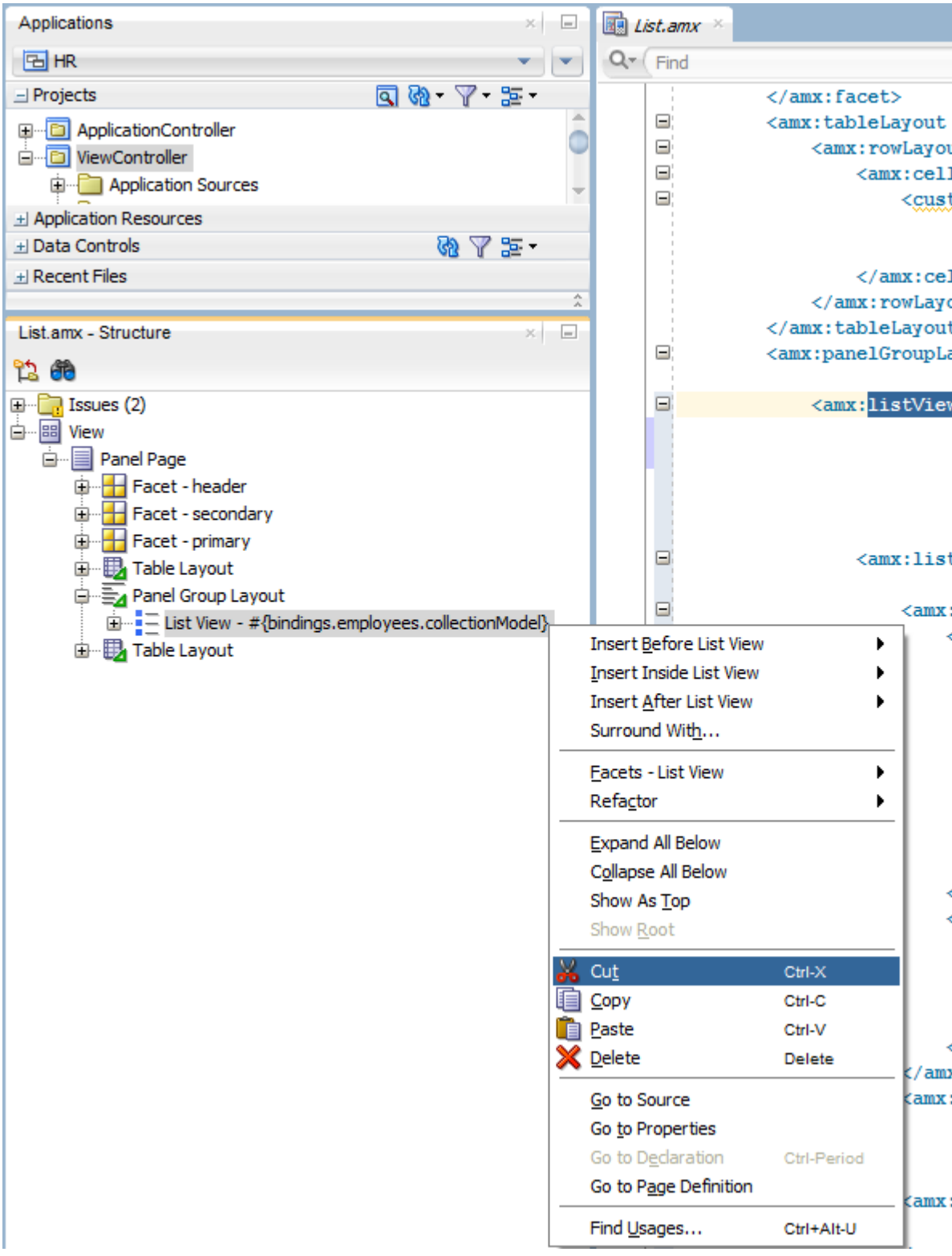


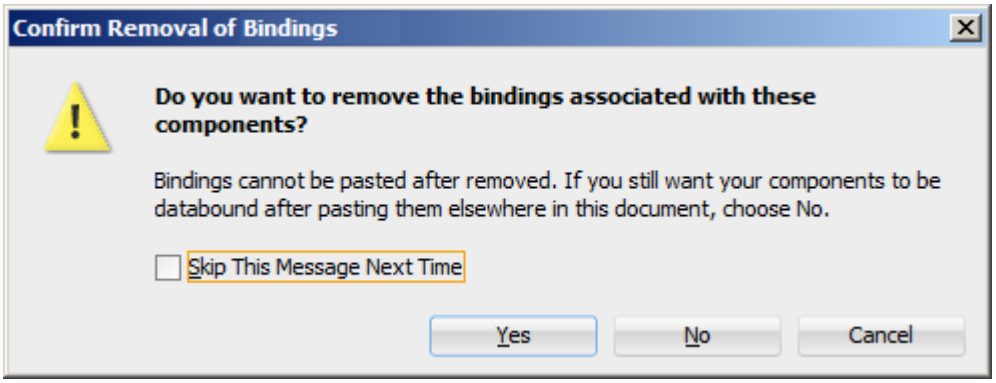
Figure 13-88 demonstrates the removal of the **List View** component by cutting it from the page.

Figure 13-88 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 13-89.

Figure 13-89 Confirm Removal of Bindings Dialog



13.3.4 What You May Need to Know About the Server Communication

The security architecture used by MAF guarantees that the browser hosting a 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 JavaScript from within MAF AMX UI components. Communication with the server must occur from the embedded Java code layer.

14

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:

- Introduction to Creating the User Interface for MAF AMX Pages
- Designing the Page Layout
- Creating and Using UI Components
- Creating Custom UI Components
- Enabling Gestures
- Implementing Application Shortcuts for Use on iOS Devices with 3D Touch Support
- Providing Data Visualization
- Styling UI Components
- Understanding MAF Support for Accessibility
- Validating Input
- Using Event Listeners

14.1 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 on the platforms that MAF supports (Android, iOS, and the Universal Windows Platform).

MAF AMX adheres to the typical JDeveloper development experience by allowing you to drag UI components and operations onto a Source editor or Structure window from either the Components window or the Data Controls 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 JDeveloper 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:

- Creating MAF AMX Pages
- Using Bindings and Creating Data Controls in MAF AMX

14.2 Designing the Page Layout

MAF AMX provides layout components 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 (listed in Table 14-1). Some of these components provide geometry management functionality, such as the capability to stretch when placed inside a component that stretches.

Table 14-1 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 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 How to Use a Panel Page Component. For more information about MAF AMX files, see 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 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 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 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 How to Use a Panel Group Layout Component.

Table 14-1 (Cont.) MAF AMX Page Management, Layout, and Spacing Components

Component	Type	Description
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 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 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 How to Use a Panel Stretch Layout Component.
Popup	Secondary Window	Creates a `popup` element in a MAF AMX file. Displays a popup window. For more information, see How to Use a Popup Component.
Panel Splitter	Interactive Page Layout Container	Creates a `panelSplitter` element in a MAF AMX file. Allows to display multiple content areas that may be controlled by a left-side navigation pane. For more information, see How to Use a Panel Splitter Component.
Panel Item	Interactive Page Layout Component	Creates a `panelItem` element in a MAF AMX file. Represents the content area of a Panel Splitter. For more information, see How to Use a Panel Splitter Component.
Deck	Page Layout Component	Creates a `deck` element in a MAF AMX file. Shows one of its child components at a time. For more information, see 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 How to Use a Flex Layout Component.
Spacer	Page Layout Component	Creates a `spacer` element in a MAF AMX file. Creates an area of blank space represented by a `spacer` element in a MAF AMX file. For more information, see 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 How to Use a Table Layout Component.

Table 14-1 (Cont.) MAF AMX Page Management, Layout, and Spacing Components

Component	Type	Description
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 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 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 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 How to Use an Accessory Layout Component.

You add a layout component by dragging and dropping it onto a MAF AMX page from the **Components** window (see How to Add UI Components to a MAF AMX Page). Then you use the **Properties** window to set the component's attributes (see Configuring UI Components). 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.

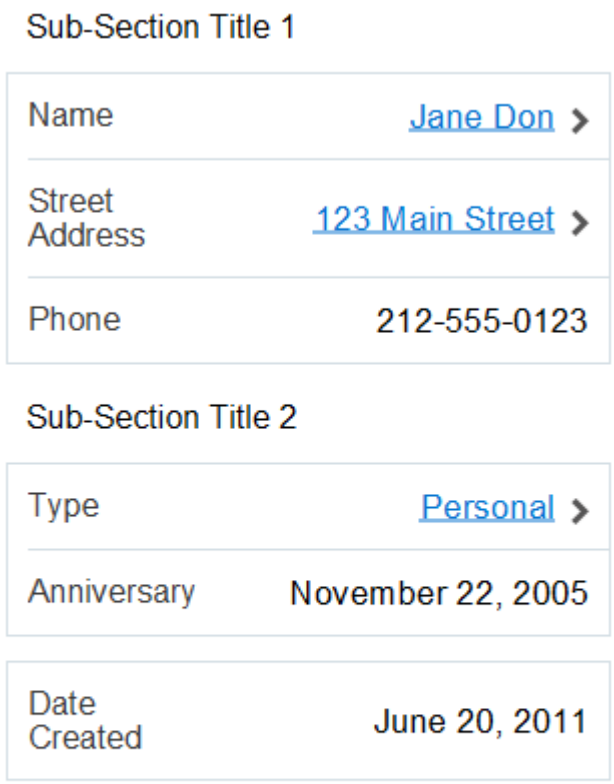
Note:

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 14-1 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 `Web Content/css` directory of your ViewController project, with default CSS provided by MAF. For more information, see How to Use Component Attributes to Define Style.

The user interface created using MAF AMX displays correctly in both the left-to-right (LTR) and right-to-left (RTL) 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 How to Use a Popup Component), Panel Item, and Panel Splitter (see How to Use a Panel Splitter Component) can be configured to enable specific RTL behavior. For more information about the RTL configuration of MAF AMX pages, see Enabling Gestures and How to Specify the Page Transition Style.

Note:

The right-to-left text direction is not supported on Android prior to version 4.2.

A MAF sample application called UILayoutDemo demonstrates how to use layout components in conjunction with such MAF AMX UI components as a Button, to achieve some of the typical layouts that follow common patterns. In addition, this sample application shows how to work with styles to adjust the page layout to a specific pattern. The UILayoutDemo application is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.2.1 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*.

14.2.2 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, JDeveloper 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 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*.

14.2.3 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 **Components** window:

1. In the **Components** window, select **MAF AMX > Layout**, and then 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.
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>
```

14.2.3.1 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 What You May Need to Know About Skinning), scrolling of the **Panel Group Layout**, **Panel Form Layout** (see How to Use a Panel Form Layout Component), and **Table Layout** (see 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.

For more information, see What You May Need to Know About Memory Consumption by MAF AMX UI Components.

14.2.4 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 **Components** window, select **MAF AMX > Layout**, and then drag and drop a Panel Form Layout component to the MAF AMX page.
2. In the **Properties** window, 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>
```

14.2.5 How to Use a Panel Stretch Layout Component

The Panel Stretch Layout (`panelStretchLayout`) component manages three child Facet components: top, bottom, and center, as shown in the following 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 **Components** window, select **MAF AMX > Layout**, and then 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*.

14.2.6 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 **Components** window, select **MAF AMX > Layout**, and then drag and drop a Panel Label And Message component into a Panel Group Layout component.
2. In the **Properties** window, set the component's attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

The following example 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>
```

14.2.7 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 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 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 following 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 14-2).

```
<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 14-2 lists predefined Facet types that you can use with specific parent components.

Table 14-2 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`
Panel Splitter (`panelSplitter`)	`navigator`
Panel Stretch Layout (`panelStretchLayout`)	`top`, `center`, `bottom`

Table 14-2 (Cont.) Facet Types and Parent Components

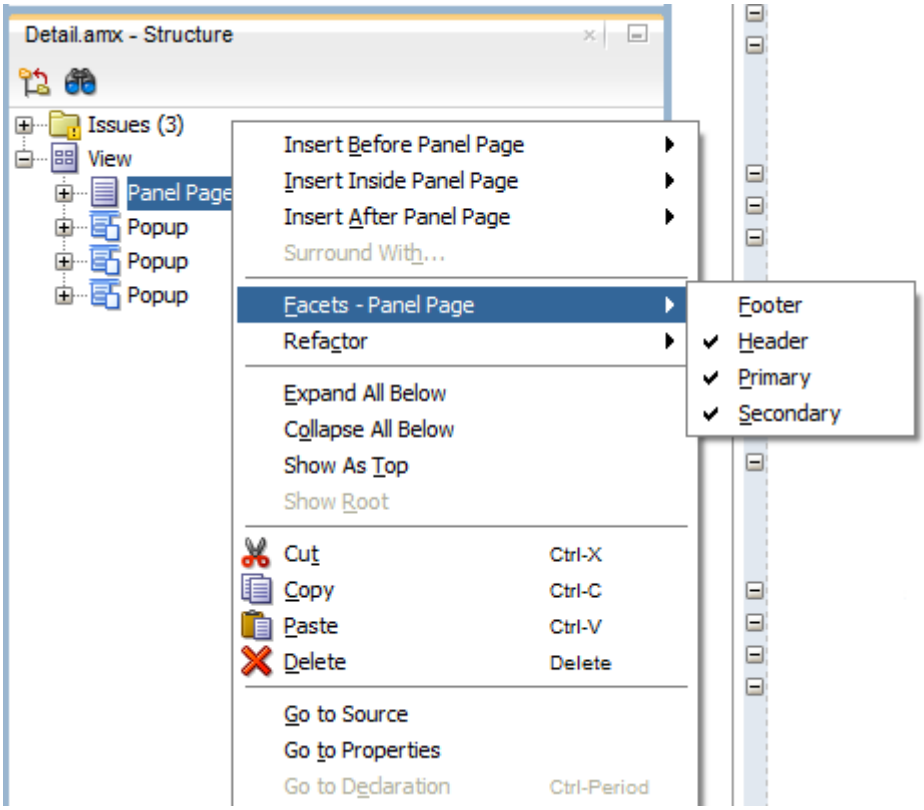
Parent Component	Facet Type (name)
Data Visualization Components. For more information, see Providing Data Visualization.	`dataStamp`, `seriesStamp`, `overview`, `rows` (applicable to NBox), `columns` (applicable to NBox), `cells` (applicable to NBox), `icon` (applicable to NBox Node), `indicator` (applicable to NBox Node)

To add the Facet component:

You can use the context menu displayed on the **Structure** window or **Source** editor to add a Facet component as a child of another component. The context menu displays only facets that are valid for your selected parent component. To add a Facet, first select and then right-click the parent component in the **Structure** window or **Source** editor, and then select one of the following:

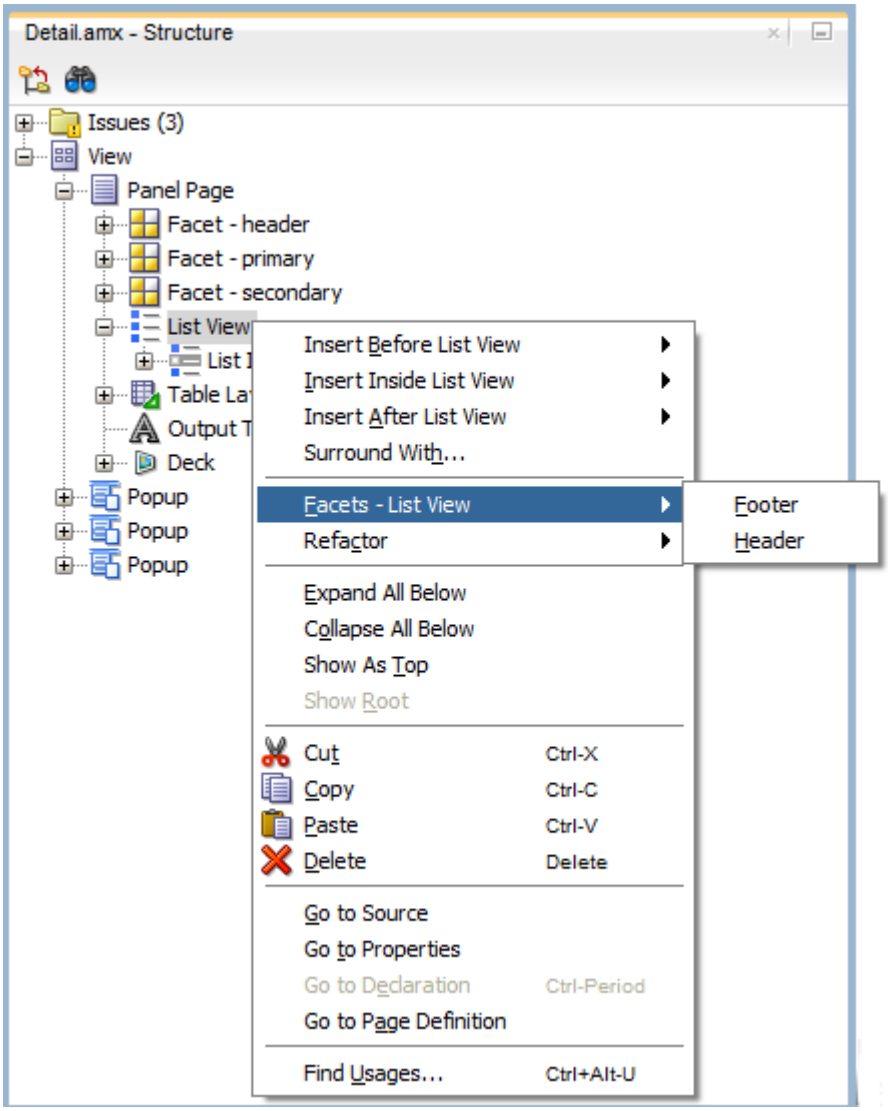
- If the parent component is a **Panel Page**, select **Facets - Panel Page** and then choose the type of Facet from the list, as Figure 14-2 shows.

Figure 14-2 Using Context Menu to Add Facet to Panel Page



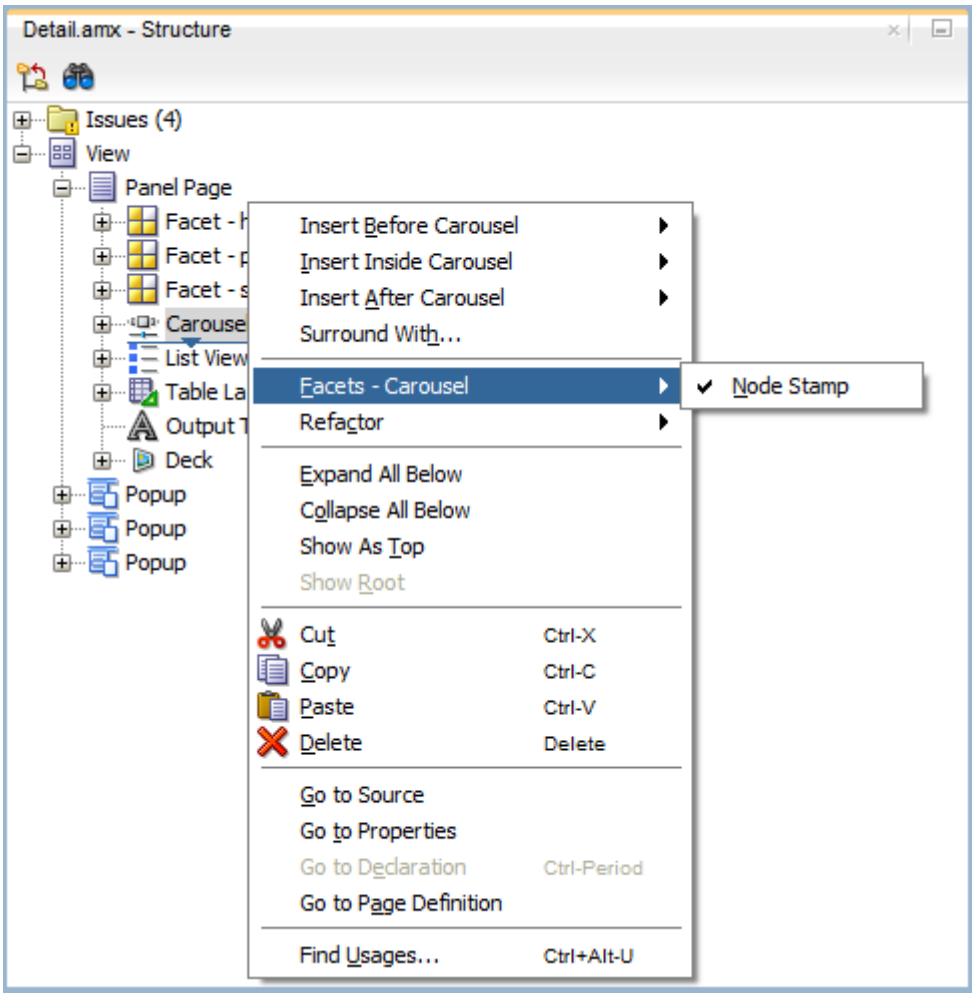
- If the parent component is a **List View**, select **Facets - List View** and then choose the type of Facet from the list, as Figure 14-3 shows.

Figure 14-3 Using Context Menu to Add Facet to List View



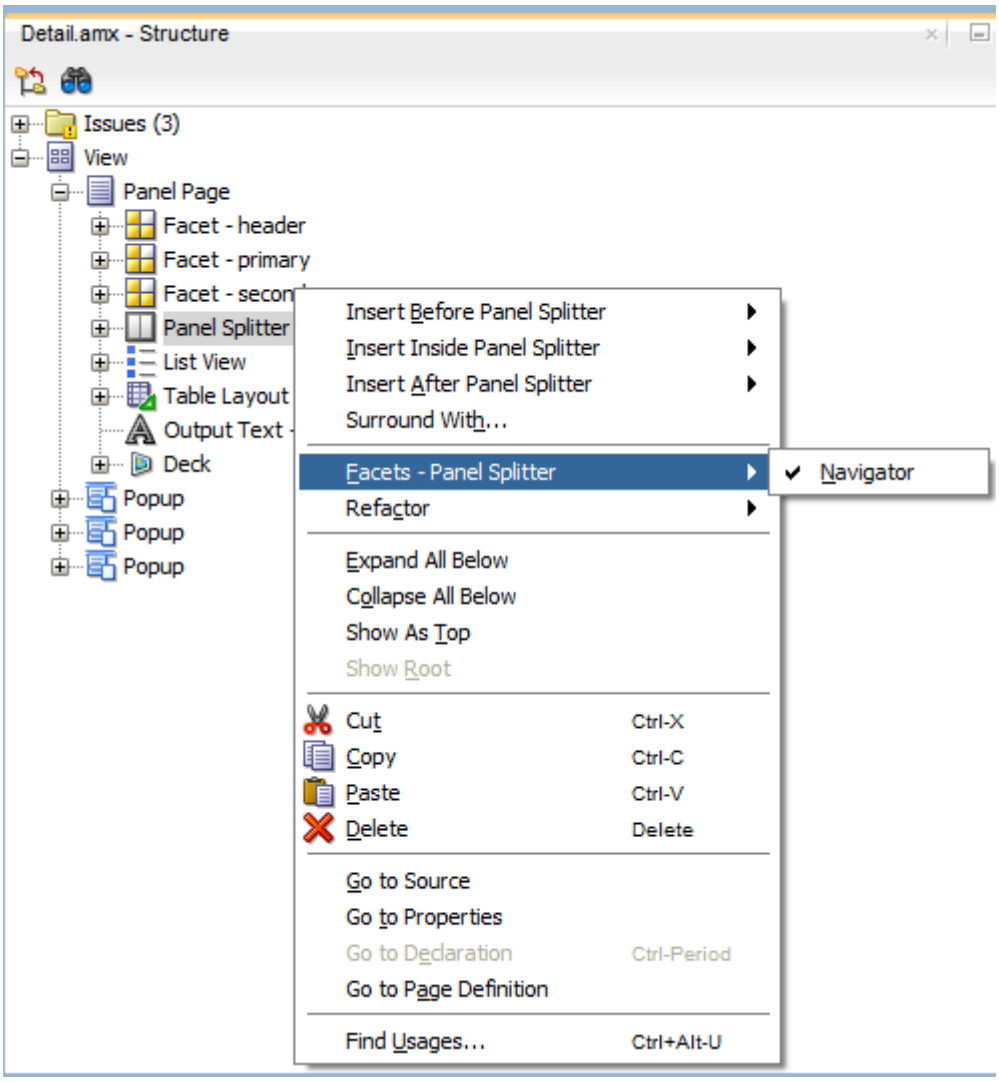
- If the parent component is a Carousel, select **Facets - Carousel > Node Stamp**, as Figure 14-4 shows.

Figure 14-4 Using Context Menu to Add Facet to Carousel



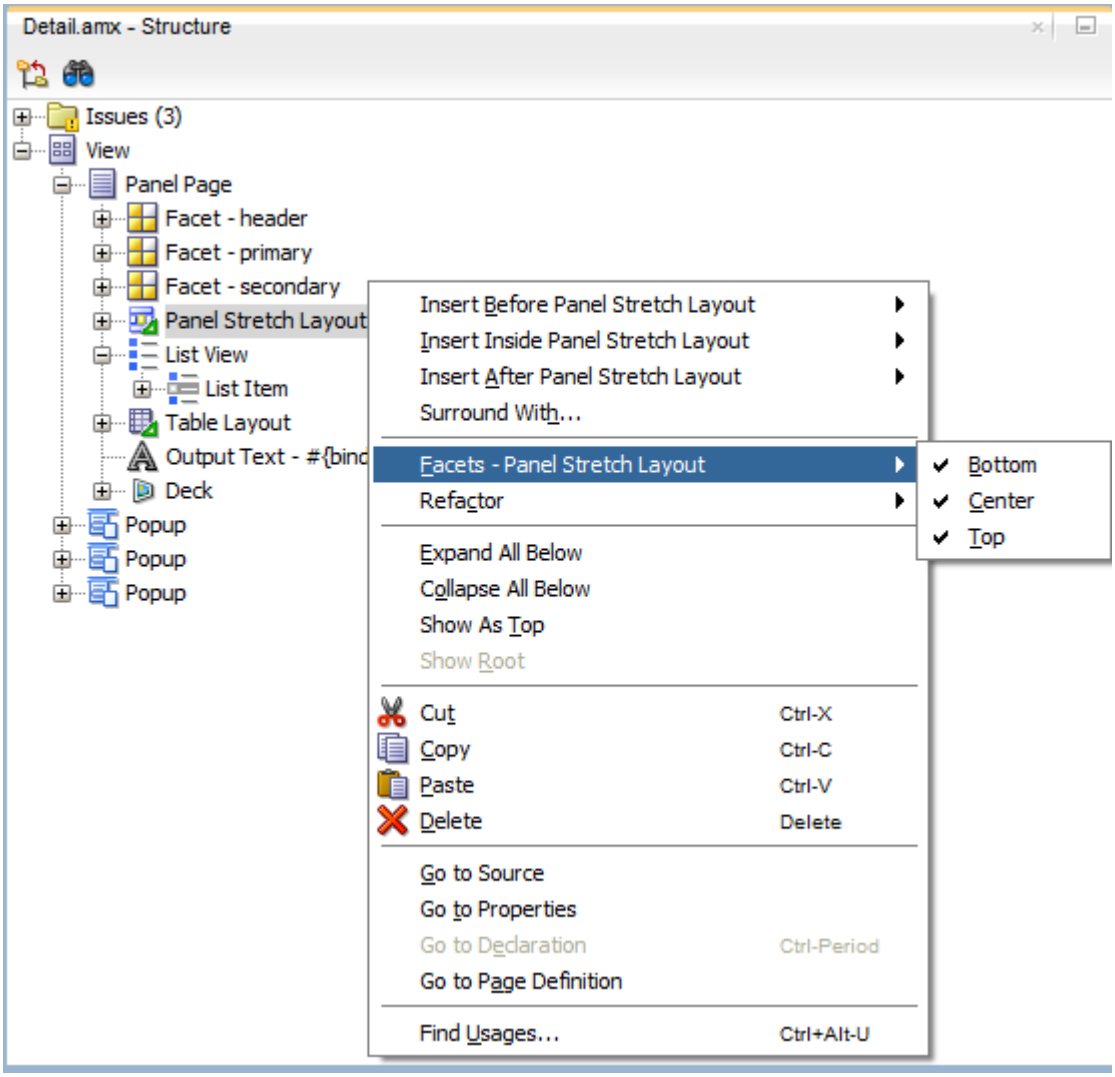
- If the parent component is a Panel Splitter, select **Facets - Panel Splitter > Navigator**, as Figure 14-5 shows.

Figure 14-5 Using Context Menu to Add Facet to Panel Splitter



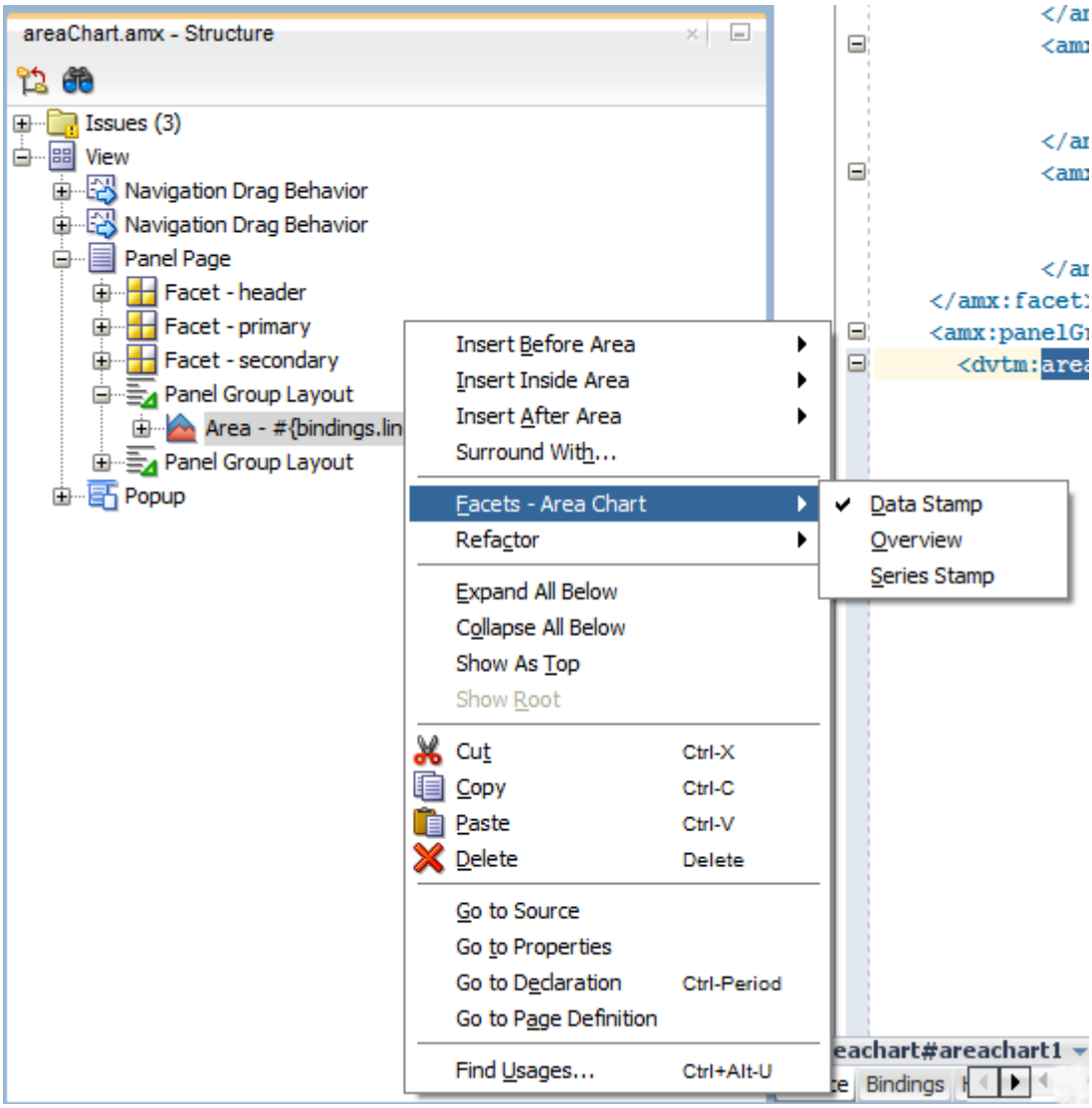
- If the parent component is a Panel Stretch Layout, select **Facets - Panel Stretch Layout** and then choose the type of Facet from the list, as Figure 14-6 shows.

Figure 14-6 Using Context Menu to Add Facet to Panel Stretch Layout



- If the parent component is one of the data visualization components, select **Facets > <MAF AMX Data Visualizations Component Name>** and then choose the type of Facet from the list, as Figure 14-7 shows.

Figure 14-7 Using Context Menu to Add Facet to Data Visualization Component



For more information about data visualization components and their attributes, see Providing Data Visualization.

Alternatively:

1. In the **Components** window, select **MAF AMX > Layout > Core Structure**, and then drag and drop a Facet component into another component listed in Table 14-2.

2. In the **Properties** window, set the component's attributes. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

14.2.8 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 specified using the `type` attribute of the Close Popup Behavior.

  For more information about the Close Popup Behavior component's attributes and their values, see *Tag Reference for Oracle Mobile Application Framework*.

- 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 (see Figure 14-9). There is an Audit rule that is specifically defined to validate these identifiers (see 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 following example shows `popup` as well as its `showPopupBehavior` and `closePopupBehavior` 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:panelGroupLayout id="pgl2" layout="vertical">
         <amx:commandButton id="commandButton3" text="Close Popup">
            <amx:closePopupBehavior type="action" popupId="popup1"/>
         </amx:commandButton>
      </amx:panelGroupLayout>
   </amx:popup>
</amx:view>
```

Popup components can display validation messages when the user input errors occur. For more information, see Validating Input.

To set a Popup Id attribute:

1. Select either the `showPopupBehavior` or `closeopupBehavior` element in the **Source** editor or **Structure** window.
2. Click the down arrow to the right of the **Popup Id** field to make a selection from a list of available Popup components (see Figure 14-8), or click the **Property Menu** icon to the right of the **Popup Id** field to open the **Popup Id** property editor (see Figure 14-9).

Figure 14-8 Selecting Popup Id from List

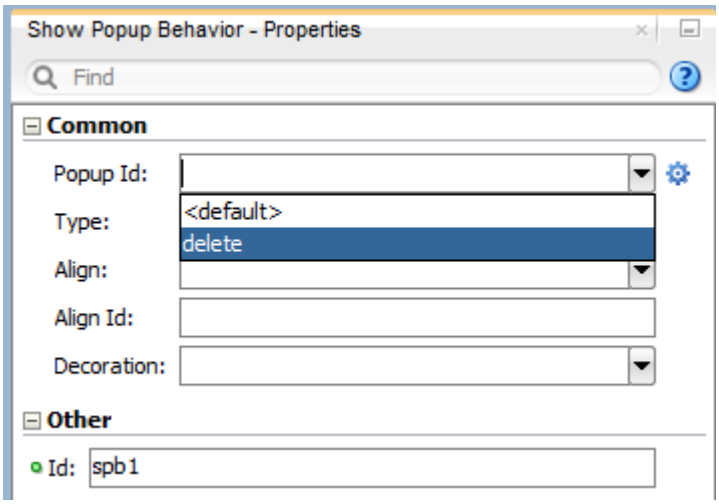
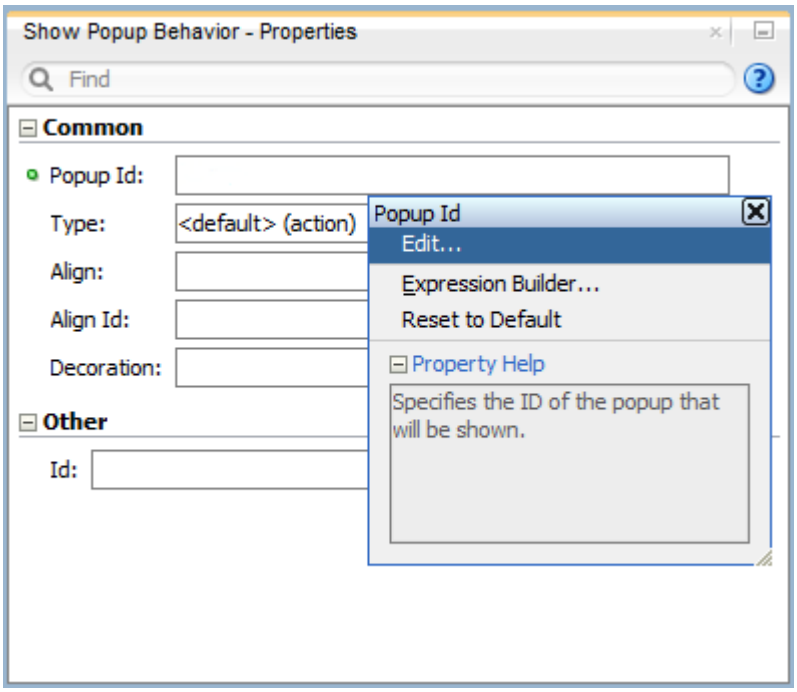
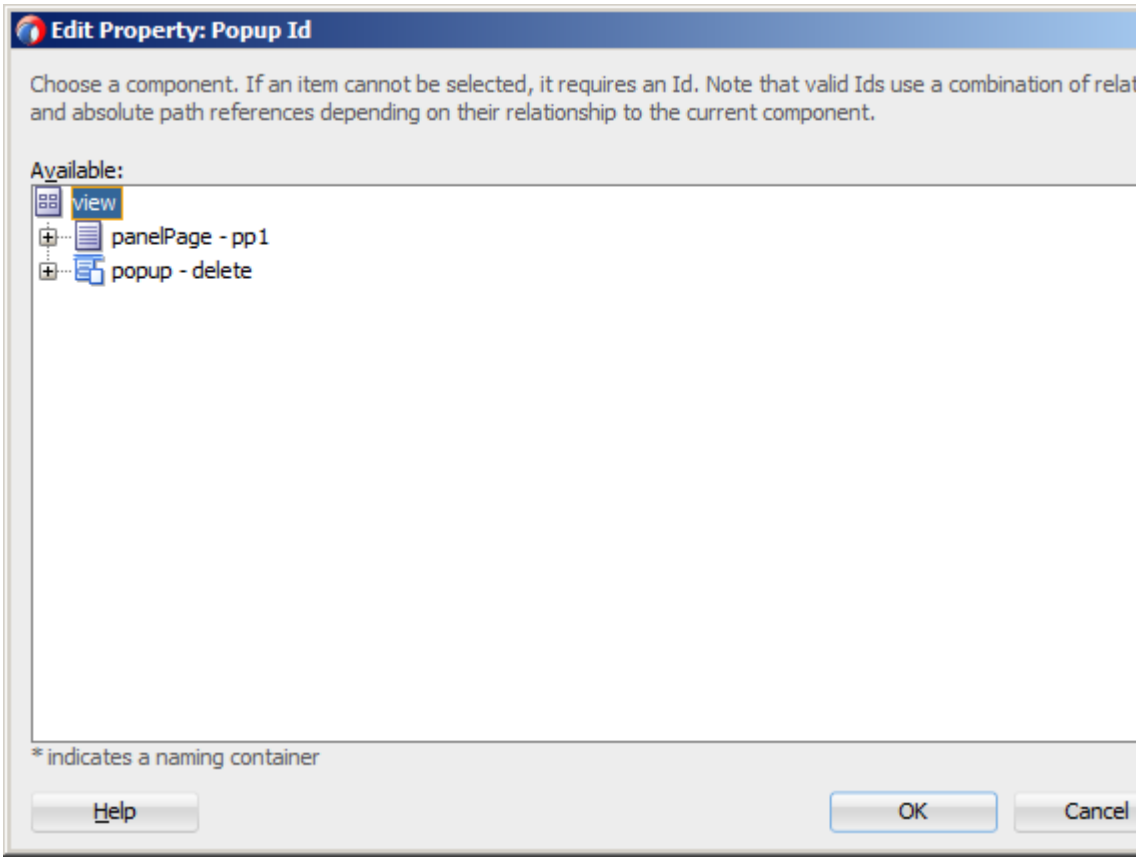


Figure 14-9 Setting Popup Id Attribute



3. If you use the property editor, select **Edit** on the **Popup Id** property editor to open the **Edit Property: Popup Id** dialog that Figure 14-10 shows.

Figure 14-10 Edit Property for Popup Id Dialog

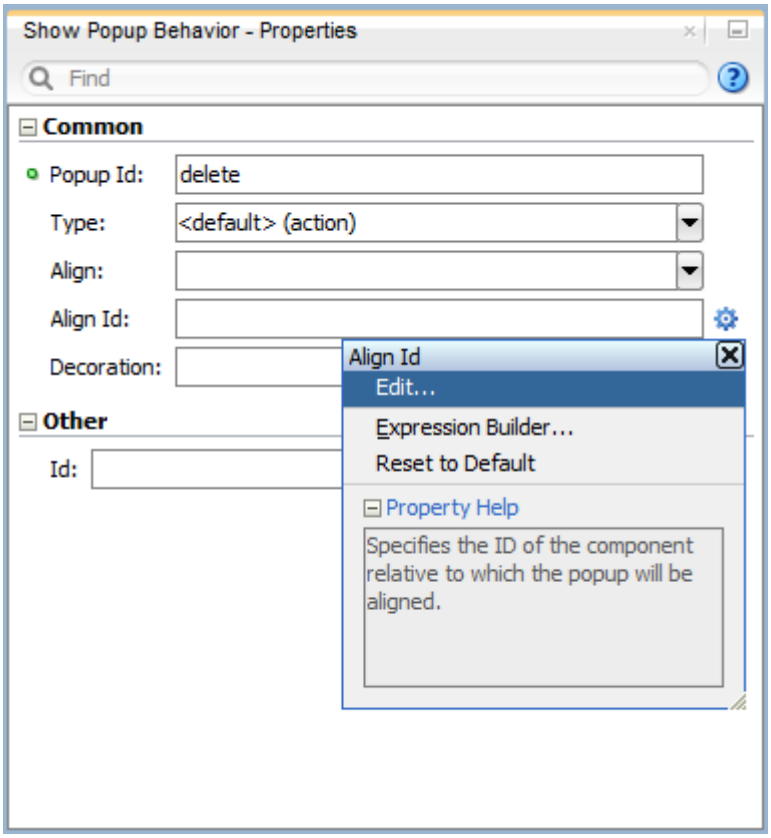


4. Select the Popup component to be displayed or the Popup component to be closed when this **Show Popup Behavior** or **Close Popup Behavior** is invoked.

To set an **Align Id** attribute:

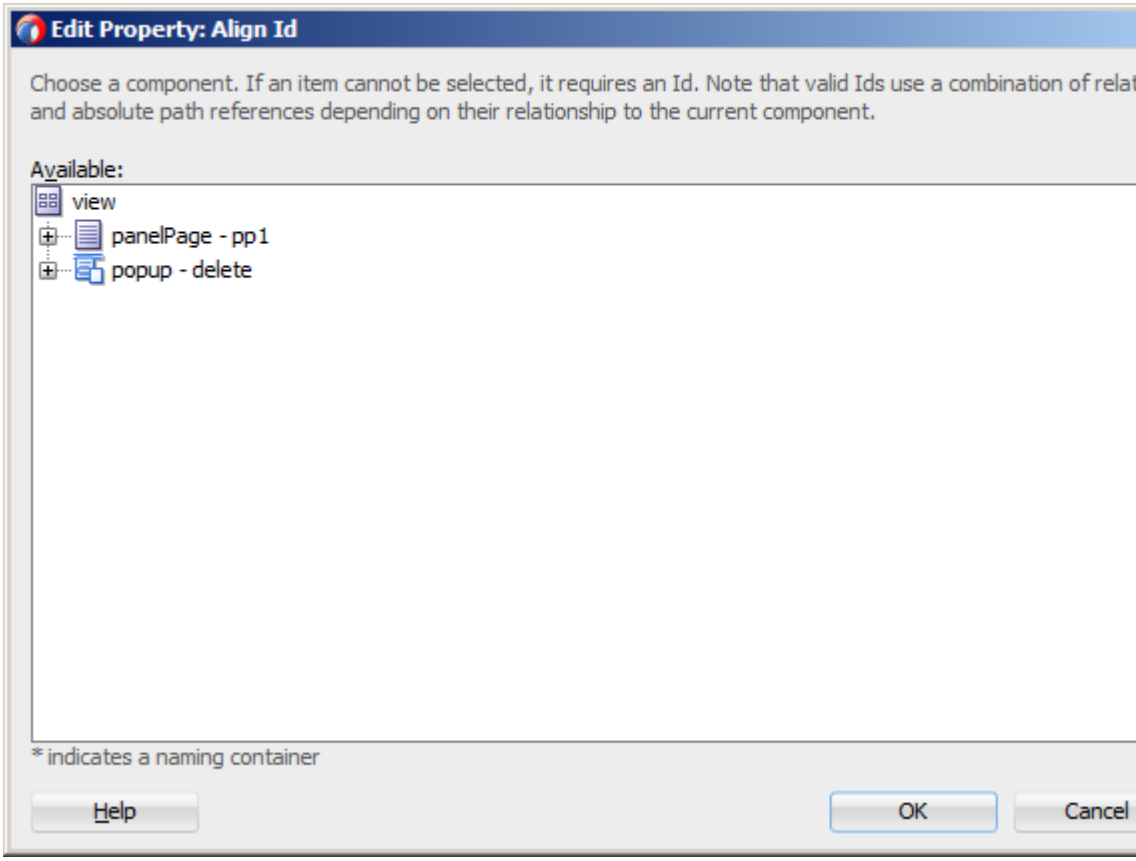
1. Select the `showPopupBehavior` element in the **Source** editor or **Structure** window.
2. Click the **Property Menu** icon to the right of the **Align Id** field to open the **Align Id** property editor, as Figure 14-11 shows.

Figure 14-11 Setting Align Id Attribute



3. Select **Edit** on the **Align Id** property editor to open the **Edit Property: Align Id** dialog that Figure 14-12 shows.

Figure 14-12 Edit Property for Align Id Dialog



4. Select the parent component of the **Show Popup Behavior** operation.

When developing for both iOS platform and Android 4.2 or later platform, you can configure the Popup to accommodate the right-to-left (RTL) language environment by setting its `animation` attribute to either `slideStart` or `slideEnd`.

By setting the `animation` attribute to `zoom`, you can enable the Popup to zoom in and out of its originating component.

A MAF sample application called UILayoutDemo 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.2.9 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 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 at 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 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 multi-content 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 renders. You can configure the navigator display and the navigator's disclosure state using the `navigatorDisplay` and `navigatorDisclosureState` attributes.

You can configure the Panel Splitter and Panel Item to accommodate the right-to-left (RTL) 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.

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>
```

The CompGallery sample application, described in MAF Sample Applications, provides an implementation of the Panel Splitter component in the Layout category. For more information about the Panel Splitter component's attributes, see *Tag Reference for Oracle Mobile Application Framework*.

14.2.10 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, select **MAF AMX > Layout**, and then 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 and its children 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..."/>
```

14.2.11 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, select **MAF AMX > Layout**, and then 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 following example 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>
```

14.2.12 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, select **MAF AMX > Layout**, and then 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*.

The 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*.

14.2.13 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, select **MAF AMX > Layout**, and then drag and drop an Accessory Layout onto the MAF AMX page.
2. Optionally, add child Facet components and populate them with other MAF AMX UI components.
3. 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 `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 application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

14.2.14 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 backward.

To add the Deck component:

1. In the **Components** window, select **MAF AMX > Layout**, and then 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 `deck` element and its children defined in a MAF AMX file. The Deck component's `displayedChild` attribute is 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*.

14.2.15 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, navigate to **MAF AMX > Layout**, and then 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>
```

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

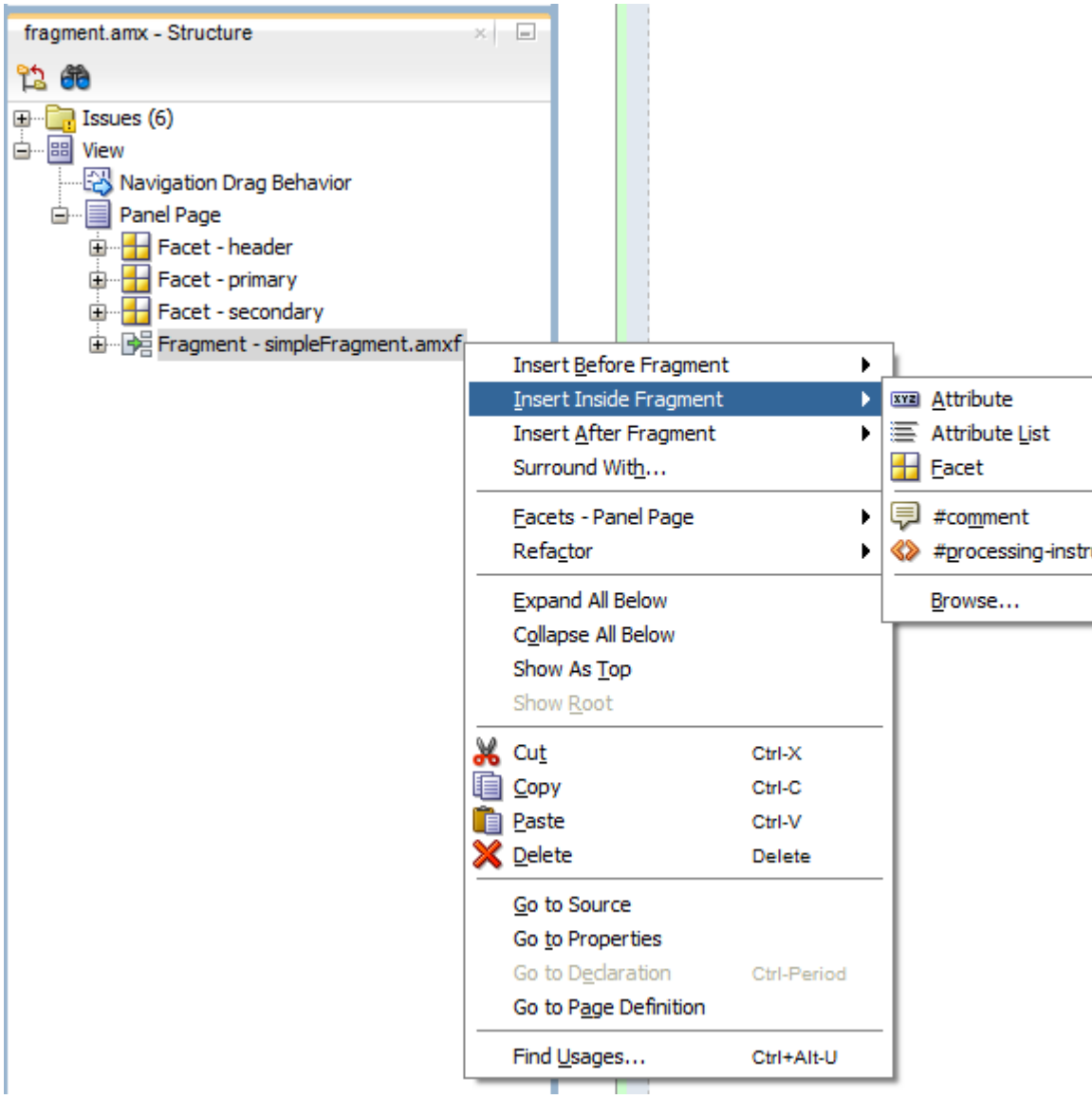
14.2.16 How to Use the Fragment Component

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 Sharing the Page Contents.

To add a 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**.

Figure 14-13 Populating Fragment



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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.3 Creating and Using UI Components

MAF provides a range of UI components that you can use when developing MAF AMX application features.

The following UI components can be used:

- Input Text (see How to Use the Input Text Component)
- Input Number Slider (see How to Use the Input Number Slider Component)
- Input Date (see How to Use the Input Date Component)
- Output Text (see How to Use the Output Text Component)
- Button (see How to Use Buttons)
- Link (see How to Use Links)

- Image (see How to Display Images)
- Checkbox (see How to Use the Checkbox Component)
- Select Many Checkbox (see How to Use the Select Many Checkbox Component)
- Select Many Choice (see How to Use the Select Many Choice Component)
- Boolean Switch (see How to Use the Boolean Switch Component)
- Choice (see How to Use the Choice Component)
- Select Button (see How to Use the Select Button Component)
- Radio Button (see How to Use the Radio Button Component)
- List View (see How to Use List View and List Item Components)
- Carousel (see How to Use a Carousel Component)
- Film Strip (see How to Use the Film Strip Component)
- Verbatim (see How to Use Verbatim Component)
- Output HTML (see How to Use an Output HTML Component)
- Iterator (see How to Enable Iteration)
- Refresh Container (see 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:

- Action Listener (see How to Use the Action Listener)
- Set Property Listener (see How to Use the Set Property Listener)
- Client Listener (see How to Use the Client Listener)
- Convert Date Time (see How to Convert Date and Time Values)
- Convert Number (see How to Convert Numeric Values)
- Navigation Drag Behavior (see How to Enable Drag Navigation)
- Loading Indicator Behavior (see How to Use the Loading Indicator)
- System Action Behavior (see How to Configure Behavior of the Android System Back Button)

You add a UI component by dragging and dropping it onto a MAF AMX page from the Components window (see How to Add UI Components to a MAF AMX Page). Then you use the Properties window to set the component's attributes (see Configuring UI Components). 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 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 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 Understanding MAF Support for Accessibility.

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 JDeveloper's Preview and the component may look different at runtime.

The user interface created for both the iOS platform and Android 4.2 or later platform using MAF AMX displays correctly in both the left-to-right (LTR) and right-to-left (RTL) language environments. In the latter case, the components originate on the right-hand side of the screen instead of on the left-hand side.

A MAF sample application called CompGallery demonstrates how to create and configure MAF AMX UI components. Another sample application called UILayoutDemo shows how to lay out components on a MAF AMX page. These sample applications are located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.3.1 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}"
```

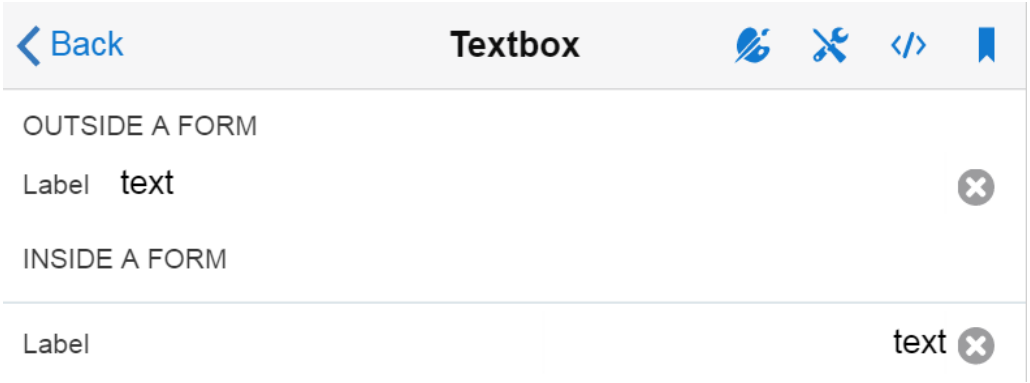
```
                simple="true"
                rows="4" />
```

- Display clear text icon:

```
<amx:inputText id="inputText1"
               label="Label"
               showClear="on"
        value="text"/>
```

Figure 14-14 shows the Input Text component with `showClear` attribute set to `on`.

Figure 14-14 inputText at Run 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 can use the Convert Number (see How to Convert Numeric Values) and Convert Date Time (see 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

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 Customizing the Input Text Component).
- To display a clear text icon to clear the text that user has entered, set the `showClear` attribute to `on`, as shown in Figure 14-14. If the `showClear` attribute is set to `off` then the clear text icon is not displayed. If the `showClear` attribute is set to `auto`, it overrides the behavior of `inputText` to match whatever is set for `keyboardDismiss` attribute. If `keyboardDismiss` attribute is set to `search` then the clear text icon is displayed.

14.3.1.1 Customizing the Input Text Component

MAF AMX provides support for the input capitalization and correction on iOS-powered devices. It also allows you to indicate whether the field is to be used for navigating or for searching. Depending on the version of the operating system and keyboard used, the return button located at the bottom right of the mobile devices's soft keypad (see Figure 14-15) might visually change to a **Go** or **Search** button (see Figure 14-16). In addition, upon activation the button triggers a `DataChangeEvent` for a single-line Input Text component.

Figure 14-15 Return Button on iOS-Powered Device at Runtime

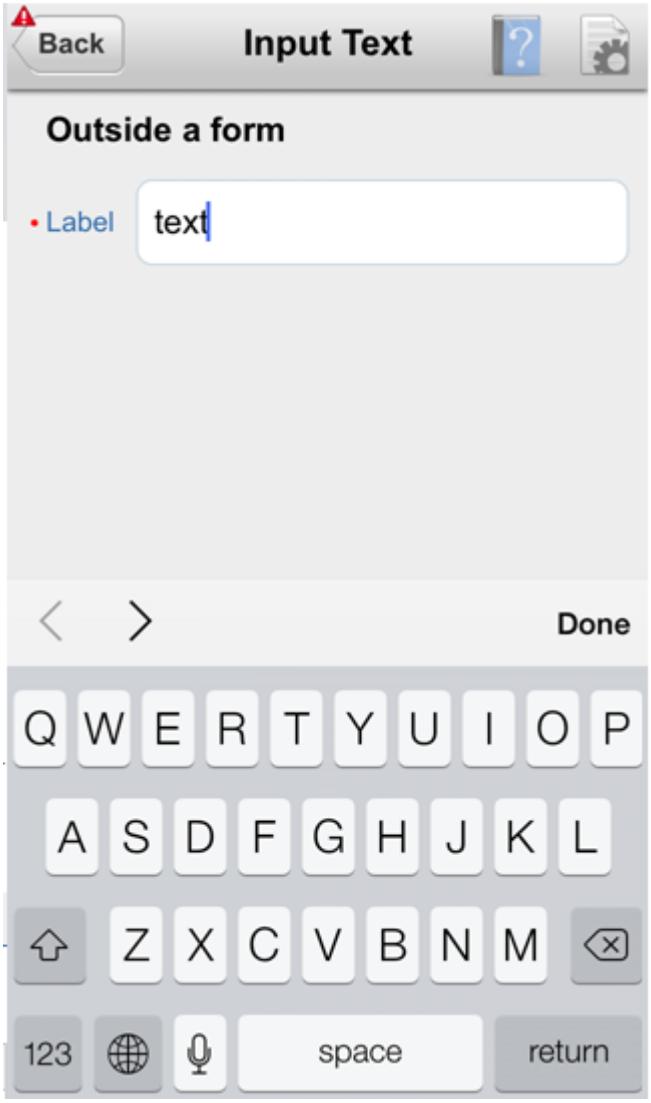
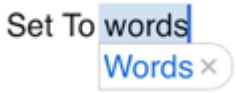
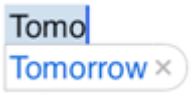


Figure 14-16 Go and Search Buttons on iOS 7 at Runtime

Table 14-3 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 14-3 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 keyboard will cause the input to blur. Some operating systems or keyboards might give special treatment to the keyboard, whereas others show no visual distinction. For example, instead of displaying a Return button on a single-line input text field, that button is replaced with a Go or a Search button. Depending on the skin, this may also alter the appearance of the input field.
`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.

14.3.2 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 following example demonstrates the `inputNumberSlider` element defined in a MAF AMX file.

```
<amx:inputNumberSlider id="slider1" value="#{myBean.count}"/>
```

Figure 14-17 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 14-17 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 How to Convert Numeric Values).

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

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.

14.3.3 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 following example 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, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.4 How to Use the Output Text Component

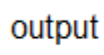
MAF AMX provides the Output Text (`outputText`) component for you to use as a label to display text.

The following example demonstrates the `outputText` element defined in a MAF AMX file.

```
<amx:outputText id="ot1"
                value="output"
                styleClass="#{pageFlowScope.pStyleClass}"/>
```

Figure 14-18 shows the Output Text component displayed in the Preview pane.

Figure 14-18 Output Text at Design Time



You use the Convert Number (see How to Convert Numeric Values) and Convert Date Time (see How to Convert Date and Time Values) converters to facilitate the conversion of numeric and date-and-time-related data for the Output Text components.

For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.3.5 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 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 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 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 14-4 to a Button placed in any valid location within the MAF AMX page.

Table 14-4 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 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 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 Displaying Highlight Style Buttons.
Alert	The Alert style adds the delete appearance to a button. For more information, see 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 14-19). You can define this style in combination with any other style.

Figure 14-19 Rounded Button at Design Time

MAF AMX provides a number of additional decorative styles (see 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 What You May Need to Know About the Order of Processing Operations and Attributes.

14.3.5.1 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 following example demonstrates the `commandButton` element declared in a MAF AMX file. This element represents 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>
```

The following example also demonstrates the `commandButton` element declared in a MAF AMX file. This element represents 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 following example shows a `commandButton` element declared inside the Panel Page's footer facet. This element represent a default Button with a text label and an image icon.

```
<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 a `commandButton` element declared as part of the Panel Page content area. This element represent a default Button with a text label.

```
<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 MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.5.2 Displaying Back Style Buttons

The following are various types of back style buttons that are placed within Panel Page facets or content area:

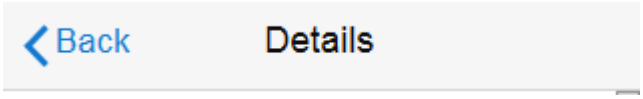
- Normal, activated, or disabled Button with a text label only.
- Normal, activated, or disabled Button with an image icon only:

The following example demonstrates the `commandButton` element declared in a MAF AMX file. This element represent a Back Button with a text label.

```
<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 14-20

Figure 14-20 Back Button an Design Time



For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

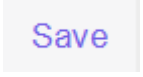
14.3.5.3 Displaying Highlight Style Buttons

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. This element represent a highlight Button with a text label.

```
<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 14-21 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.5.4 Displaying Alert Style Buttons

Alert style buttons placed within the Panel Page can have normal, activated, or disabled state.

The following example demonstrates the `commandButton` element declared in a MAF AMX file. This element represent an Alert Button with a text label.

```
<amx:commandButton id="cb1"
                   text="Delete"
                   actionListener="#{myBean.delete}"
                   styleClass="adfmf-commandButton-alert" />
```

Figure 14-22 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.5.5 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 14-23 Additional Button Styles

14.3.5.6 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

14.3.5.6.1 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.

14.3.5.6.2 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:

14.3.5.6.3 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.

14.3.5.6.4 Alert Messages

An OK button can be placed within a validation message, such as a login validation after a failed password input.

14.3.5.7 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 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 Displaying Back Style Buttons). To disable the styling, 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.5.8 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.

14.3.6 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 How to Use the Set Property Listener), Action Listener (see How to Use the Action Listener), Show Popup Behavior, Close Popup Behavior see How to Use a Popup Component), and Validation Behavior (see 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 How to Display Images) inside a Link you can create a clickable image.

The following example demonstrates a basic `commandLink` element declared in a MAF AMX file.

```
<amx:commandLink id="cl1"
                 text="linked"
                 action="gotolink"
                 actionListener="#{myBean.doSomething}"/>
```

Figure 14-24 shows the basic Link component displayed in the Preview pane.

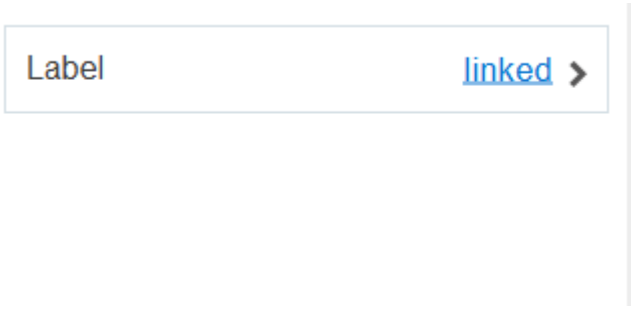
Figure 14-24 Link at Design Time

The following example demonstrates a `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 14-25 shows the Link component placed within a form and displayed in the Preview pane.

Figure 14-25 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 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 14-26 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 14-26 Link Go at Design Time

Go Link

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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.7 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 How to Use Links) to create a clickable image.

The following example 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" />
```

In addition to a URI to an image file, the source can contain a base 64 encoded image data which is required for images loaded from REST web services. You use the

`` prefix to define the source of such images and set the `source` attribute of the `image` element similar to the following:

```
<amx:image id="i2" source="#{row.ImageBase64}" />
```

where values supplied for the GIF file vary depending on the image type.

The following are supported formats on the Android platform:

- GIF
- JPEG
- PNG
- BMP

The following are supported formats on iOS platform:

- PNG

For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample applications located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.8 How to Use the Checkbox Component

The Checkbox (`selectBooleanCheckbox`) component represents a checkbox 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.

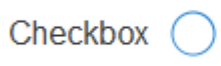
The following example demonstrates the `selectBooleanCheckbox` element declared in a MAF AMX file.

```
<amx:selectBooleanCheckbox id="check1"
                           label="Agree to the terms:"
                           value="#{myBean.bool1}"
                           valueChangeListener=
                              "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 14-27 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 14-27 Unchecked Checkbox at Design Time



shows the checked Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectBooleanCheckbox id="selectBooleanCheckbox1"
                           label="Checkbox"
                           value="true"
                           valueChangeListener=
                                "#{PropertyBean.ValueChangeHandler}"/>
```

Figure 14-28 Checked Checkbox Definition

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.8.1 Support for Checkbox Components on the 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 How to Use the Boolean Switch Component).

14.3.8.2 Support for Checkbox Components on the 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.

14.3.9 How to Use the Select Many Checkbox Component

The Select Many Checkbox (`selectManyCheckbox`) component represents a group of checkboxes that you use to enable multiple selection of `true` or `false` values, which allows toggling between selected and deselected states of each checkbox in the group. The selection mechanism is provided by the Select Items or Select Item component (see 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.

The following example demonstrates a `selectManyCheckbox` element declared in a MAF AMX file.

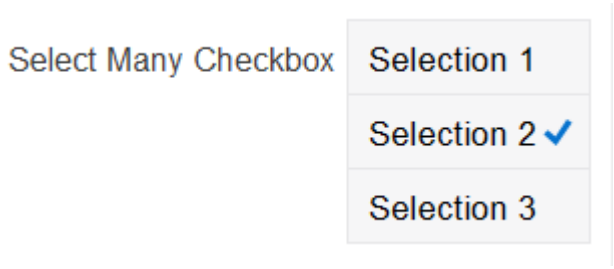
```
<amx:selectManyCheckbox id="selectManyCheckbox1"
                        label="Select shipping options"
                        value="#{myBean.shipping}"
                        valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItem id="selectItem1"
                   label="Air"
```

```
                  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 14-29 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 14-29 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.9.1 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 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.

14.3.10 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 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.

The following 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: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>

<amx:selectOneChoice id="choice1"
                     label="Your state:"
                     value="#{myBean.myState}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItems id="selectItems1" value="myBean.allStates"/>
</amx:selectOneChoice>
```

Figure 14-30 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 14-30 Choice at Design Time

The initial value of the `selectOneChoice` element cannot be `null`. Instead, it must be set to the value displayed in the Select One Choice component. To accomplish this, you must ensure that the value in the model (in the bean or binding) is identical to the default value displayed in JDeveloper at design time.

For more information, illustrations, and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.10.1 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 components 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.

14.3.10.2 What You May Need to Know About the User Interaction with Choice Component on the 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.

14.3.10.3 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.

14.3.11 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 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 a `selectManyChoice` element declared in a MAF AMX file. This component uses a Select Item (`selectItem`) component as a child.

```
<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 14-31 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 located in the PublicSamples.zip file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

The look and behavior of the Select Many Choice component on all supported devices is almost identical to the Select Many Checkbox component (see How to Use the Select Many Checkbox Component for more information).

14.3.12 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 14-32 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 14-32 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.12.1 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.

14.3.12.2 What You May Need to Know About Support for Boolean Switch Components on the Android Platform

The Android platform does not directly support a Boolean Switch component. Instead, Android provides a toggle button that allows tapping to switch between selected and deselected states.

14.3.13 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 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.

The following 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 14-33 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 14-33 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.14 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 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 example demonstrate the `selectOneRadio` element definition in a MAF AMX file. This component uses a Select Item (`selectItem`) component as its child.

```
<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>
```

The following example also demonstrate the `selectOneRadio` element definition in a MAF AMX file. This component uses a Select Items (`selectItems`) component as its child.

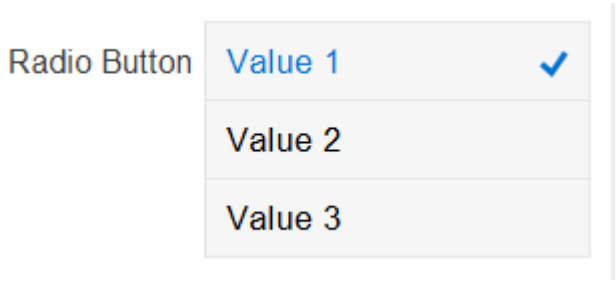
```
<amx:selectOneRadio id="radio1"
                    label="Choose a pet:"
                    value="#{myBean.myPet}"
                     valueChangeListener="#{PropertyBean.ValueChangeHandler}">
   <amx:selectItems id="selectItems1" value="myBean.allPets"/>
</amx:selectOneRadio>
```

Figure 14-34 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 14-34 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.15 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 option.

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 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.

At runtime, List Item components respond to swipe gestures (see Enabling Gestures).

You can create the following types of List View components:

- Basic List

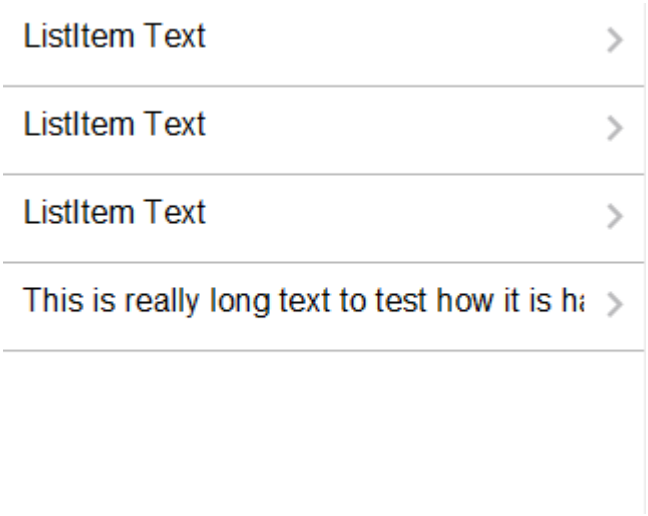
  The following example shows the `listView` element defined in a MAF AMX file. This definition corresponds to the basic component.

  ```
  <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 14-35 demonstrates a basic List View component at design time.

Figure 14-35 Basic List View at Design Time



The following 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.

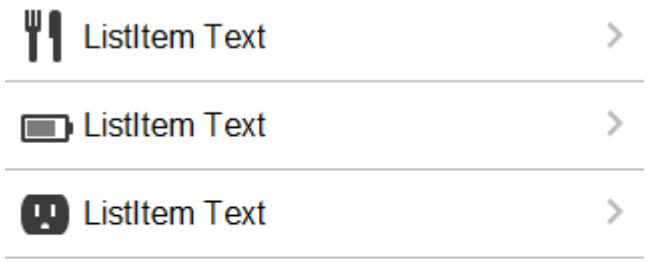
- List with icons

  The following 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 14-36 demonstrates a List View component with icons and text at design time.

Figure 14-36 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.

The following 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 14-5 describes styles that you can define for the index utility.

Table 14-5 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 must 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.

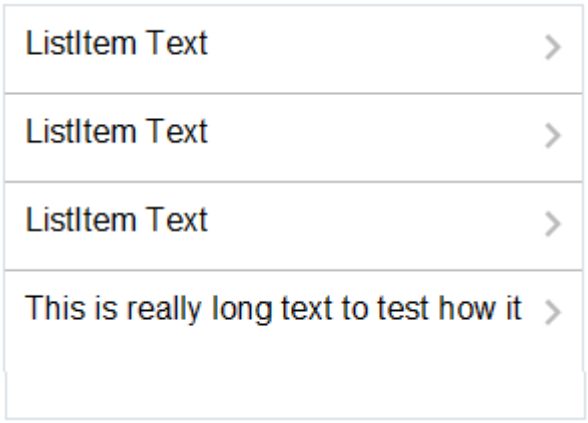
- Inset List

  The following example shows the `listView` element defined in a MAF AMX file. This 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 14-37 demonstrates an inset List View component at design time.

Figure 14-37 Inset List View at Design Time



The following 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 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, a dialog called **ListView Gallery** appears (see Figure 14-38). This dialog allows you to choose a specific layout for the List View.

Figure 14-38 ListView Gallery Dialog

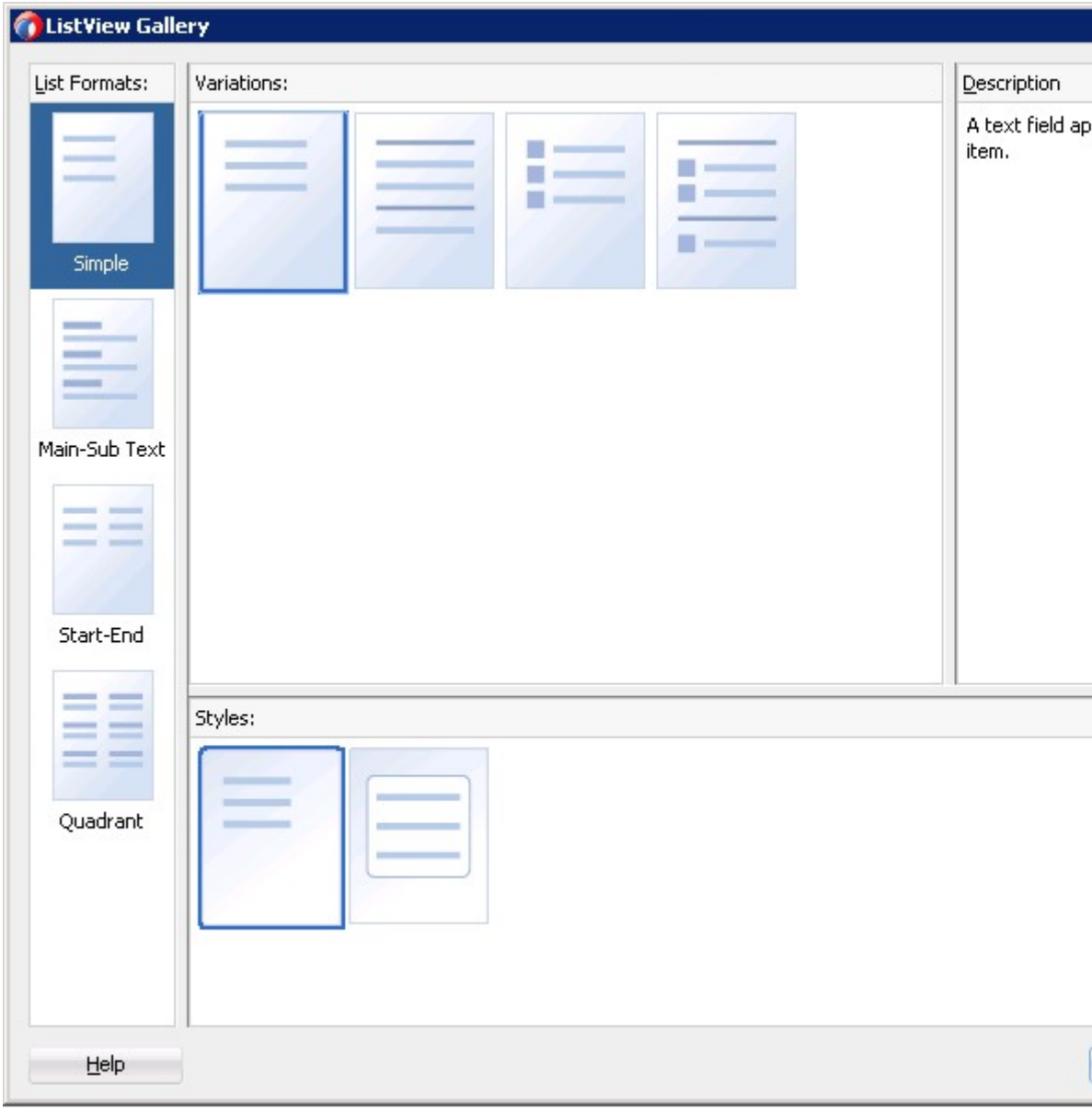


Table 14-6 lists the supported **List Formats** that are displayed in the ListView Gallery.

Table 14-6 List View Formats

Format	Element Row Values
Simple	• Text
Main-Sub Text	• Main Text • Subordinate Text
Start-End	• Start Text • End Text

Table 14-6 (Cont.) List View Formats

Format	Element Row Values
Quadrant	• Upper Start Text • Upper End Text • Lower Start Text • Lower End Text

The **Variations** presented in the ListView Gallery (see Figure 14-38) 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 (see Figure 14-39).
- 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 (see Figure 14-38) 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 (see Figure 14-39). 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 14-7.

Table 14-7 List View Variations and Styles

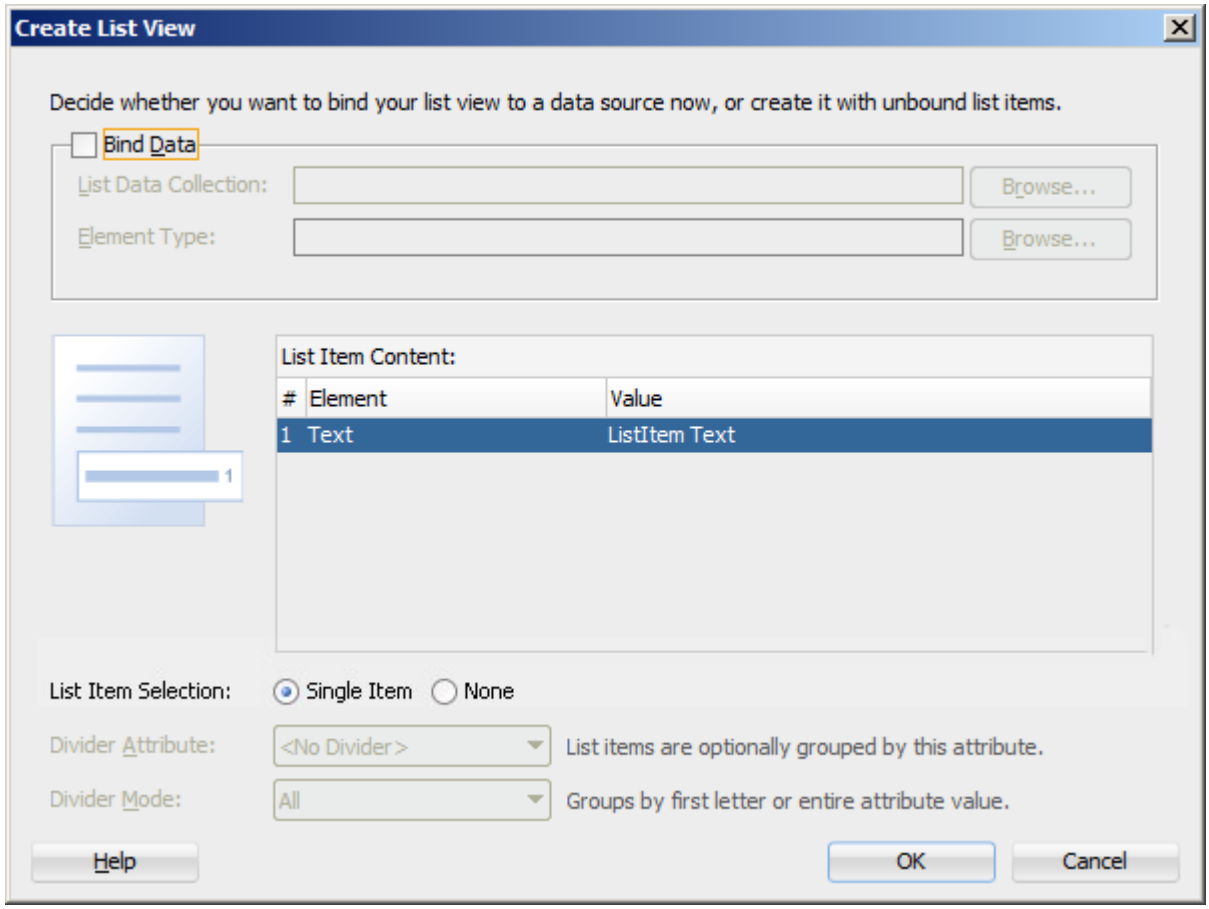
List Format	Variation	Description
Simple	Basic	A text field appears at the start side of the list item.
Simple	Dividers	A text field appears at the start side of the list item, with items grouped by dividers.
Simple	Images	A text field appears at the start side of the list item, following a leading image.

Table 14-7 (Cont.) List View Variations and Styles

List Format	Variation	Description
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 make your selection from the ListView Gallery and click **OK**, the **Edit List View** wizard is invoked that lets you create either an unbound List View component that displays static text in the List Item child components (see Figure 14-39), or choose a data source for the dynamic binding (see Figure 14-40).

Figure 14-39 Creating Unbound List View



When completing the dialog that Figure 14-39 shows, consider the following:

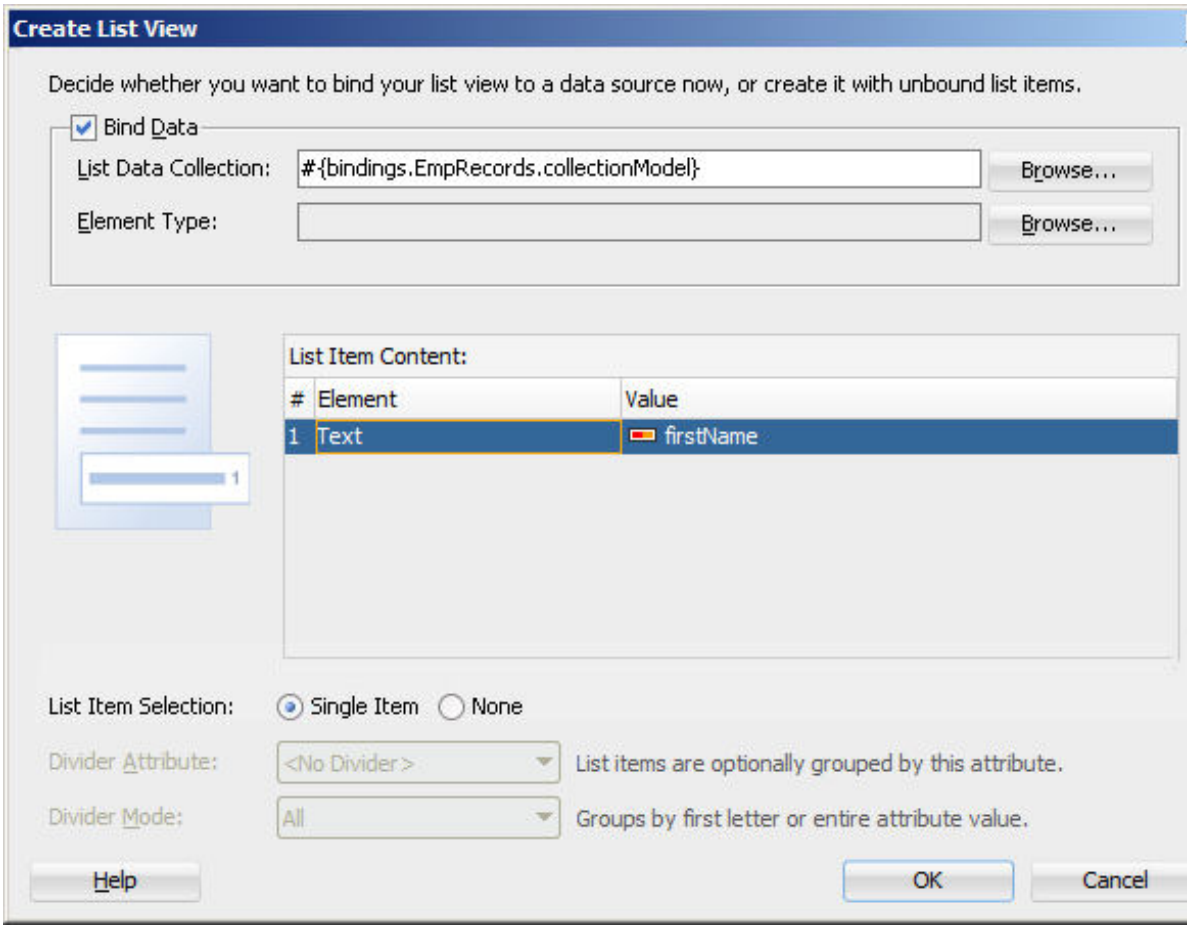
- The **List Data Collection** and **Element Type** fields are disabled when the **Bind Data** checkbox is in the deselected state.
- The image on the left reflects the main content elements from the selected List View format
- The editable cells of the **Value** column are populated with static text strings (see Table 14-8).
- 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 must replace it with the path to the image.
- The **List Item Selection** indicates the selection mode. For details, see the description of this option following Figure 13-79.
- 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 14-8 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 14-40 shows the **Create List View** dialog when you choose to bind the List View component to data.

Figure 14-40 Creating Bound List View



When completing the dialog that Figure 14-40 shows, consider the following:

- When you select the **Bind Data** checkbox, the **List Data Collection** field becomes enabled. The **Element Type** field becomes enabled if you set the **List Data**

Collection field to an EL expression by using the Expression Builder. If you choose a data control from the **Data Control Definitions** tab, the **Element Type** field will continue to be disabled.

- To select a data source, click **Browse**. This opens the **Select List View Data Collection** dialog that enables you to either choose a data control definition (see Figure 14-41) or to use the EL Builder to select an appropriate EL expression (see Figure 14-42).

Figure 14-41 Selecting Data Control Definition

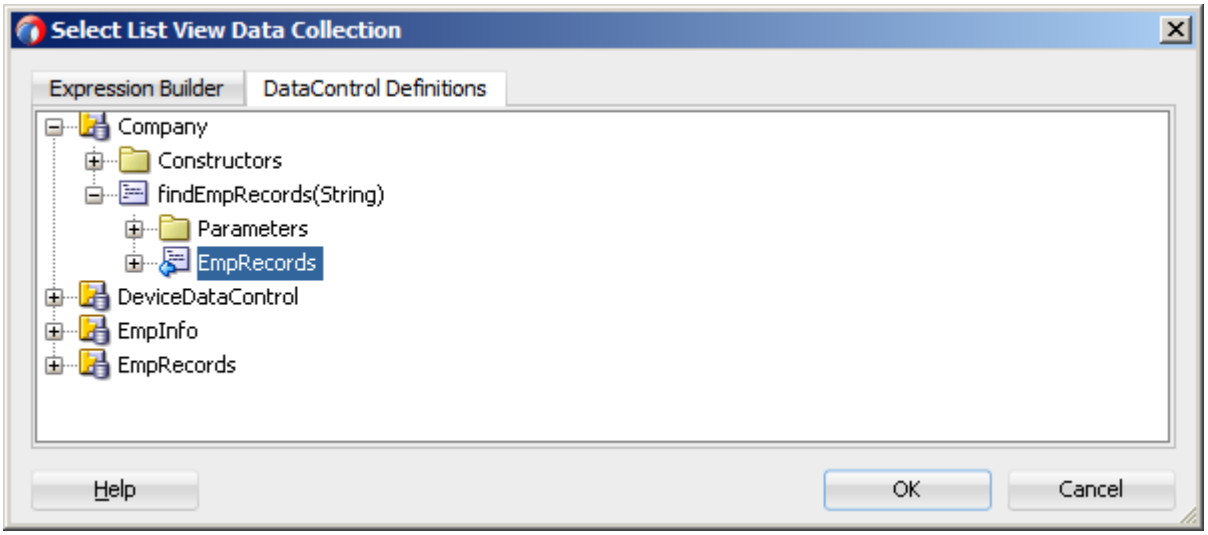
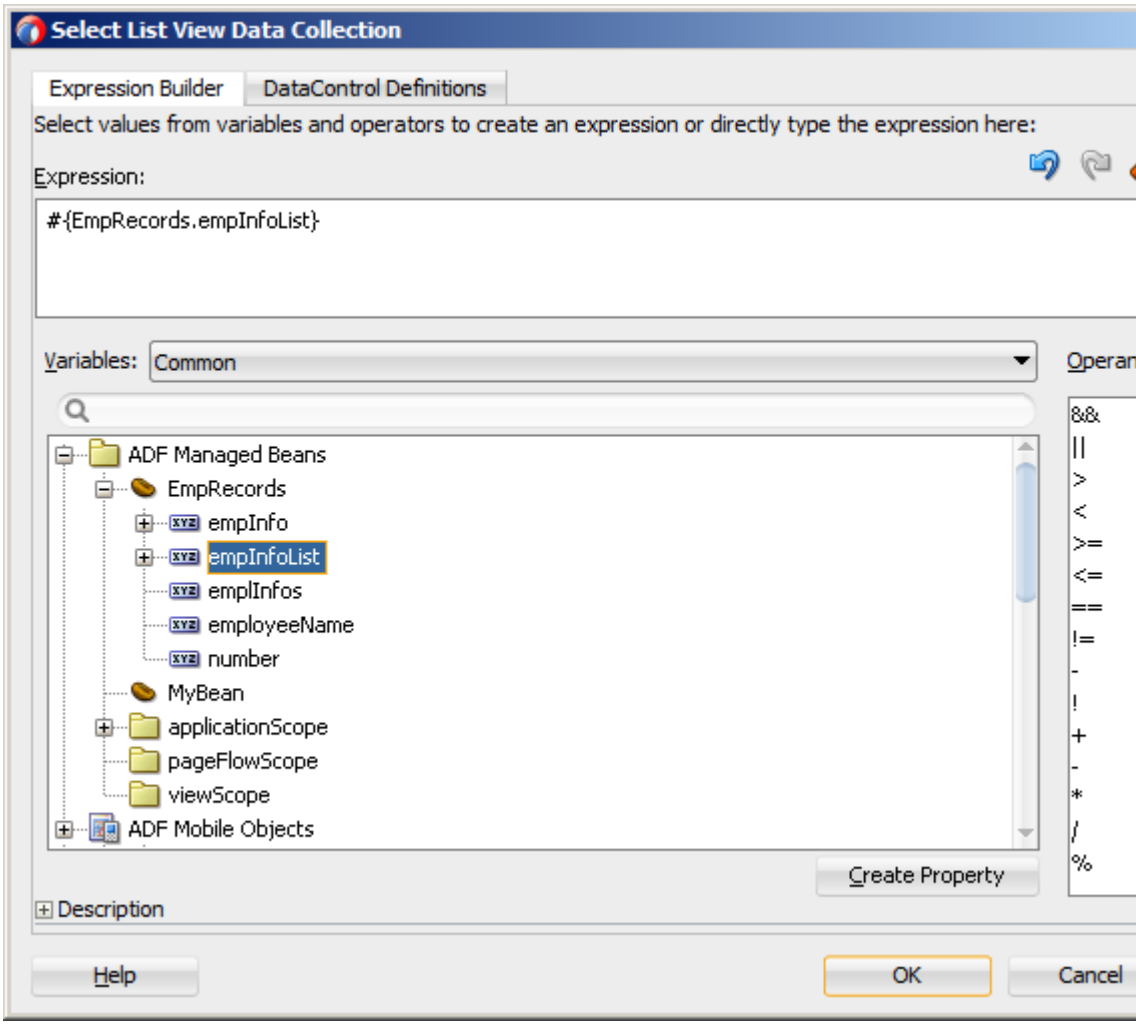


Figure 14-42 Selecting EL Expression



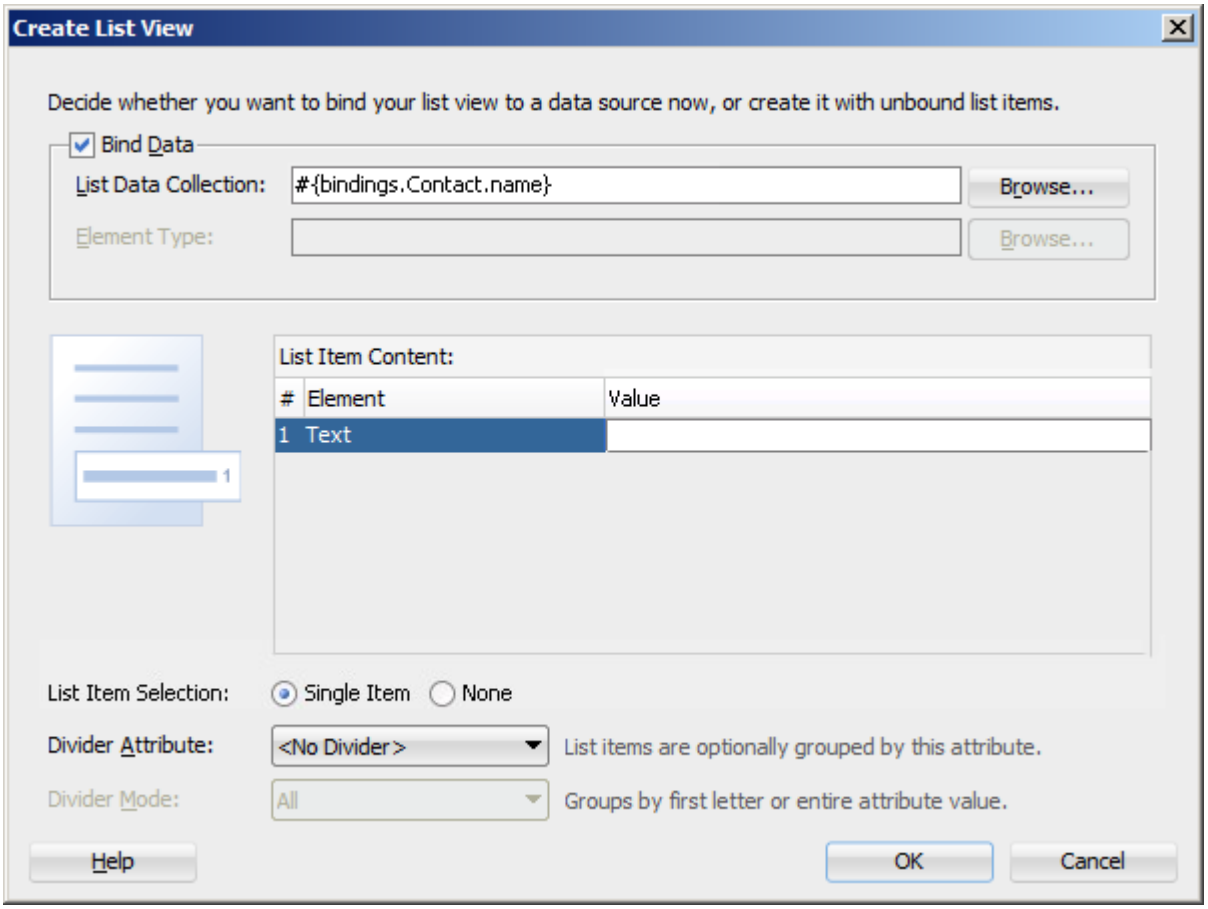
- You may declare the type of the data collection using the **Element Type** field (see Figure 14-40).
- After you have selected a valid data collection, the **Value** column in the **List Item Content** table changes to **Value Bindings** whose editable cells are populated with lists of attributes from the data collection. For a description of a special case setting, refer to Figure 14-43.
- The image on the left reflects the main content elements from the selected List View format and provides a mapping from the schematic representation to the named elements in the underlying table.
- The List Item is generated as either an Output Text or Image component, depending on whichever is appropriate for the particular element.
- Since the number of elements (rows) is predetermined by the selected List View format, rows cannot be added or removed.
- The order of elements cannot be modified.
- The default value of the **Divider Attribute** field is **No Divider**, in which case the **Divider Mode** field is disabled. If you select value other than the default, then

you need to specify Divider Mode parameters. In addition, if you chose a Variation in the ListView Gallery that includes dividers, the default value will be set to the first attribute in the list.

The following are special cases to consider when creating a bound List View:

- If a Java bean method returns a list without generics, you should use the **Element Type** field to create the List Item content, as Figure 14-40 shows.
- If the list data collection value provided is not collection-based, a **Value** column replaces the **Value Bindings** column with blank values, as Figure 14-43 shows.

Figure 14-43 Providing Non-Collection-Based Values



For more information, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- Configuring the List View Layout
- Dragging and Dropping Collections
- CompGallery and UILayoutDemo, MAF sample applications located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer. These samples demonstrate how to use various types of the List View component and how to apply styles to adjust the page layout to a specific pattern.

14.3.15.1 Configuring Paging and Dynamic Scrolling

You can configure the List View component to display data in a list that is arbitrarily long by appending data to the bottom of the list as requested by a user gesture.

The `fetchSize` attribute determines how many rows the List View component should initially load. The `maxRows` attribute specifies the maximum number of list items to display. If there are more rows available than defined by the `maxRows` attribute, the List View waits for a specific user gesture before loading and displaying more rows (see List View Scrolling Strategies). The `fetchSize` attribute is then used to determine how many rows will be loaded and displayed each time the user gestures for more rows to be displayed.

If the `fetchSize` attribute is set to -1 and you do not specify a value for the `maxRows` attribute, all records are retrieved and displayed, in which case neither paging nor dynamic scrolling behavior occurs.

When the List View component is created by dragging a collection from the Data Controls window onto a MAF AMX page, the `fetchSize` attribute is by default set to use an EL expression that points to the `rangeSize` of the `PageDef` iterator and should not be modified. For more information, see the following:

- How to Reference Binding Containers
- What You May Need to Know About Generated Drag and Drop Artifacts
- Figure 13-85

The following example shows the `listView` element that was created from a collection called `testResults` of a data control (see How to Add Data Controls to a MAF AMX Page).

```
<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 following 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"/>
```

You can specify the initial scroll position of a self-scrollable List View (that is, the List View provides its own scrolling; see List View's Own Scrolling) using its `initialScrollRowKeys` attribute. For convenience, the value of this attribute can be set to the same EL expression as value of the `selectedRowKeys` attribute, as the following example shows.

```
<amx:listView id="lv1"
              var="row"
              value="#{bindings.departments.collectionModel}"
              fetchSize="25"
              inlineStyle="height: 200px"
              selectedRowKeys="#{bindings.departments.collectionModel.selectedRow}
```

```
                selectionListener="#{bindings.departments.collectionModel.makeCurrent}"
initialScrollRowKeys="#{bindings.departments.collectionModel.selectedRow}">
   <amx:listItem id="li1">
      <amx:outputText id="ot1" value="#{row.name} #{row.id}"/>
   </amx:listItem>
</amx:listView>
```

Since MAF assigns the scroll position when the first set of rows is rendered, you must ensure that the specified row key is a part of the initial set of fetched rows.

The List View component also exposes a `preserveState` attribute that, when set to `none`, prevents MAF applying the past state of the List View component when the end user navigates back to the AMX page that renders the component. In this scenario the values of the `initialScrollRowKeys` and `maxRows` attributes determine the scroll position when an end user navigates back to the page. The default behavior where MAF applies the past state of the component, if available, can be re-enabled by setting `preserveState="auto"`.

For more information about the List View component and its attributes, see *Tag Reference for Oracle Mobile Application Framework*.

14.3.15.1.1 List View Scrolling Strategies

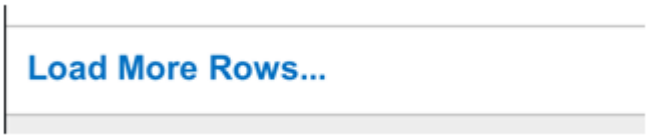
The following attributes of the List View component enable its scrolling behavior:

- `showMoreStrategy`: defines the List View component's strategy for loading more rows when required.

  When a List View component provides its own scrolling (see List View's Own Scrolling) and that List View is scrolled to the end, it automatically invokes the `showMoreStrategy` based on the attribute's value, as follows:

  - `autoLink`: If more rows are available from the model, the List View displays a Load More Rows link at the bottom of the displayed list, as Figure 14-44 shows.

    Figure 14-44 Loading More Rows in List View

    The end user must tap on this link to cause the List View to load and display more rows.

  - `autoScroll`: If more rows are available from the model, the List View displays a load indicator while it loads more rows for display.

  - `forceLink`: A Load More Rows link is displayed (see Figure 14-44). When the end user taps on the link, the List View attempts to load and display more rows.

  - `off`: The List View does not perform any actions. Only the initially loaded rows are displayed.

- `bufferStrategy`: defines the List View component's strategy for buffering displayed rows.

When the List View's height is constrained allowing it to provide its own scrolling (see List View's Own Scrolling), it retains rows in the rendering engine's buffer based on the bufferStrategy attribute's value, as follows:

  - `additive`: New rows are added to the rendering engine's buffer and all rows are retained in the buffer. This option is useful for short lists where you are not concerned about memory consumption.
  - `viewport`: Rows are added to the rendering engine's buffer only when they become visible within the List View's viewport. Rows are removed from the buffer when they are no longer visible, based on the List View's `bufferSize` attribute. This option is useful for reducing the amount of memory consumption when displaying long lists.

- `bufferSize`: when the bufferStrategy attribute is set to viewport, the bufferSize attribute defines the distance (in pixels) at which the row must be located from the viewport to become hidden.

For more information, see *Tag Reference for Oracle Mobile Application Framework*.

14.3.15.1.2 List View's Own Scrolling

By default, the scrolling behavior of the List View component is controlled by its parent container (which, in turn, may default to its parent container, and so forth).

To force the List View component to provide its own scrolling, you can do one of the following

- 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;"
```

14.3.15.1.3 Server-Side Paging

The List View component supports server-side paging through events such as the `oracle.adfmf.amx.event.RangeChangeEvent`.

When the List View component is created by dragging a collection from the Data Controls window onto a MAF AMX page, the List View component retrieves the rows from the binding iterator (see Configuring Paging and Dynamic Scrolling). The binding iterator retrieves rows from the data control's collection in batches defined by the `AttributeIterator`'s `RangeSize` attribute. When all the available data has been exhausted, a `RangeChangeEvent`is fired. To catch this event, the data control's provider code must implement the `oracle.adfmf.amx.event.RangeChangeListener` and provide a `rangeChange` method. Within this method, you can load more data from the server and append it to the collection. You must call the `addDataControlProviders` method of the `AdfmfJavaUtilities` class to inform the data control framework of the newly added rows so these rows can be displayed by the List View component.

```
public class Departments implements RangeChangeListener {
   public void rangeChange(RangeChangeEvent rce) {
      List newRows = null;
      if (rangeChangeEvent.isDataExhausted()) {
         newRows = loadMoreData(rangeChangeEvent.getFetchSize());
         AdfmfJavaUtilities.addDataControlProviders("Departments",
                                    rangeChangeEvent.getProviderKey(),
                                    rangeChangeEvent.getKeyAttribute(),
```

```
                                    newRows,
                                    newRows.size() > 0);
        }
    }
...
}
```

Note:

When instantiating the data control's provider class, the initial load of data from the server should request the same number of rows as defined by the binding iterator's `RangeSize` attribute.

When using a managed bean to provide the model for a List View, the `rangeChangeListener` attribute (see Using Event Listeners) of the List View component allows you to bind a Java handler method that is called when the end user gestures for more rows to be loaded. 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, as Figure 14-45 and Figure 14-46 show.

Figure 14-45 Editing Range Change Listener Attribute

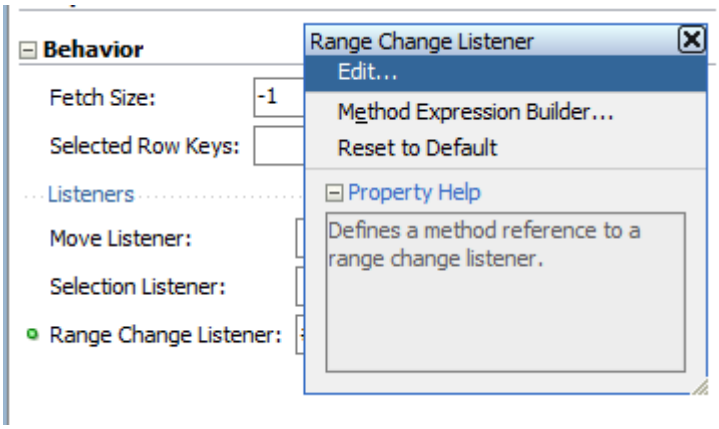
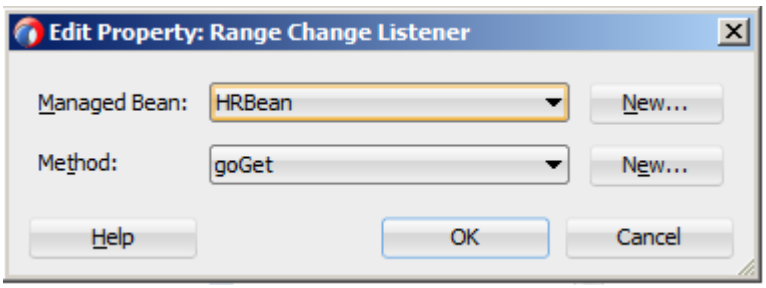


Figure 14-46 Edit Property Dialog



When you click **OK** on the dialog, the following setting is added to the `listView` element definition in the MAF AMX page:

```
<amx:listView id="listView1" rangeChangeListener="#{pageFlowScope.HRBean.goGet}" >
```

In addition, the Java method shown in the following example is added to a sample `HRBean` class:

```
public void goGet(RangeChangeEvent rangeChangeEvent) {
   // Add event code here
   ...
}
```

Note:

When using the `RangeChangeEvent` to support server-side paging, you should not set the ListView's `fetchSize` attribute to -1.

For additional examples, see a MAF sample application called RangeChangeDemo located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.3.15.2 What You May Need to Know About Memory Consumption by MAF AMX UI Components

All scrollable MAF AMX UI components, including the List View, are optimized to conserve resources when a mobile device is running low on memory. These components lose their flickability (that is, the end user cannot flick the component with their finger in order for that component to continue to scroll after the end user has stopped touching the screen) and scrolling is powered by inertia.

14.3.15.3 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.

The following 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*.

14.3.15.4 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 s 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 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 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 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 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 Styling UI Components) that uses a set of predefined styles.

A MAF sample application called UILayoutDemo demonstrates all the optional styles for each component and their associated rendering. This application is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

The following example 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 14-47 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 14-47 List View with Start and End Text at Design Time

The following example 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 14-48 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.

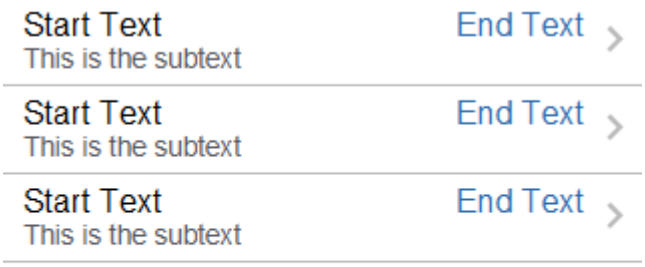
Figure 14-48 List View with Start and End Text Without Expansion Links at Design Time

The following example 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 gray 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 gray 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 14-49 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 14-49 List View with Start and End Text and Subtext at Design Time

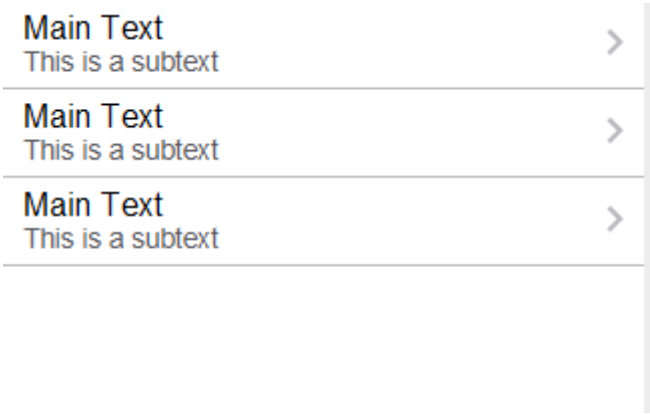


The following example shows the `listView` element and its child elements defined in a MAF AMX file. This List View is populated with rows containing a main text and subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. 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 gray 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 14-50 shows a List View component with differently styled text added as a main text and subtext to each row.

Figure 14-50 List View with Main Text and Subtext at Design Time

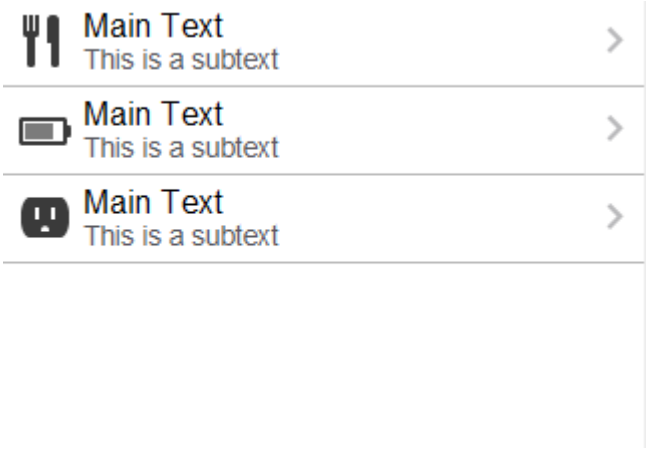


The following example shows the `listView` element and its child elements defined in a MAF AMX file. This List View is populated with cells containing an icon, main text, and subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `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 gray 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 14-51 shows a List View component with icons and differently styled text added as a main text and subtext to each row.

Figure 14-51 List View with Icons, Main Text and Subtext at Design Time



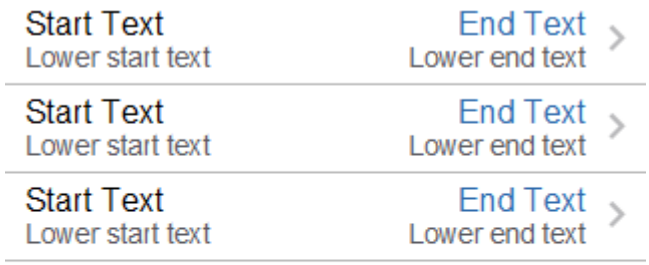
The following example shows the `listView` element and its child elements defined in a MAF AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid 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 gray 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 gray 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 gray 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 14-52 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 14-52 List View with Four Types of Text at Design Time



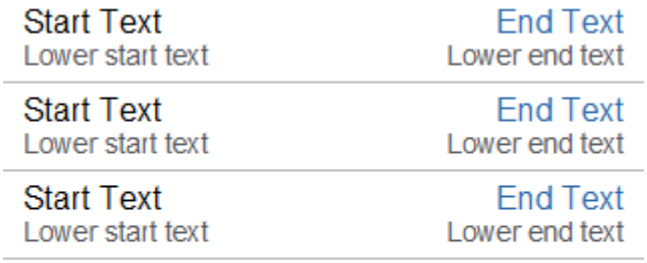
The following example shows the `listView` element and its child elements defined in a MAF AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid 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 gray 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 gray 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 gray 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 14-53 shows a List View component with two types of differently styled text added to the start and end of each row.

Figure 14-53 List View with Four Types of Text and Without Expansion Links at Design Time



14.3.15.5 What You May Need to Know About Using Static List View

If you create a List View component that is not populated from the model but by hard-coded 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).

14.3.16 How to Use a 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.

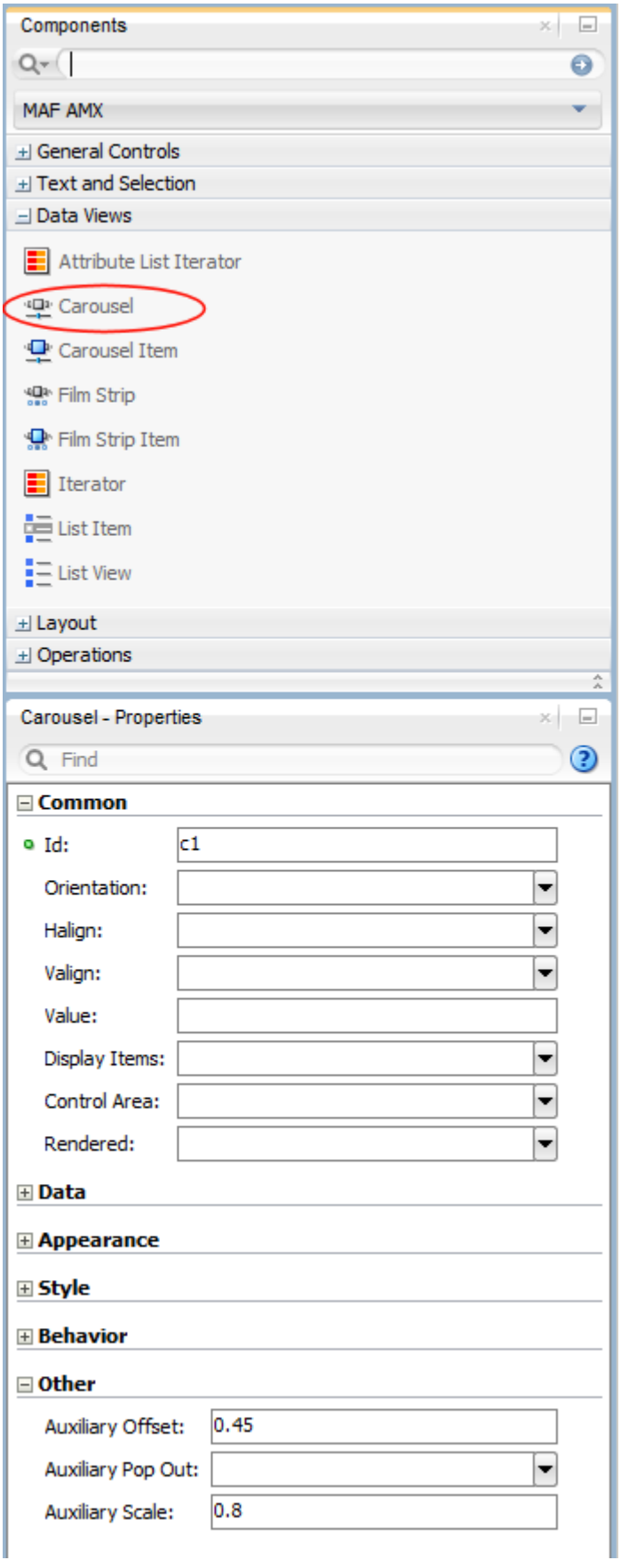
By default, the Carousel displays horizontally. The objects within the horizontal orientation of the Carousel are vertically-aligned to the middle and the Carousel itself is horizontally-aligned to the center of its container. You can configure the Carousel so that it can be displayed vertically, as 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 JDeveloper, the Carousel is located under Data Views in the Components window (see Figure 14-54).

Figure 14-54 Carousel in Components Window



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 following example 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:

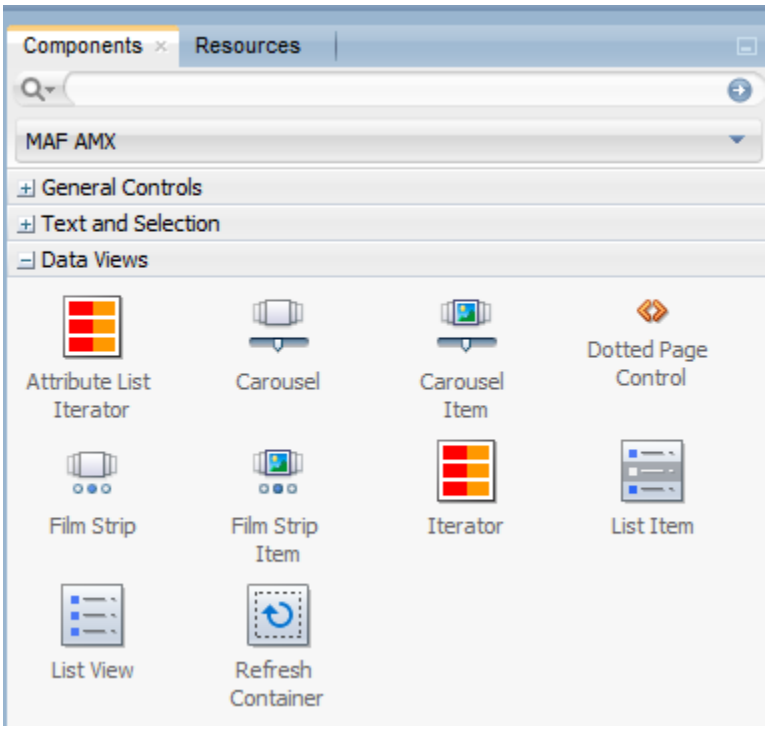
- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.17 How to Use the Film Strip Component

A Film Strip (`filmStrip`) is a container 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 JDeveloper, in the **Components** window, the Film Strip is located under **Data Views** (see Figure 14-55).

Figure 14-55 Film Strip in Components Window



The following example 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}"
               pageControlPosition="#{pageFlowScope.pageControlPosition}">
   <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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.17.1 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.

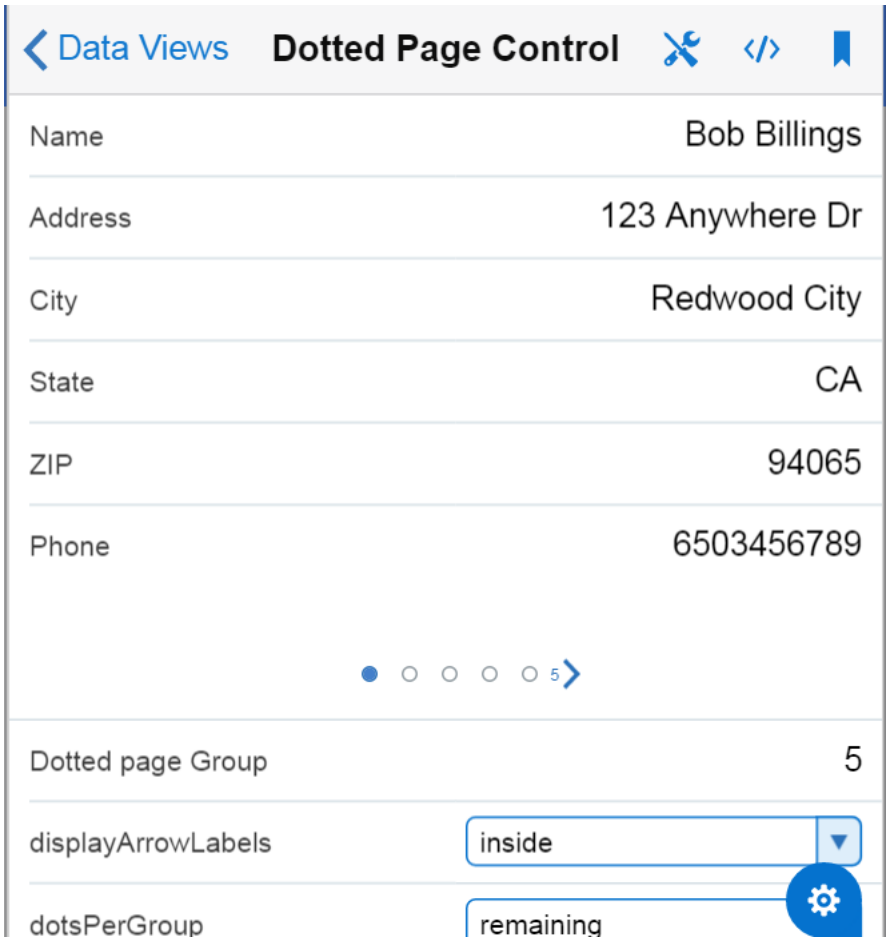
A horizontally laid out Film Strip should reflect the text direction mode: in the left-to-ride mode, the first item is located on the left; in the right-to-left mode, the first item is located on the right. In both cases, the page control is at the bottom.

Using the `pageControlPosition` attribute of the Film Strip component, you can configure the location of the page control to be either inside or outside of a Film Strip Item.

When the `dottedPageControl` component is configured, it allows you to display an overflow dotted page control for navigation among Film Strip pages as shown in Figure 14-56. The following example demonstrates the Film Strip element along with `dottedPageControl` component configured in a MAF AMX file.

```
<amx:filmStrip>
<amx:filmStripItem/>
   <amx:facet name="pageControl">
   <amx:dottedPageControl id="pc1" dotsPerGroup="10" lastGroupBehavior="remaining|
full" displayArrowLabels="none|inside|outside"/>
   </amx:facet>
</amx:filmStrip>
```

Figure 14-56 Dotted Page Control Component



14.3.17.2 What You May Need to Know About the Film Strip Navigation

The navigation of the Film Strip component is similar to the Deck (see How to Use a Deck Component) and Panel Splitter component (see How to Use a Panel Splitter Component): one page at a time is displayed and the page change is triggered by selection of the page ID or number.

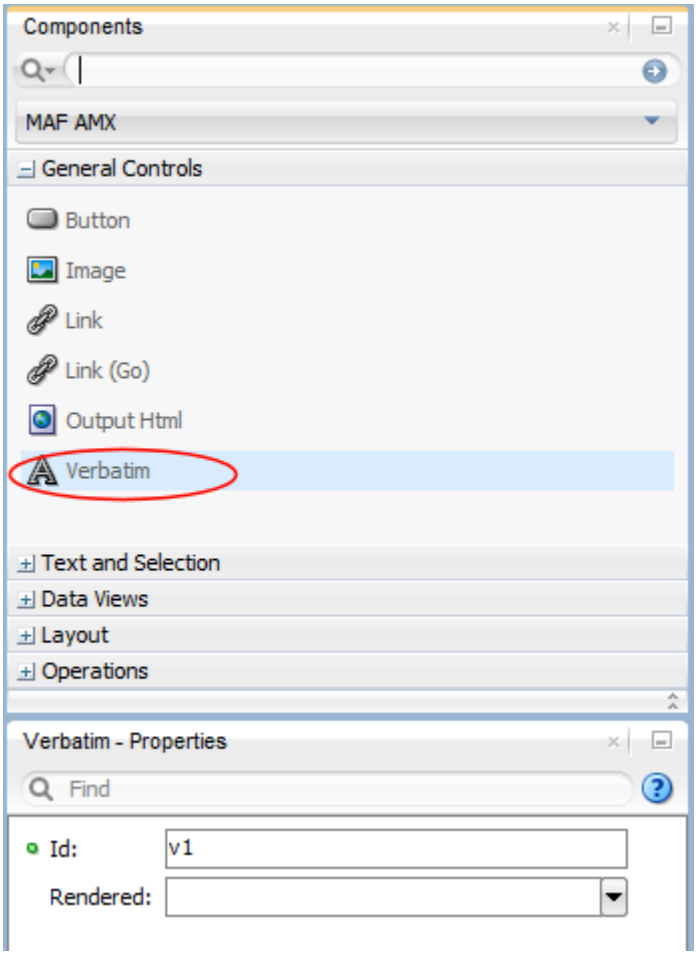
Since the child Film Strip Item component is not meant to be used for navigation to other MAF AMX pages, it does not support the `action` attribute. When required, you can enable navigation through the nested Command Link component.

14.3.18 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 JDeveloper, Verbatim is located under **General Controls** in the Components window (see Figure 14-59).

Figure 14-57 Verbatim in Components Window



For more information and examples, see the following:

- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.18.1 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.

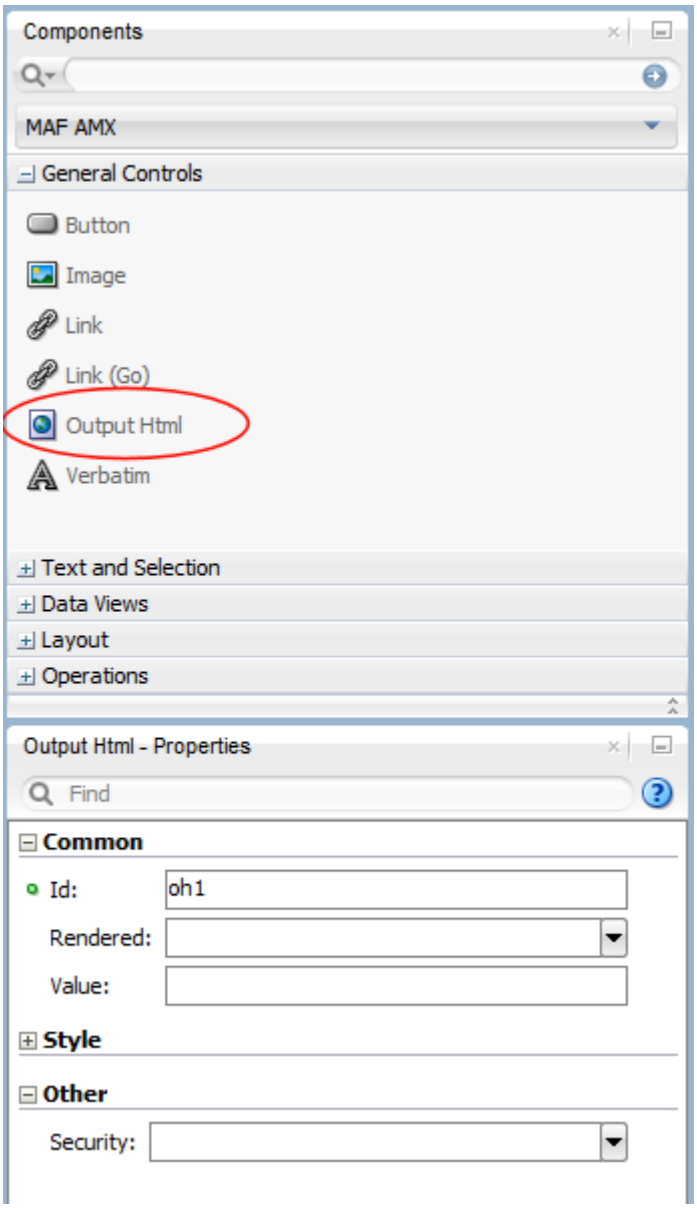
In addition, the use of JavaScript with the Verbatim component is affected by the fact that AJAX calls from an AMX page to a server are not supported. This is due to the security architecture that guarantees that the browser hosting the MAF AMX page does not have access to the security information needed to make connections to a secure server to obtain its resources. Instead, communication with the server must occur from the embedded Java code layer.

14.3.19 How to Use an 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 How to Use Verbatim Component) and the same restrictions with regards to JavaScript and AJAX usage apply to it (see What You May Need to Know About Using JavaScript and AJAX with Verbatim Component).

In JDeveloper, **Output HTML** is located under **General Controls** in the **Components** window (see Figure 14-58).

Figure 14-58 Output HTML in Components Window



The following example 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 following example shows the code from the Java bean called `MyPurchaseBean` that provides HTML for the Output HTML component shown in the preceding example.

```
// 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 Java bean 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 Securing MAF Applications and About Injection Attack Risks from Custom HTML Components. 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:

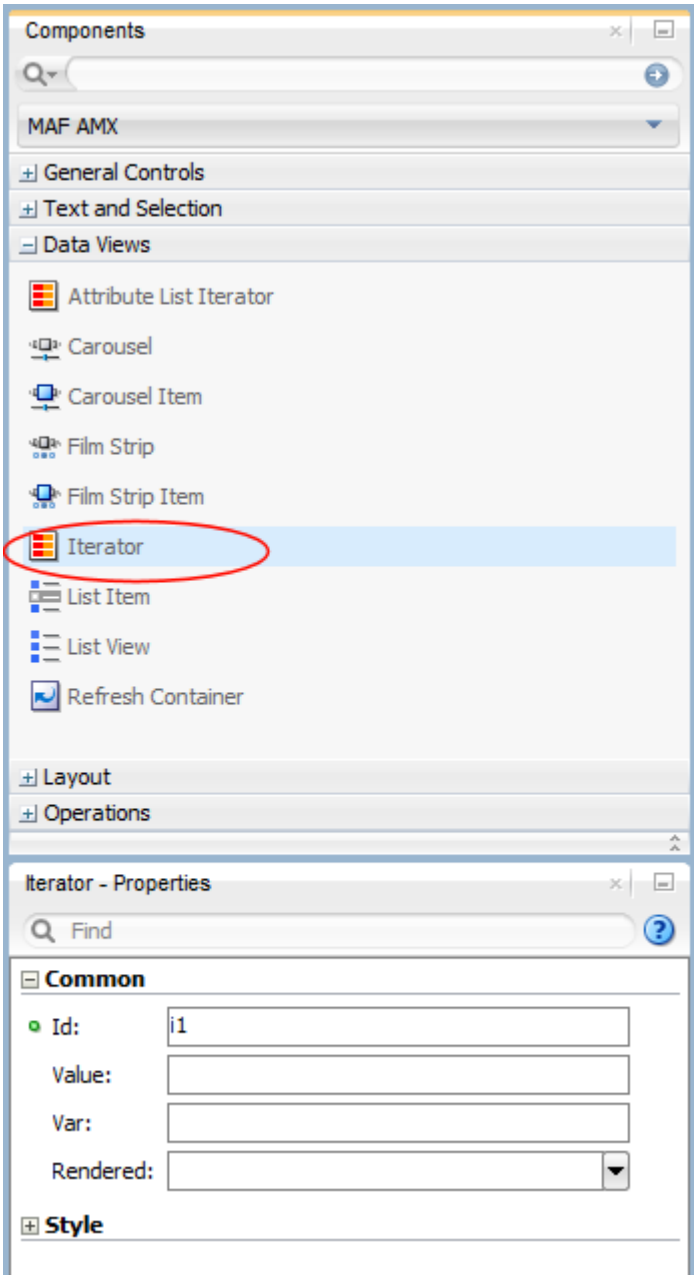
- *Tag Reference for Oracle Mobile Application Framework*
- CompGallery, a MAF sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.20 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 JDeveloper, the Iterator is located under Data Views in the **Components** window (see Figure 14-59).

Figure 14-59 Iterator in Components Window



For more information, see *Tag Reference for Oracle Mobile Application Framework*.

14.3.21 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.

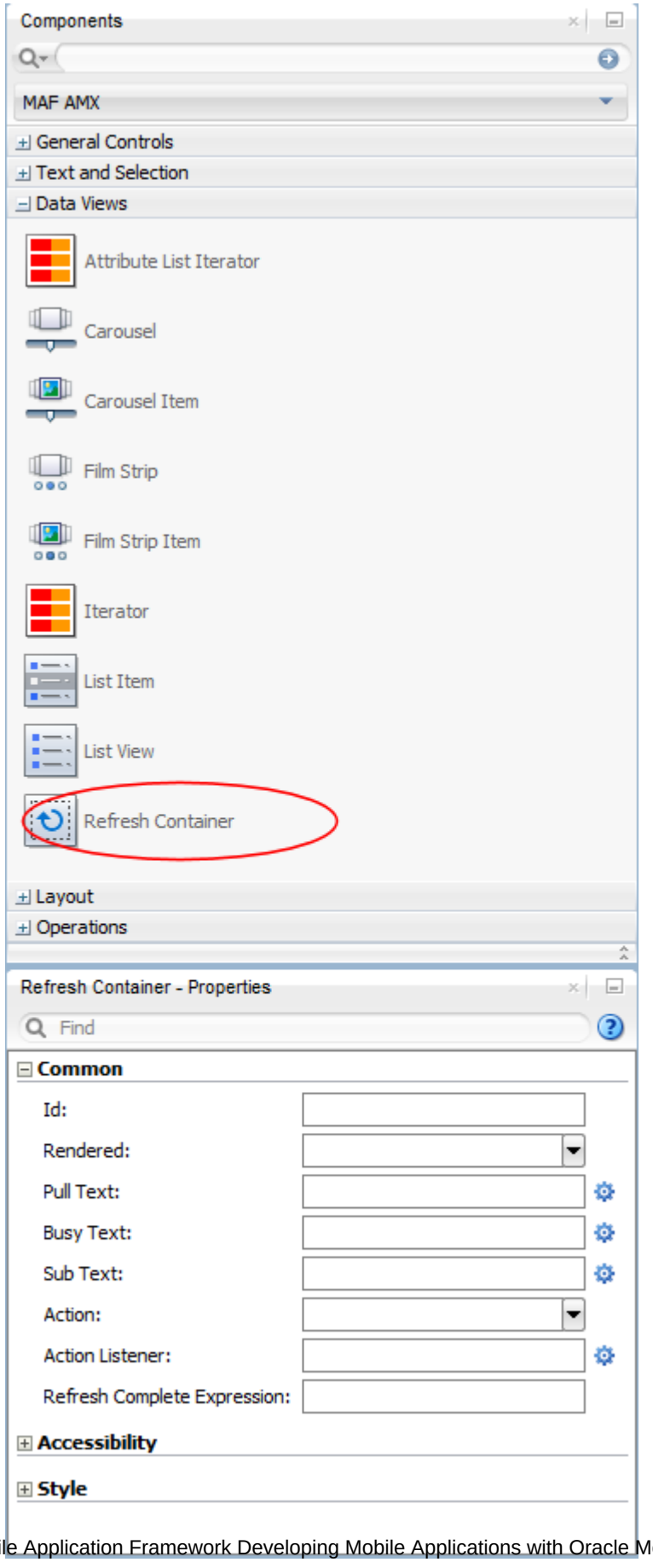
Upon release of a pull down gesture or reaching threshold, the contents' update begins and the status indicator changes until the contents, such as List Item instances, are refreshed.

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.

In JDeveloper, the **Refresh Container** is located under **Data Views** in the **Components** window.

Figure 14-60 Refresh Container in Components Window



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**.

```
<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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

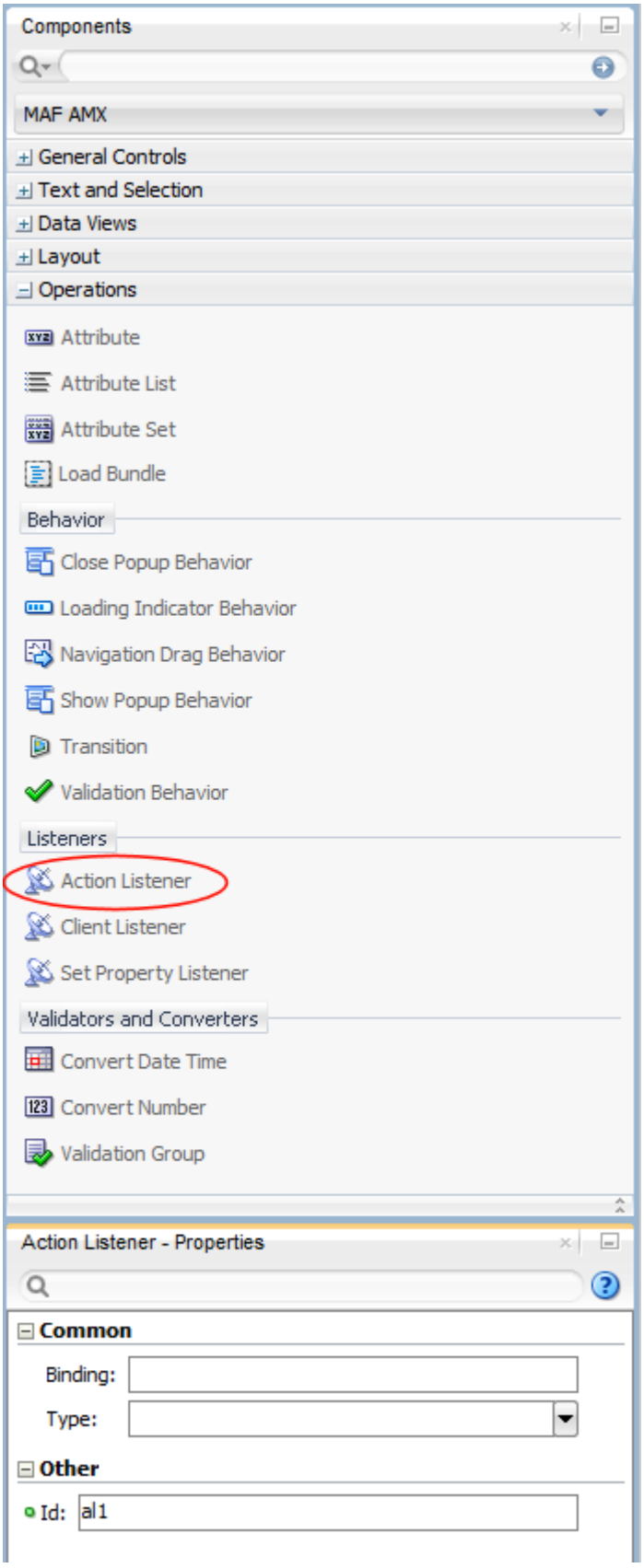
14.3.22 How to Use the Action Listener

The Action Listener (`actionListener`) component allows you to declaratively invoke commands through EL based on the type of the parent component's usage.

The predominate 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 What You May Need to Know About Differences Between the Action Listener Component and Attribute.

In JDeveloper, the **Action Listener** component is located under **Operations -> Listeners** in the **Components** window (see Figure 14-61).

Figure 14-61 Action Listener in Components Window



You can add zero or more Action Listener components as children of any command component (Button, Link, List Item, Film Strip Item). 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:

- Using Event Listeners
- How to Use the Set Property Listener
- Enabling Gestures
- What You May Need to Know About Differences Between the Action Listener Component and Attribute
- *Tag Reference for Oracle Mobile Application Framework*

14.3.22.1 What You May Need to Know About Differences Between the Action Listener Component and Attribute

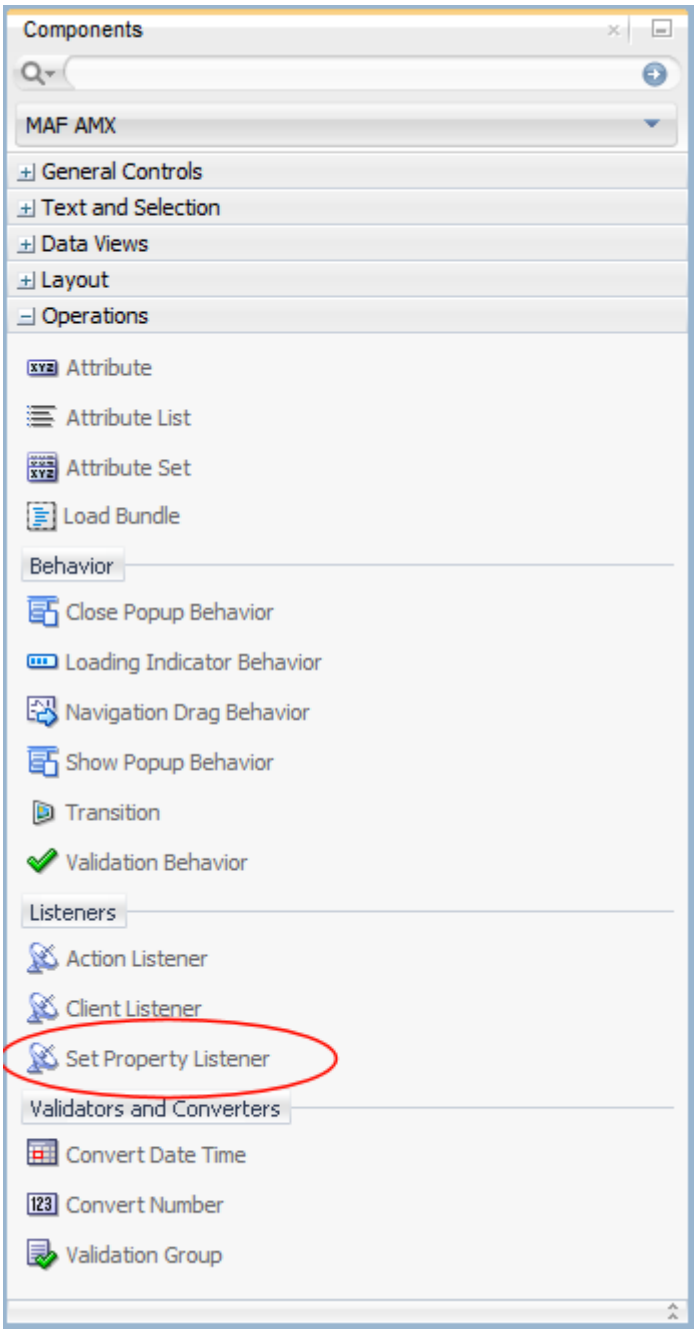
Components such as the Button, Link, and List Item have an `actionListener` attribute, which by inference seems to make the Action Listener component redundant. However, unlike the Action Listener component, these components do not have the `type` attribute that supports gestures, which is the reason MAF provides the Action Listener component in addition to the `actionListener` attribute of the parent components.

14.3.23 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 JDeveloper, the **Set Property Listener** component is located under **Operations -> Listeners** in the **Components** window (see Figure 14-62).

Figure 14-62 Set Property Listener in Components Window



The following 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, as well as a subset of data visualization components and their child 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:

- Using Event Listeners
- How to Use the Action Listener
- Enabling Gestures
- *Tag Reference for Oracle Mobile Application Framework*

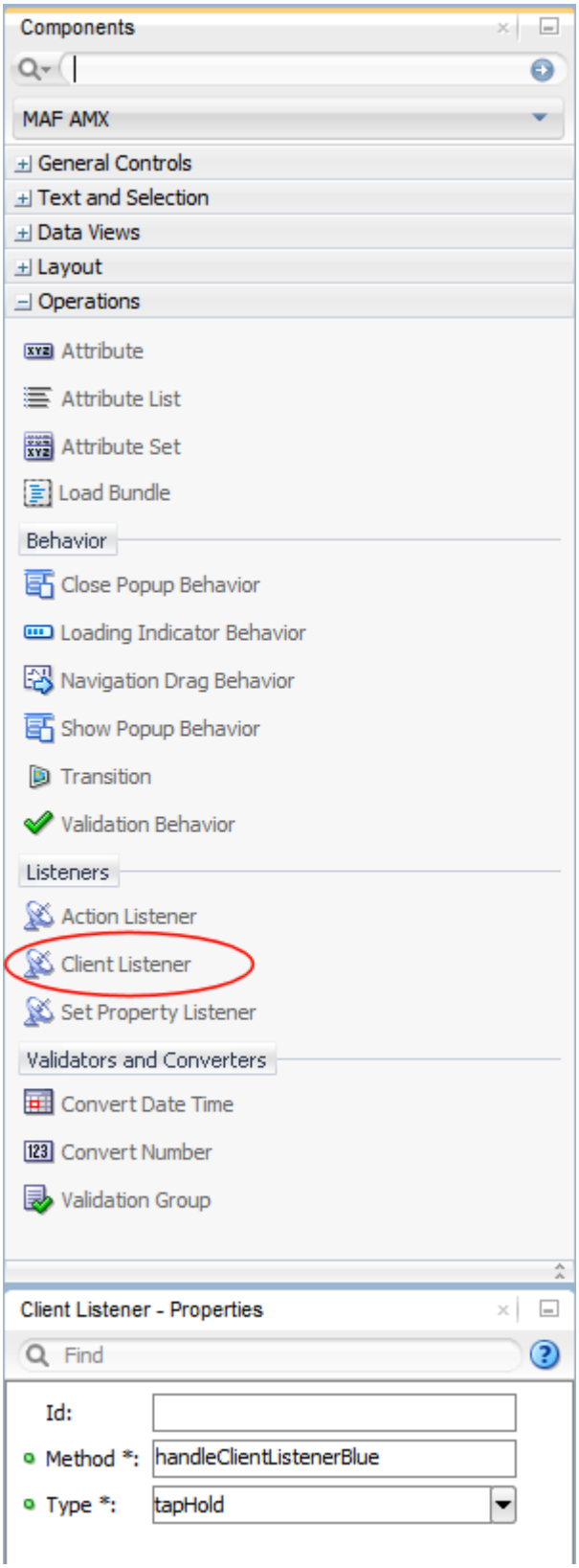
14.3.24 How to Use the Client Listener

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 JDeveloper, the **Client Listener** component is located under **Operations -> Listeners** in the **Components** window (see Figure 14-63).

Figure 14-63 Client Listener in Components Window



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 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:

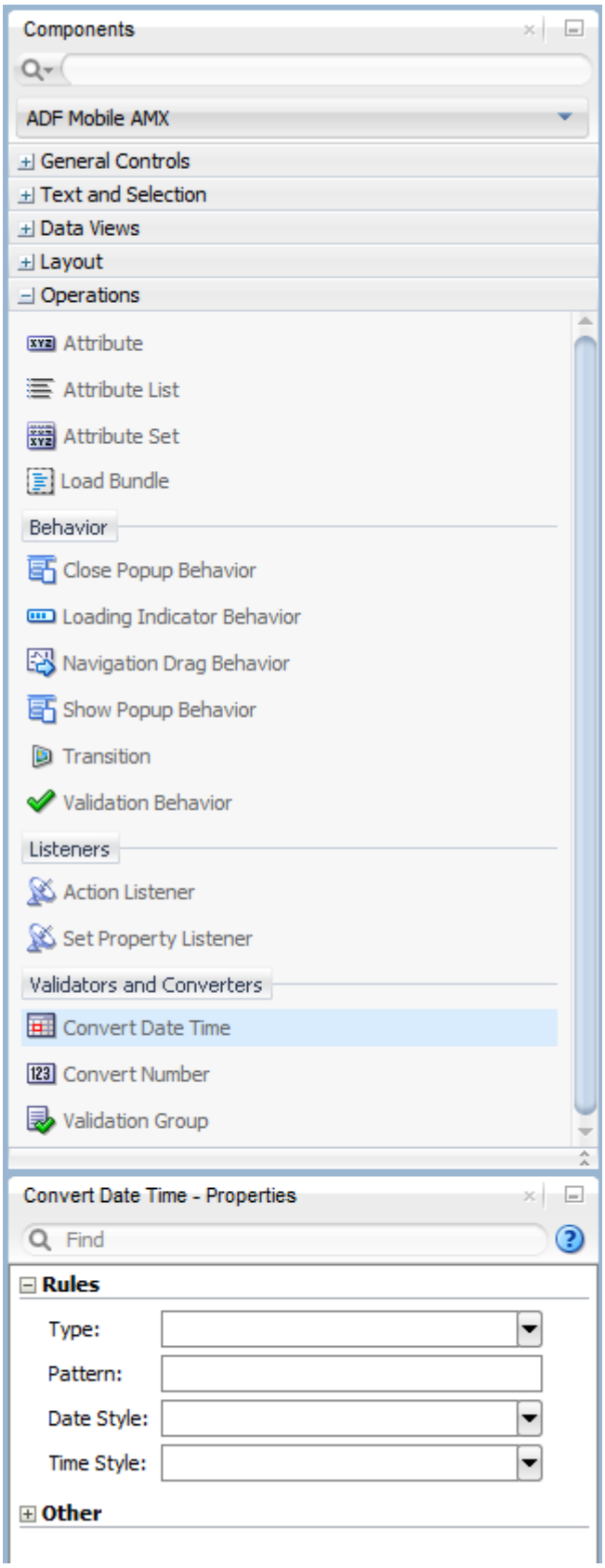
- Using Event Listeners
- *Tag Reference for Oracle Mobile Application Framework*

14.3.25 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 JDeveloper, the **Convert Date Time** is located under **Validators and Converters** in the **Components** window (see Figure 14-64).

Figure 14-64 Convert Date Time in Components Window



The following 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 **Properties** window 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 Time zone 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

14.3.25.1 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 14-9 lists the defined pattern letters (all other characters from `A` to `Z` and from `a` to `z` are reserved).

Table 14-9 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

Table 14-9 (Cont.) Date and Time Pattern Letters

Letter	Date or Time Component	Presentation	Examples
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.

14.3.26 How to Convert Numeric 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.

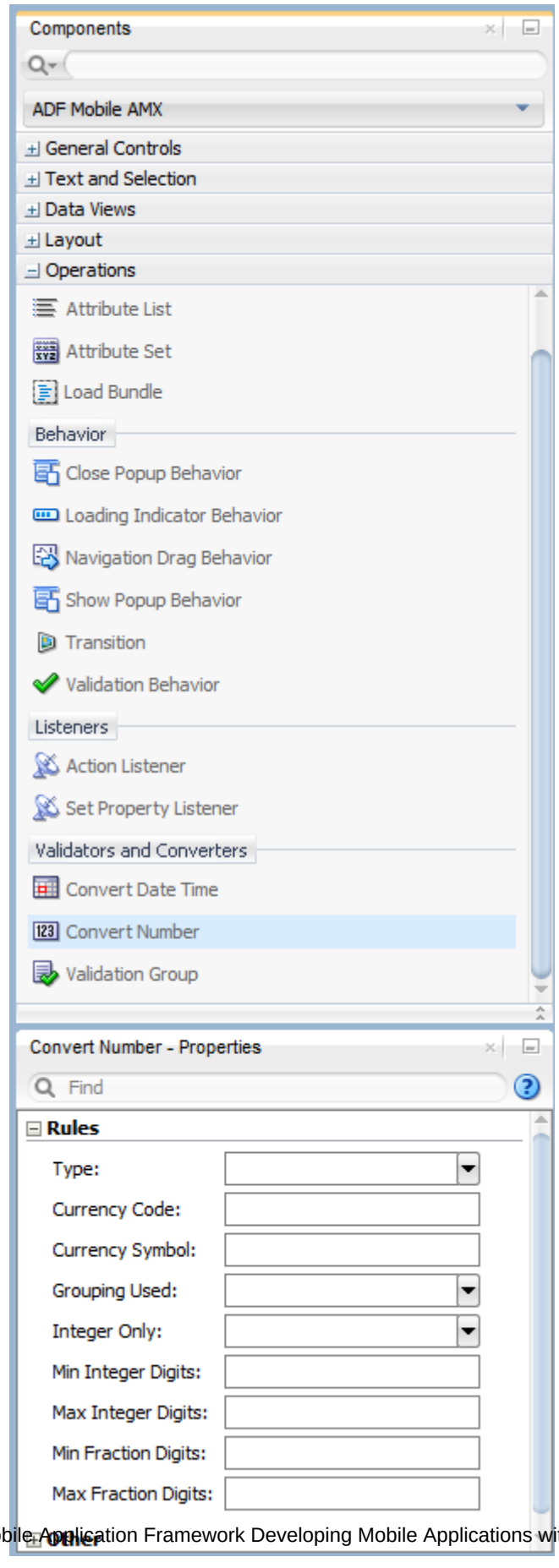
The Convert Number component provides the following types of conversion:

- 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 JDeveloper, the **Convert Number** is located under **Validators and Converters** in the **Components** window (see Figure 14-65).

Figure 14-65 Convert Number in Components Window



The following 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 numeric 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 **Properties** window 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.

14.3.27 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.

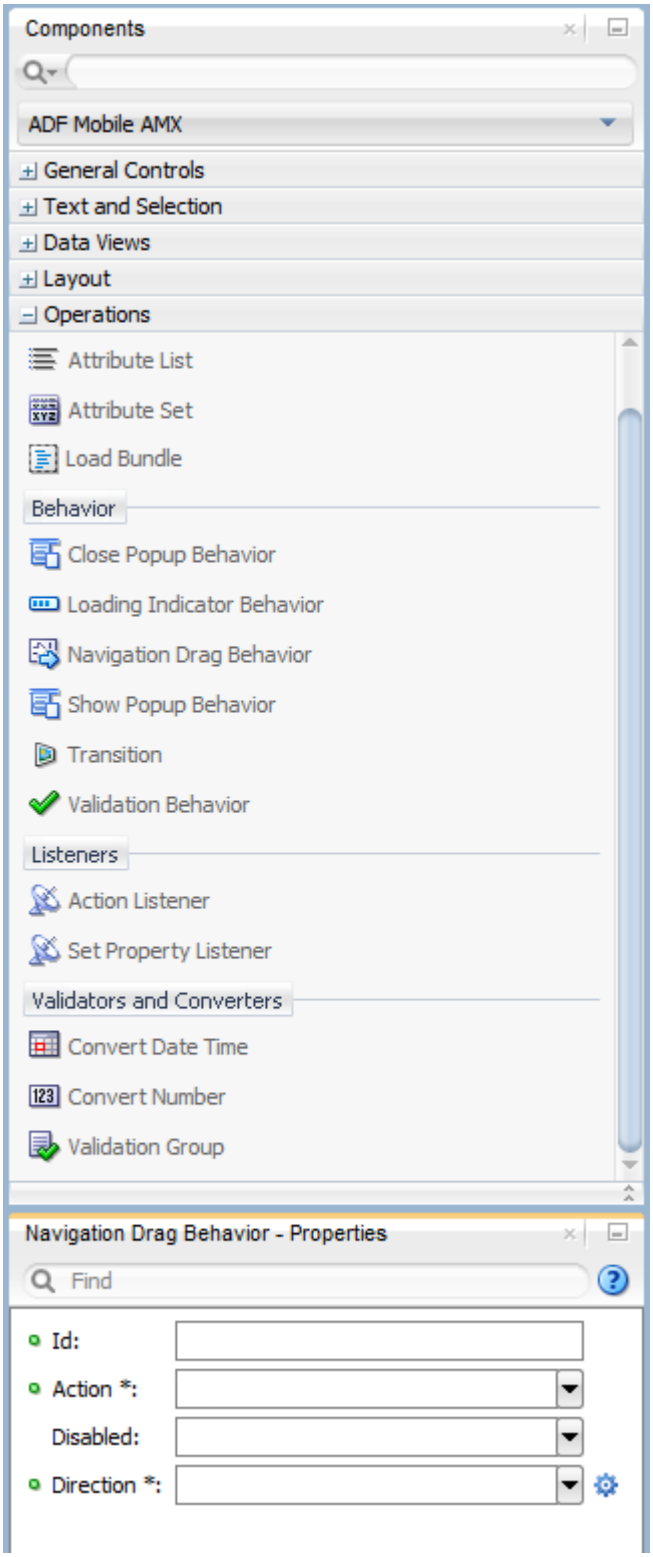
Note:

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 JDeveloper, the **Navigation Drag Behavior** is located under **Operations** in the **Components** window (see Figure 14-66).

Figure 14-66 Navigation Drag Behavior in Components Window



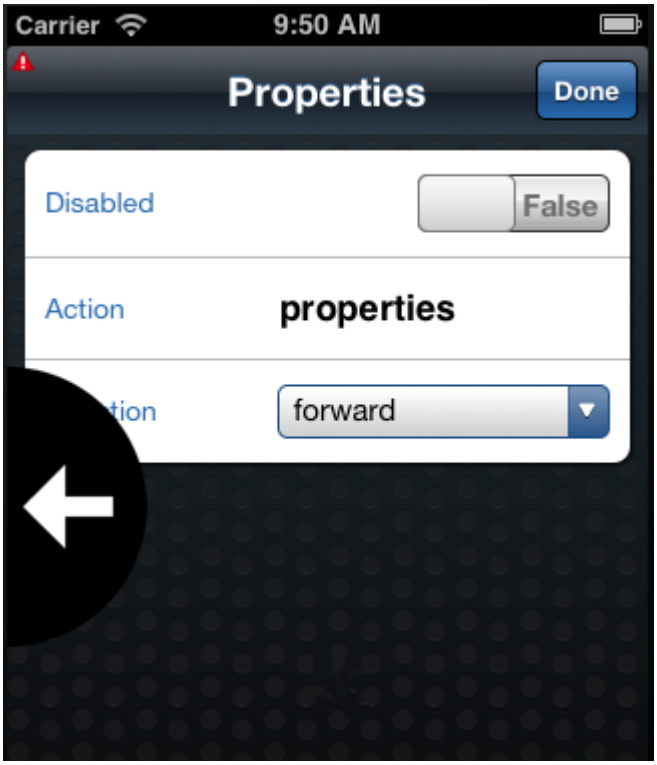
The following example demonstrates the `navigationDragBehavior` element defined in a MAF AMX file. This element can only be a child of the `view` element.

```
<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:navigationDragBehavior id="ndb1"
                               direction="forward"
                               action="gotoDetail"/> [11]
   <amx:panelPage id="pp1">
      <amx:facet name="header">
      ...
   </amx:panelPage>
</amx:view>
```

Figure 14-67 shows the Navigation Drag Behavior at runtime (displayed using the `mobileFusionFx` skin).

Figure 14-67 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.3.27.1 What You May Need to Know About the disabled Attribute

The value of the `disabled` attribute 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

[1] See What You May Need to Know About the disabled Attribute for details.

Behavior, you use the `PropertyChangeListener` (similarly to how it is used with other components that require updates from a bean).

The following example 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 `MyBean` example) from which the `disabled` attribute reads its value. The backing bean, in turn, uses the Property Change Listener.

```
<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 following example shows the backing bean (`myBean` in the `navigationDragBehavior` example) 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);
   }
}
```

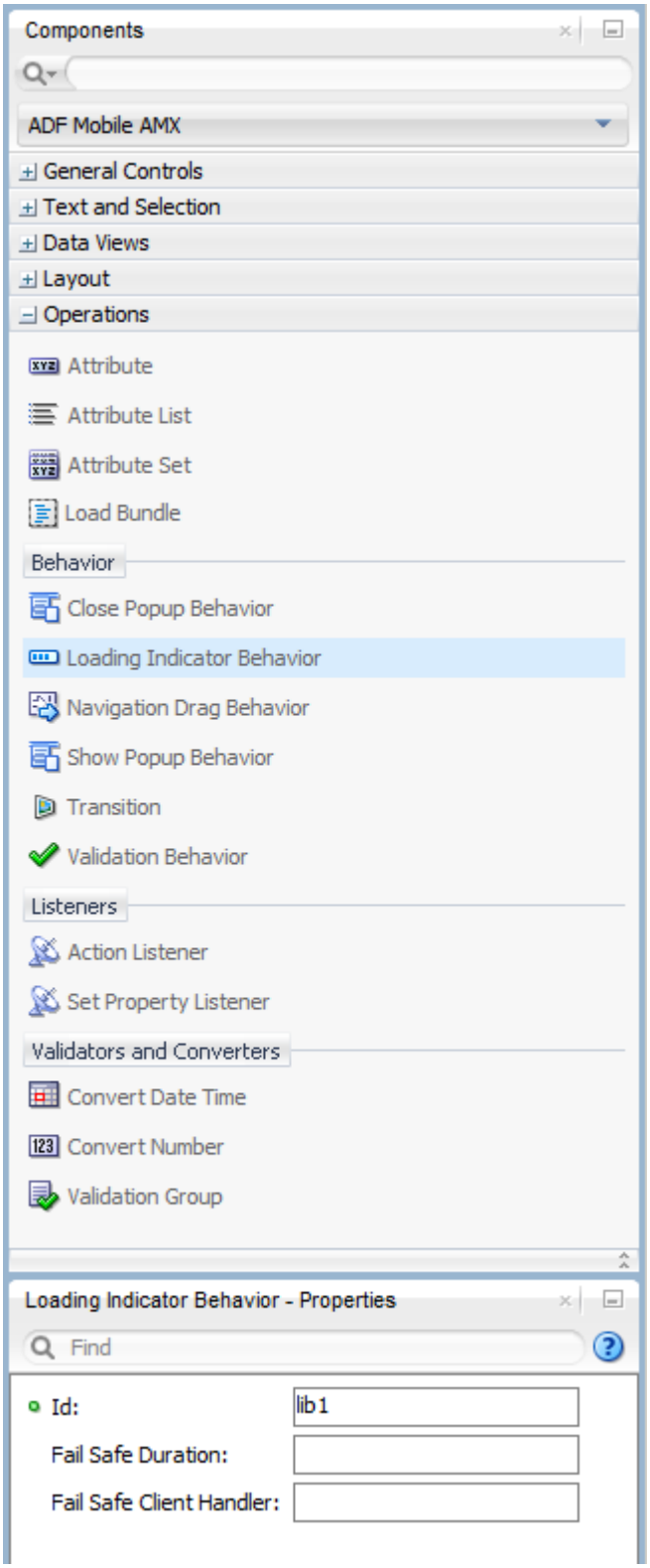
14.3.28 How to Use the Loading Indicator

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.

In JDeveloper, the **Loading Indicator Behavior** is located under **Operations** in the **Components** window (see Figure 14-68).

Figure 14-68 Loading Indicator Behavior in the Components Window



The following example demonstrates the `loadingIndicatorBehavior` 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: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 following example 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 *Tag Reference for Oracle Mobile Application Framework*.

```
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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer

14.4 Creating Custom UI 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 *JSDoc Reference for Oracle Mobile Application Framework*.

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 the example that follows this procedure).
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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer:

   - `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"
   ```

The following 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`, `exampleEvents.amx`, and `exampleList.amx` files included in the CompGallery MAF sample application, described in MAF Sample Applications.

14.5 Enabling Gestures

You can configure Button, Link, List Item, as well as a number of data visualization components to react to end-user gestures.

The following are the gestures:

- Swipe to the right
- Swipe to the left
- Swipe up
- Swipe down
- Tap-and-hold
- Action: as a gesture, Action represents a basic tap.
- Swipe to the start: this gesture is used for accommodating the right-to-left (RTL) 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 for accommodating the right-to-left (RTL) 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`, `action`, and `tapHold` values for the `type` attribute of the following operations:

- Set Property Listener (see How to Use the Set Property Listener)
- Action Listener (see How to Use the Action Listener)
- Show Popup Behavior (see How to Use a Popup Component)
- Close Popup Behavior (see 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 for accommodating the right-to-left (RTL) text direction. It is generally recommended to set the start and end swipe style as opposed to left and right.

The following 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 following example 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*.

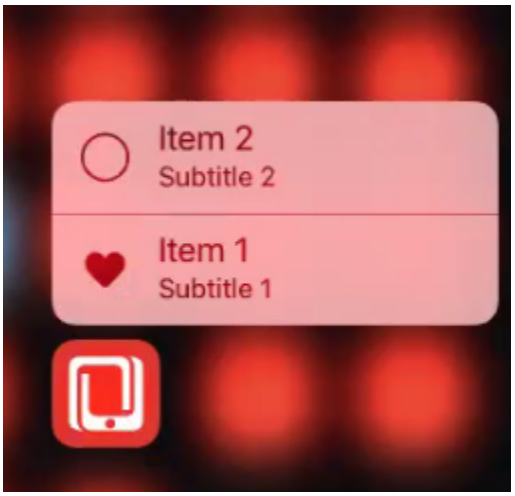
A MAF sample application called GestureDemo demonstrates how to use gestures with a variety of MAF AMX UI components. This sample application is located in the `PublicSamples.zip file` within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

14.6 Implementing Application Shortcuts for Use on iOS Devices with 3D Touch Support

MAF applications that you deploy to iOS devices with 3D Touch support can provide static and dynamic application shortcuts to end users.

3D Touch is an iOS feature where end users can access additional functionality on supported devices by applying varying levels of pressure to the touchscreen. Figure 14-69 shows two application shortcuts that appear when an end user presses the MAF application's icon in the foreground. Once these application shortcuts appear, the end user taps the desired shortcut menu to invoke the associated action (for example, navigate to an application feature different to the MAF application's default application feature).

Figure 14-69 Application Shortcuts on an iOS Device with 3D Touch Support



Implement this functionality by using the following Java APIs that MAF provides:

- `oracle.adfmf.framework.event.ApplicationShortcutEvent`
- `oracle.maf.api.platform.ios.ApplicationShortcutItem`
- `oracle.maf.api.platform.ios.ApplicationShortcutItem.Icon`
- `oracle.maf.api.platform.ios.ApplicationShortcuts`

For more information about these APIs, see the *Java API Reference for Oracle Mobile Application Framework*.

MAF ignores a 3D Touch implementation in a MAF application that you deploy to Android, the Universal Windows Platform, or iOS devices without 3D Touch support.

The following code excerpts demonstrate how you create application shortcuts, register them with the application, listen for the event that occurs when the end user accesses the shortcut, and invoke some logic in response.

```
// Create a list of application shortcuts.
...
import java.util.ArrayList;
import java.util.List;
...

public void createAppShortcuts(ActionEvent e) {
```

```
    List<ApplicationShortcutItem> items = new ArrayList<>();

    // Create a shortcut item
    ApplicationShortcutItem item1 = new ApplicationShortcutItem("type1", "Feature
1");
    item1.setLocalizedSubtitle("Subtitle 1");
    item1.setIcon(ApplicationShortcutItem.Icon.LOVE);
    items.add(item1);

    // Create a second shortcut item. This second shortcut demonstrated how you can
insert user information,
     // perhaps contact information, into a shortcut using the setUserInfo method.
       ApplicationShortcutItem item2 = new ApplicationShortcutItem("type2", "Feature
2");
       item2.setLocalizedSubtitle("go to feature2");
       item2.setIcon(ApplicationShortcutItem.Icon.TASK);
       HashMap<String,Object> map = new HashMap<>();
       map.put("alpha", "beta");
       item2.setUserInfo(map);
       items.add(item2);

...
// Create more shortcut items, if desired, and once complete, configure the
application with these shortcut items.
ApplicationShortcuts.setShortcutItems(items);
}

// Clear previously-created application shortcuts by passing an empty list to the
setShortcutItems method
public void clearShortcuts(ActionEvent e)
    {
    System.out.println("In clearShortcuts");
    List<ApplicationShortcutItem> items = new ArrayList<>();
    ApplicationShortcuts.setShortcutItems(items);
    }


// Listen for the event where end user invokes the application shortcut
// Consider registering this listener in your application's LifeCycleListener start
method, for example.
EventSourceFactory.getEventSource(EventSourceFactory.APPLICATION_SHORTCUT_EVENT_SOURC
E_NAME).addListener(new ShortcutEventListener());
...
    private final class ShortcutEventListener implements EventListener
      {
        @Override
        public void onMessage(Event event)
        {
          ApplicationShortcutItem item = (ApplicationShortcutItem)
event.getPayloadObject();
          AdfmfJavaUtilities.setELValue("#{applicationScope.applicationShortcut}",
item.toString());

// Once you get the event payload, decide what to do. The following example assumes
that the shortcut(s) are
// implemented to navigate end users to application features within the MAF
application.

     if ("type1".equals(item.getType()))
     AdfmfContainerUtilities.gotoFeature("feature1");
```

. . . .

14.7 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:

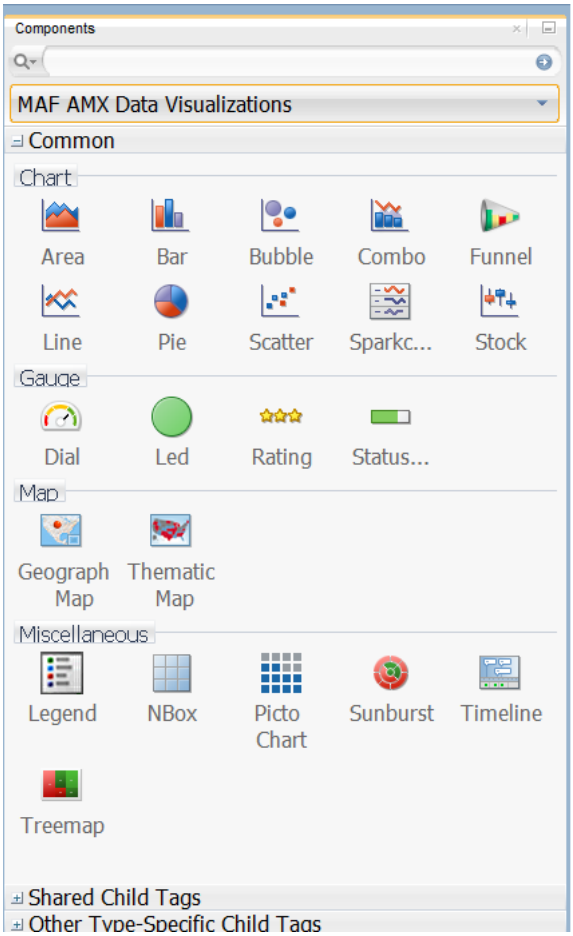
- `areaChart` (see How to Create an Area Chart)
- `barChart` (see How to Create a Bar Chart)
- `bubbleChart` (see How to Create a Bubble Chart)
- `comboChart` (see How to Create a Combo Chart)
- `lineChart` (see How to Create a Line Chart)
- `pieChart` (see How to Create a Pie Chart)
- `scatterChart` (see How to Create a Scatter Chart)
- `sparkChart` (see How to Create a Spark Chart)
- `funnelChart` (see How to Create a Funnel Chart)
- `stockChart` (see How to Create a Stock Chart)
- `ledGauge` (see How to Create a LED Gauge)
- `statusMeterGauge` (see How to Create a Status Meter Gauge)
- `dialGauge` (see How to Create a Dial Gauge)
- `ratingGauge` (see How to Create a Rating Gauge)
- `geographicMap` (see How to Create a Geographic Map Component)
- `thematicMap` (see How to Create a Thematic Map Component)
- `treemap` (see How to Create a Treemap Component)
- `sunburst` (see How to Create a Sunburst Component)
- `timeline` (see How to Create a Timeline Component)
- `nBox` (see How to Create an NBox Component)
- `pictoChart` (see How to Create a Picto Chart)

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 JDeveloper, data visualization components are located as follows in the **Components** window:

- Chart components are located under **MAF AMX Data Visualizations > Common > Chart**
- Gauge components are located under **MAF AMX Data Visualizations > Common > Gauge**
- Map components are located under **MAF AMX Data Visualizations > Common > Map**
- Treemap, Sunburst, Timeline, and NBox are located under **MAF AMX Data Visualizations > Common > Miscellaneous**

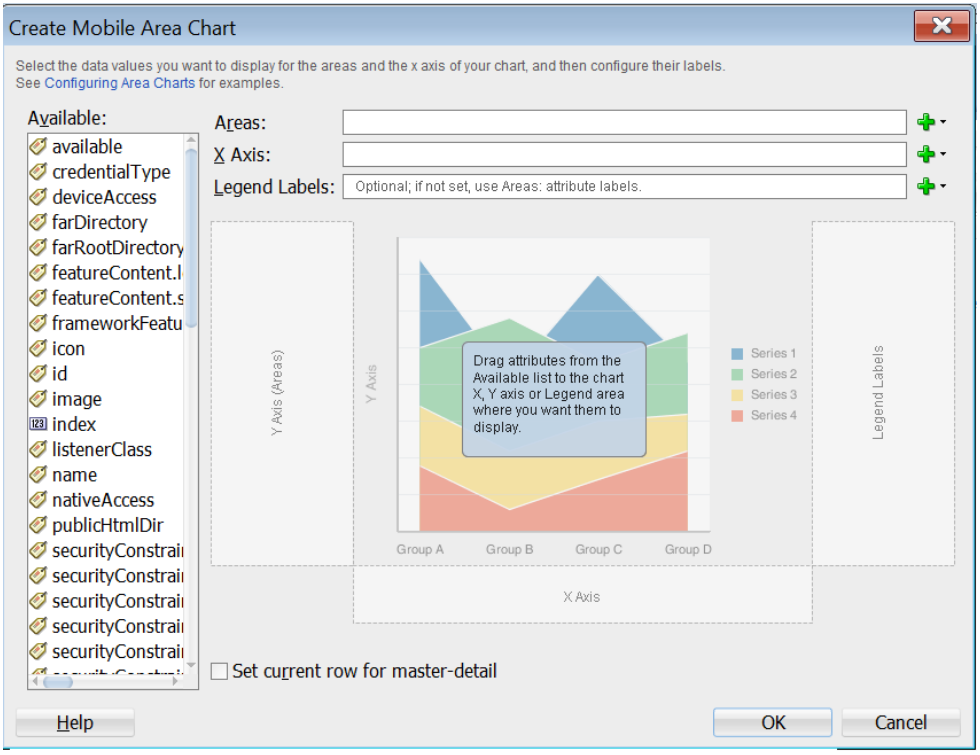
Figure 14-70 Data Visualization Components in the Components Window



When you drag and drop a data visualization component, a dialog similar to one of the following opens to display the information about the type of component you are creating:

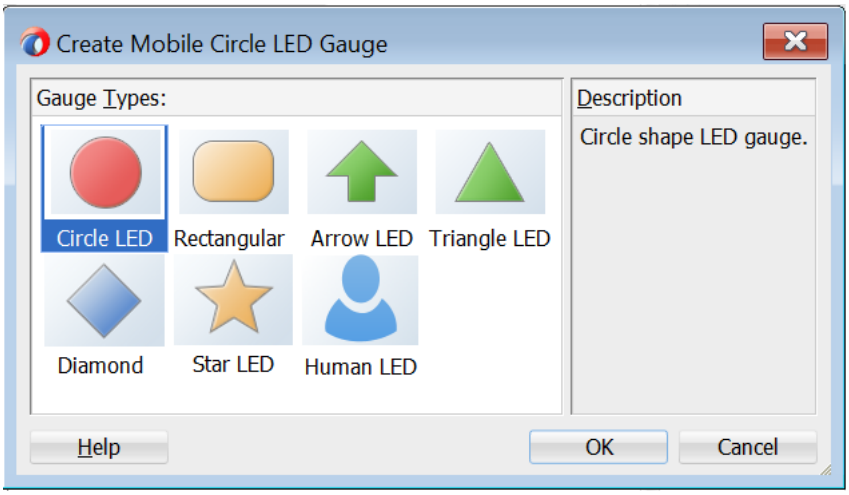
- **Create Mobile Chart** (see Figure 14-71)

Figure 14-71 Creating Chart Components



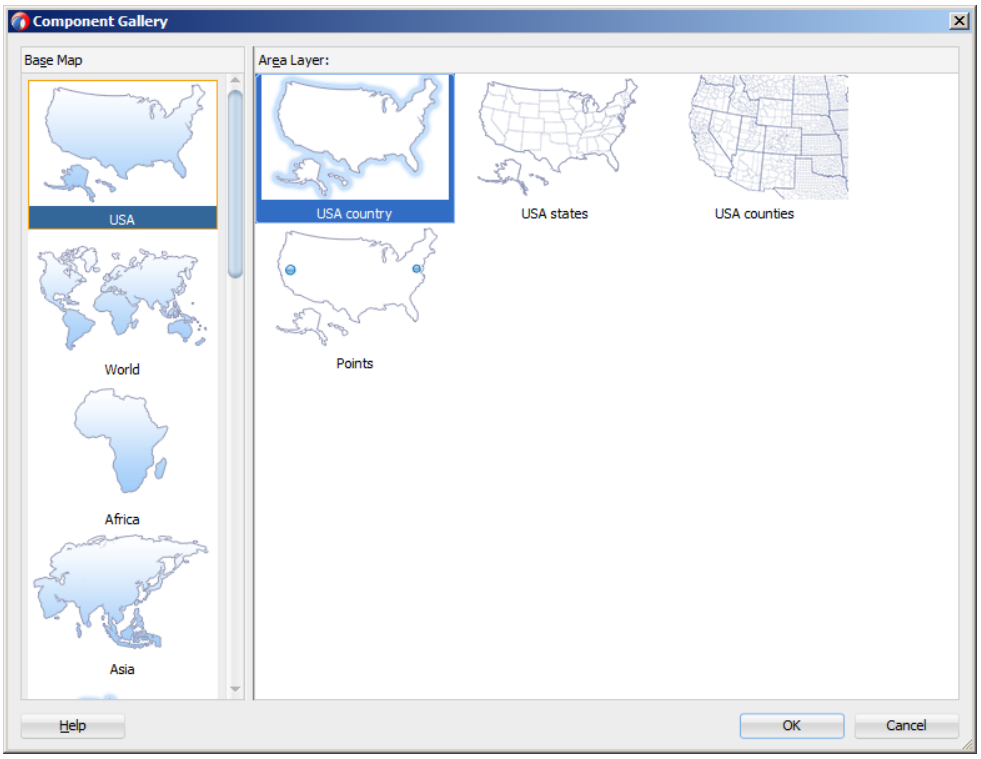
- **Create Mobile Gauge** (see Figure 14-72)

Figure 14-72 Creating Gauge Components



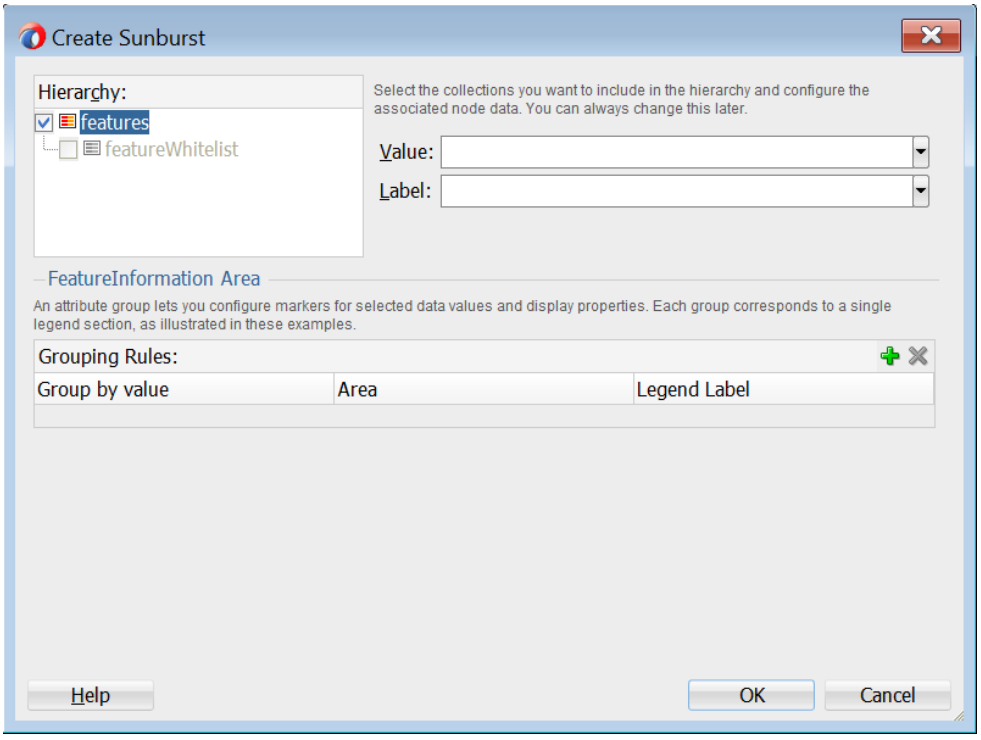
- **Component Gallery** (see Figure 14-73)

Figure 14-73 Creating Map Components



- **Create Sunburst** (see Figure 14-74)

Figure 14-74 Creating Sunburst



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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

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 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 How to Create Databound Data Visualization Components.
- For information on providing static data for charts and other data visualization components, see How to Create Data Visualization Components Based on Static Data.
- For information on chart components' interactivity, see How to Enable Interactivity in Chart Components.
- For information on creating polar charts, see How to Create Polar Charts.
- For information on data visualization components' support for accessibility, see Understanding MAF Support for Accessibility.

14.7.1 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 How to Create a Line Chart.

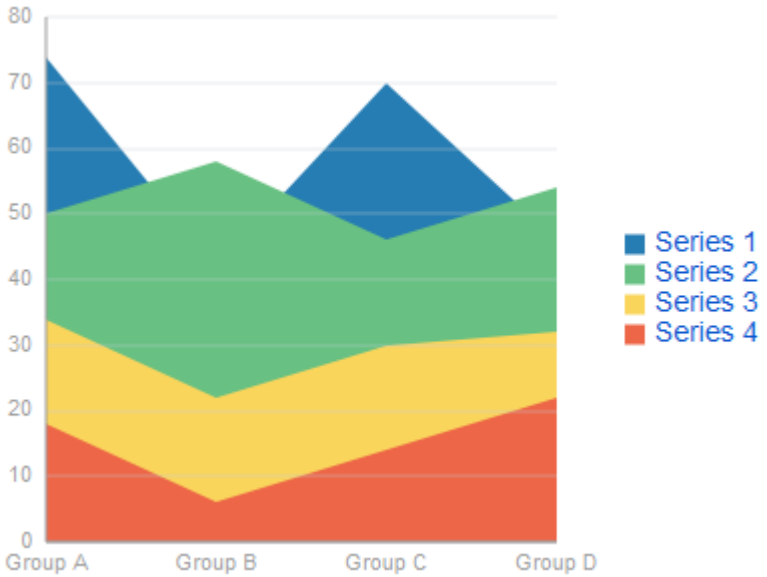
The Area Chart can be zoomed and scrolled along its X Axis. This is enabled through the use of the `zoomAndScroll` attribute.

The 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 14-75 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 `markerShape`, 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 following example shows. In this case, the `chartDataItem`'s `color` and `markerShape` attributes are set based on the additional grouping expression.

The `attributeGroups` settings can be shared between data visualization components and attribute values can be automatically applied across these components. You enable this functionality by setting the `discriminant` attribute of the `attributeGroups`: components with the same `discriminant` value share their

settings, including value of the `attributeMatchRule` child element of their `attributeGroups`.

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:

As the preceding example and Figure 14-75 show, since custom styles are not set at the series level, series are displayed with the colors based on the default color ramp.

The `orientation` attribute allows you to define the Area Chart as either horizontal or vertical.

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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.2 How to Create a Bar Chart

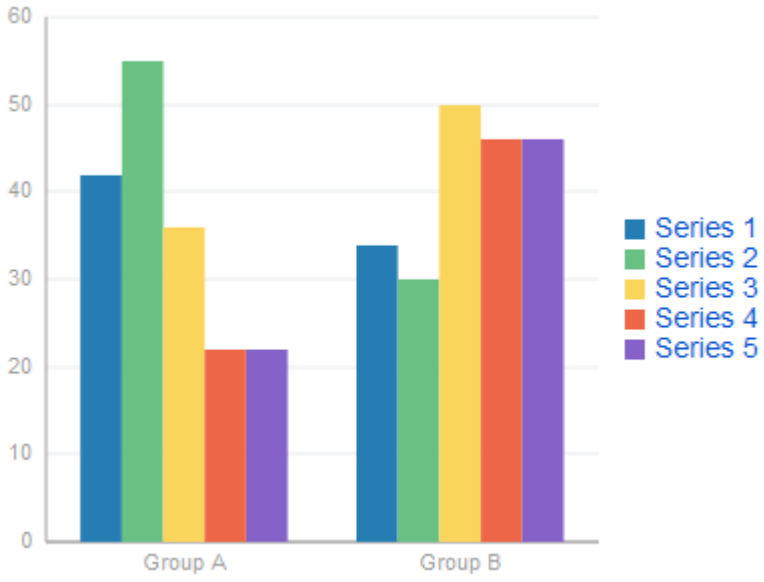
You use a Bar Chart (`barChart`) to visually display data as vertical bars, where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties that you have to apply at the series level instead of per each 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.

The following 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 14-76 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`.

The `orientation` attribute allows you to define the Bar Chart as either horizontal or vertical.

By setting the `z` attribute in addition to the `x` and `y` attributes of the `chartDataItem`, you can enable the bar widths to behave as a third dimension. This is useful when describing discrete data points where each bar carries a different weight.

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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

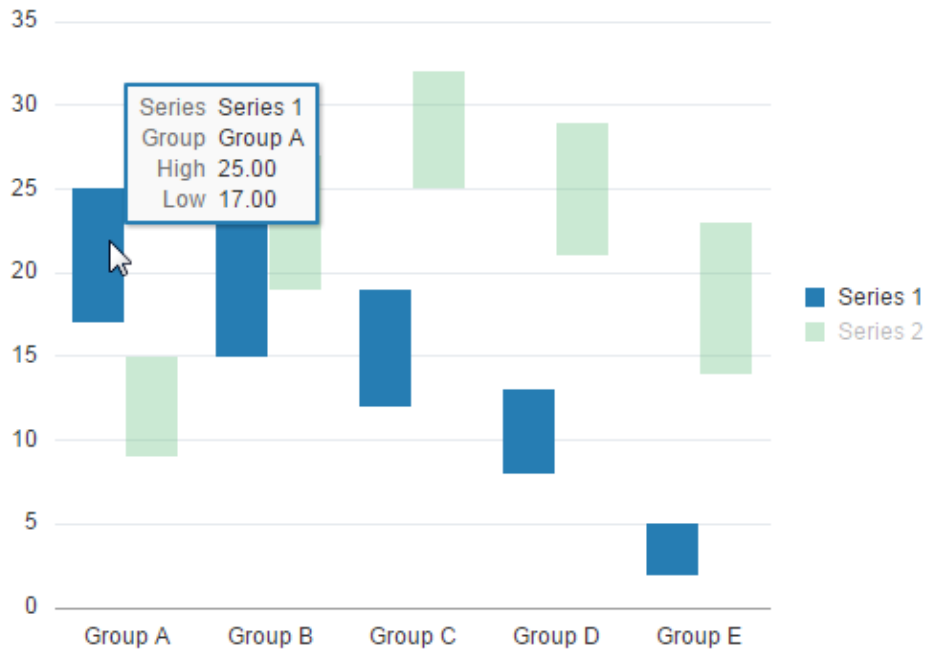
14.7.3 How to Create a Range Chart

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 14-77 shows an example of a range chart rendered by the bar chart component.

```
 <dvtm:areaChart var="row" value="#{bindings.rangeData.collectionModel}" id="ac1">
      <amx:facet name="dataStamp">
        <dvtm:chartDataItem group="#{row.group}" series="#{row.series}"
low="#{row.low}" high="#{row.high}" id="cdi1"/>
      </amx:facet>
      <dvtm:legend position="end" id="l1"/>
    </dvtm:areaChart>
```

Figure 14-77 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, see:

- How to Create a Bar Chart
- How to Create an Area Chart
- *Tag Reference for Oracle Mobile Application Framework*

14.7.4 How to Create a Bubble Chart

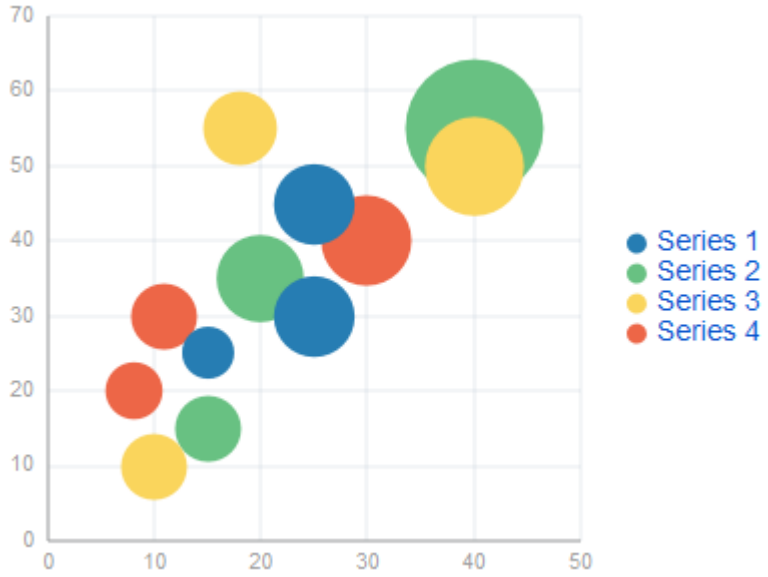
A Bubble Chart (`bubbleChart`) displays a set of data items where each data item has `x`, `y` coordinates and size (bubble). In addition, each data item can have various style attributes, such as `color` and `markerShape`. You can either set properties of each data item individually, or categorize the 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 How to Create a Line Chart) or area charts (see 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 following example 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 14-78 Bubble Chart at Design Time



In the following 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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.5 How to Create a Combo Chart

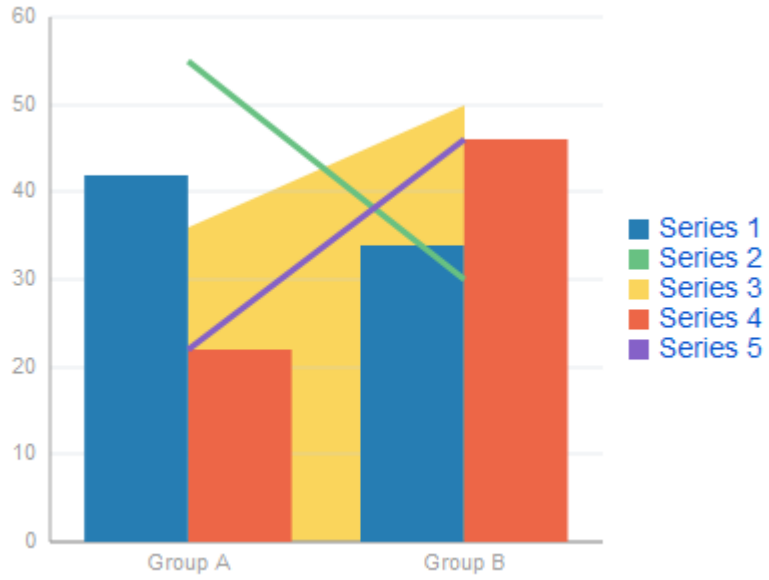
A Combo Chart (`comboChart`) represents an overlay of two or more different charts, such as a line and bar chart.

The following example 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 14-79 Combo Chart at Design Time



The `orientation` attribute allows you to define the Combo Chart as either horizontal or vertical.

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*.

You can define a `facet` child element from the `amx` namespace. The `facet` can have a `chartDataItem` as its child (see 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.6 How to Create a Line Chart

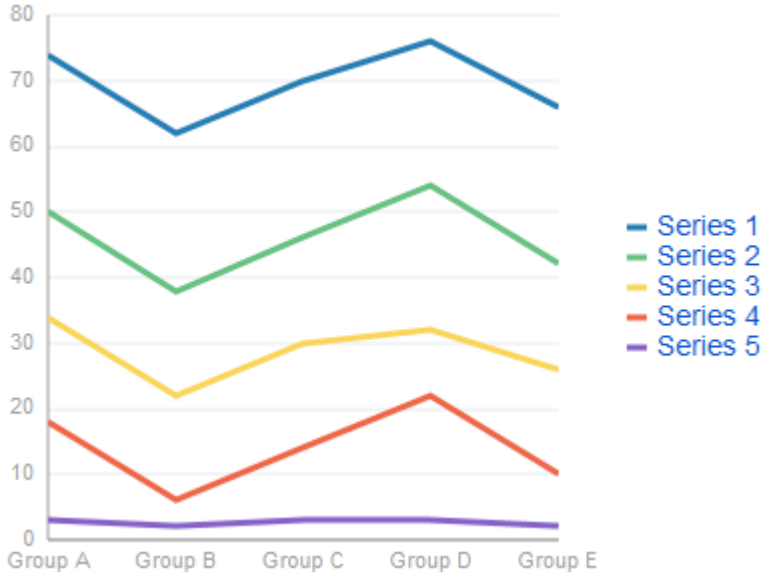
You use the Line Chart (`lineChart`) to visually represent data where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties (such as, for example, a line color, width, or style). Those properties have 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 14-80 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 `markerShape`, 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 following example 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 preceding examples, 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 the following 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}" />
   </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 the following 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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

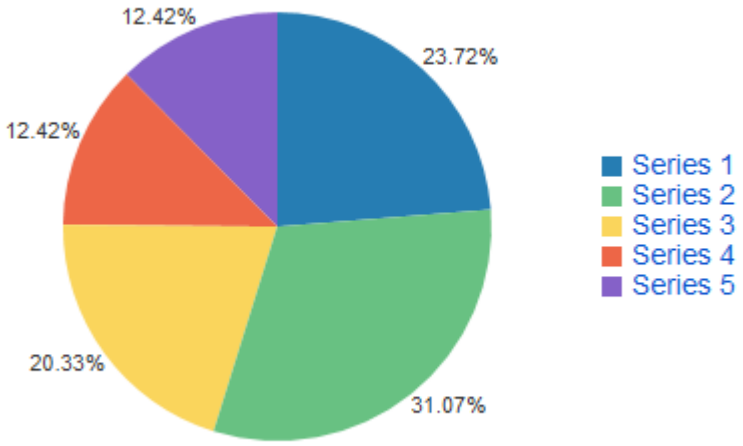
14.7.7 How to Create a Pie Chart

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.

The following 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>
```

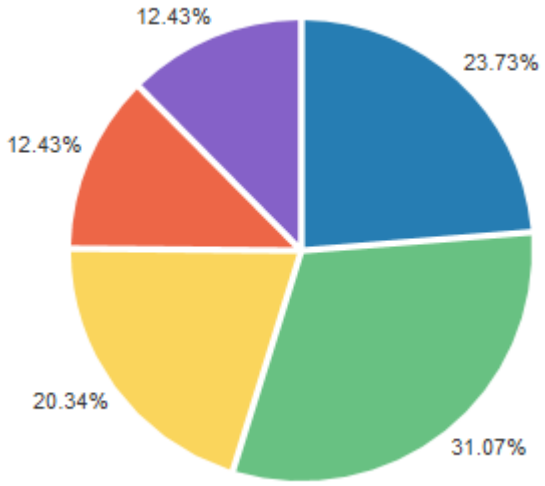
Figure 14-81 Pie Chart at Design Time



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*.

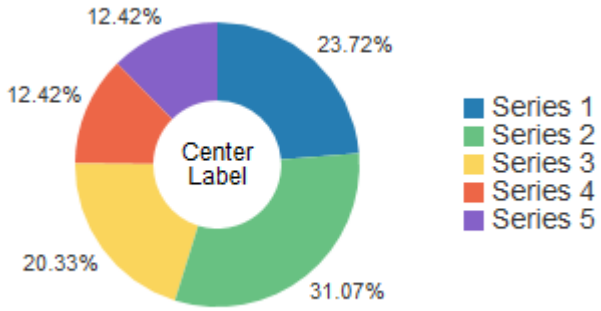
You can define a `facet` child element from the `amx` namespace. The `facet` can have a `pieDataItem` as its child (see Defining Pie Data Item).

14.7.7.1 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 14-82 Ring Chart at Design Time



14.7.7.2 Styling the Pie Chart

You can style the Pie Chart component by overwriting the default CSS settings defined in `dvtm-pieChart`, `dvtm-chartPieLabel`, `dvtm-chartPieCenterLabel`, and `dvtm-chartSliceLabel` classes:

- The top-level element can be styled using

```
.dvtm-pieChart
   - supported properties: all
```

- The pie labels can 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
```

- The ring center label can be styled using

```
.dvtm-chartPieCenterLabel
   - supported properties:
               font-family, font-size, font-weight, color, font-style
```

For more information on chart styling, see How to Style Chart Components.

For more information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.8 How to Create a Scatter Chart

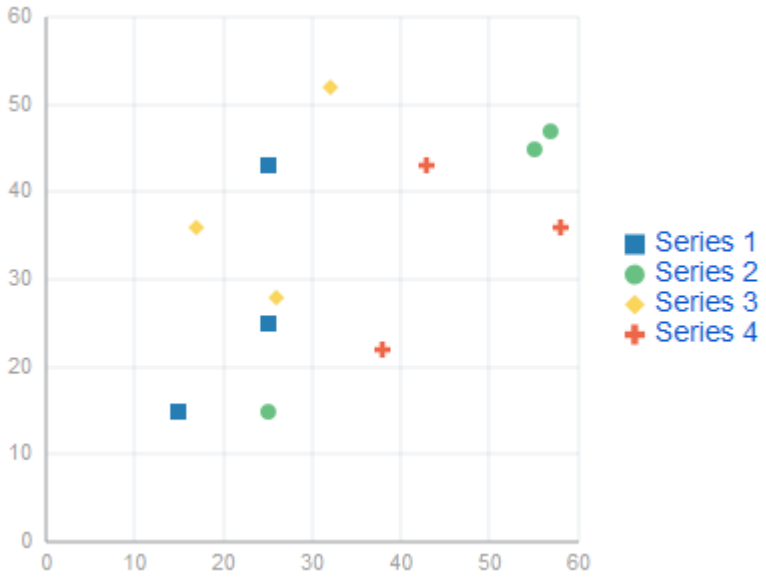
A Scatter Chart (`scatterChart`) displays data as unconnected dots that represent data items, where each item has `x`, `y` coordinates and size. In addition, each data item can have various style attributes, such as `color` and `markerShape`. You can either set properties of each data item individually, 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 How to Create a Line Chart) or area charts (see 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 following example 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 14-83 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.

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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.9 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.

The following 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 14-84 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 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 How to Style Chart Components.

For information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.10 How to Create a Funnel Chart

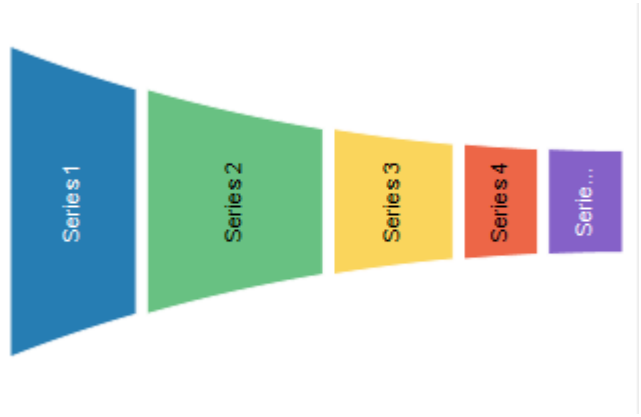
A Funnel Chart (`funnelChart`) component provides a visual representation of data related to steps in a process. The steps appear as vertical slices across a horizontal cylinder. As the actual value for a given step or slice approaches the quota for that slice, the slice fills. Typically, a Funnel Chart requires actual values and target values against a stage value, which might be time.

The following 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 14-85 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 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 How to Style Chart Components.

For more information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.11 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.

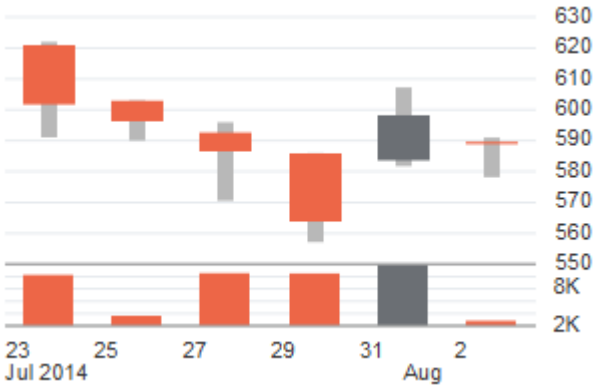
The following example shows the `stockChart` element defined in a MAF AMX file. The `dataStamp` facet contains a `stockDataItem` element.

```
<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 14-86 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 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 How to Style Chart Components.

For more information on how to extend CSS files, see How to Style Data Visualization Components.

14.7.12 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.

14.7.13 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, 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:

- Using Event Listeners
- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*

14.7.14 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*.

14.7.15 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.

14.7.16 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.

14.7.17 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.

14.7.18 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.

14.7.19 How to Create a 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.

The following 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 14-87 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 How to Use a Popup Component)
- `closePopupBehavior` (see How to Use a Popup Component)
- `validationBehavior` (see Validating Input)

14.7.20 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.

The following 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 14-88 Rectangular Status Meter Gauge at Design Time

To create a Status Meter Gauge represented by a vertical rectangle, you set its `orientation` attribute to `vertical`. 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 14-89), you set its `orientation` attribute to `circular`.

Figure 14-89 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 How to Use a Popup Component)
- `closePopupBehavior` (see How to Use a Popup Component)
- `validationBehavior` (see Validating Input)

14.7.21 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.

The following 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 14-90 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 following example 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 14-91 Dial Gauge with Metric and Tick Labels at Design Time

You can define the following `amx` child elements for the `dialGauge`:

- `showPopupBehavior` (see How to Use a Popup Component)
- `closePopupBehavior` (see How to Use a Popup Component)
- `validationBehavior` (see Validating Input)

14.7.22 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.

The following 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 14-92 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 How to Use a Popup Component)
- `closePopupBehavior` (see How to Use a Popup Component)
- `validationBehavior` (see Validating Input)

14.7.22.1 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>
```

14.7.22.2 Applying Custom Styling to the Rating Gauge Component

Depending on the action performed by the end 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 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;
}
```

14.7.23 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 Defining Chart Data Item)
- `xAxis`, `yAxis`, and `y2Axis` (see Defining and Configuring X Axis, YAxis, and Y2Axis)

- `legend` (see Defining and Configuring Legend)
- `pieDataItem` (see Defining Pie Data Item)
- `sparkDataItem` (see Defining Spark Data Item)
- `threshold` (see Defining Threshold)
- `funnelDataItem` (see Defining Funnel Data Item)
- `stockDataItem` (see Defining Stock Data Item)

For more information on these and other child elements, see *Tag Reference for Oracle Mobile Application Framework*.

In JDeveloper, child components of data visualization components are located under **MAF AMX Data Visualization > Shared Child Tags** and **MAF AMX Data Visualization > Other Type-Specific Child Tags** in the **Components** window (see Figure 14-93).

Figure 14-93 Creating Chart and Gauge Child Components

14.7.23.1 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*.

14.7.23.2 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 more information on attributes of the `legend` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.3 Defining and Configuring 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 more information on attributes and child elements of `xAxis`, `yAxis`, and `y2Axis` elements, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.4 Defining Pie Data Item

The Pie Data Item (`pieDataItem`) element specifies the parameters of the pie chart slices (see How to Create a Pie Chart).

For information on attributes of the `pieDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.5 Defining Spark Data Item

The Spark Data Item (`sparkDataItem`) element specifies the parameters of the spark chart items (see How to Create a Spark Chart).

For information on attributes of the `sparkDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.6 Defining Funnel Data Item

The Funnel Data Item (`funnelDataItem`) element specifies the parameters of the funnel chart items (see How to Create a Funnel Chart).

For information on attributes of the `funnelDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.7 Defining Stock Data Item

The Stock Data Item (`stockDataItem`) element specifies the parameters of the stock chart items (see How to Create a Stock Chart).

For information on attributes of the `stockDataItem` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.23.8 Defining Threshold

The Threshold (`threshold`) element specifies the threshold ranges of a gauge (see How to Create a LED Gauge and How to Create a Status Meter Gauge).

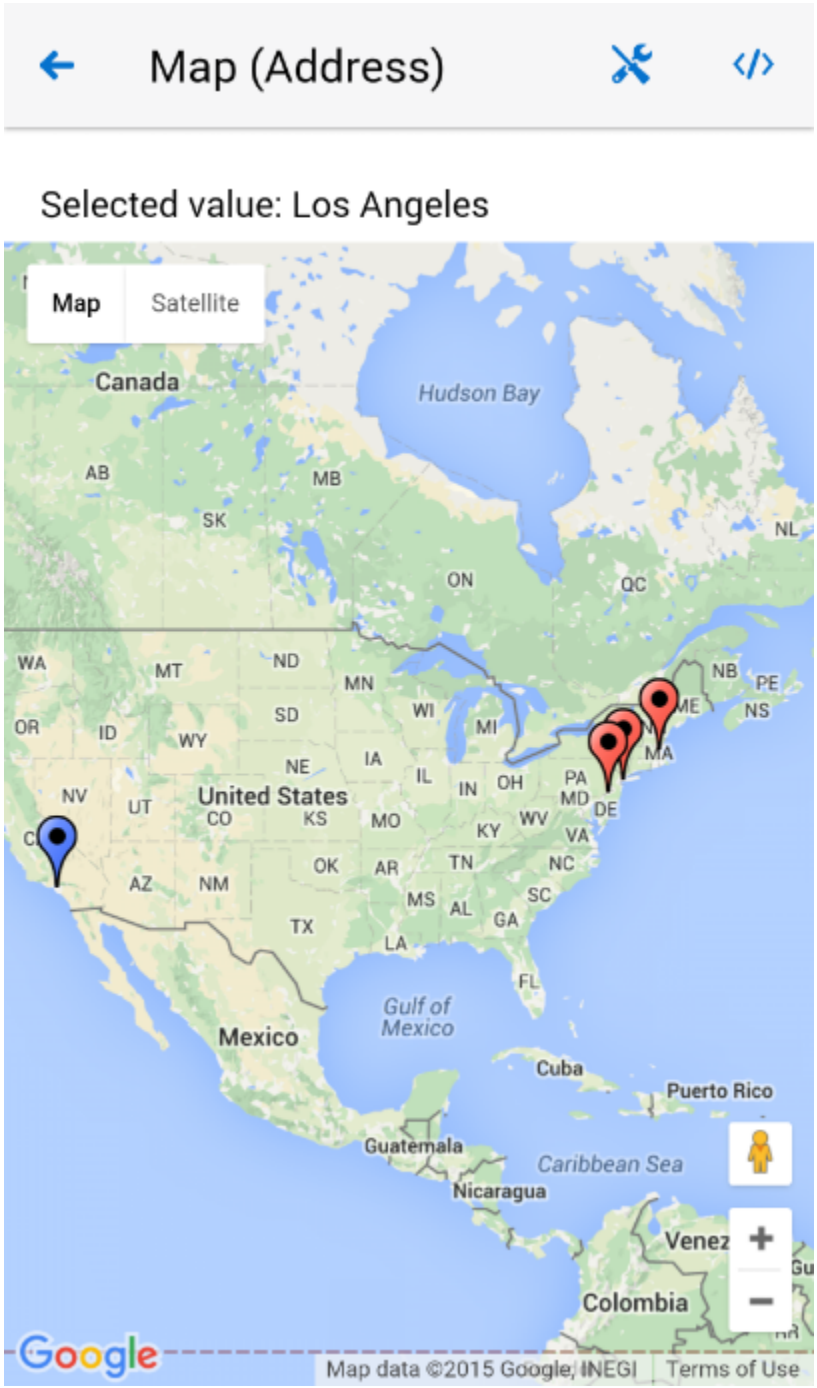
For information on attributes of the `threshold` element, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.24 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 as the underlying map provider. If you do not specify a map provider, the component uses Google Maps.

Figure 14-94 shows the `geographicMap` component in the CompGallery sample application.

Figure 14-94 Geographic Map in CompGallery Sample Application



The `geographicMap` component uses the latest stable v3 version of the Google Maps JavaScript API.

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 How to Convert Numeric Values). In addition, you can enable a Popup (see 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 `geographicMap` 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 How to Use Events with Map Components.

For more information about related tasks with the `geographicMap` component, see:

- Configuring Geographic Map Components With the Map Provider Information
- Displaying Routes in Geographic Map Components

14.7.24.1 Configuring Geographic Map Components With the Map Provider Information

To configure a `geographicMap` 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 Cloud Service.
- `baseMap`: if the `mapProvider` is set to `oraclemaps`, specify the base map to use with Oracle Maps Cloud Service.

Note:

To configure the `geographicMap` component to use Google Maps, you must obtain an appropriate license from Google.

The following 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 14-10

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:

When using the Oracle Maps Cloud Service, you must abide by the Terms of Use and also abide by the Supplier Notices. Oracle Maps Cloud Service is provided under the terms of the "Oracle Maps Cloud Service Enterprise Hosting and Delivery Policies" described in http://www.oracle.com/us/corporate/contracts/maps-cloud-hd-policies-2767907.pdf.

The following example shows the configuration for Oracle Maps Cloud Service.

```
<adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="mapProvider" value="oraclemaps"/>
   <adf-property name="mapViewerUrl" value="your-mapviewer-server-url"/>
   <adf-property name="baseMap" value="your-basemap-id"/>
</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 Cloud Service on an iOS 9 device:

1. Disable ATS when you deploy the MAF application. For more information, see Defining the iOS Build Options. This option is not recommended.

2. Configure the Oracle Maps Cloud Service 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 Cloud Service on an iOS 9 device.

   ```
   <adf:adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
       <adf-property name="mapProvider" value="oracleMaps"/>

        <!-- Configure the URL for the mapviewer service that provides map data-->
       <adf-property name="mapViewerUrl" value="https://elocation.myserver.com/
   mapviewer"/>

       <!-- Configure the URL for the eLocation service that provides geocoding
   services, such as address resolution,
            directions, and so on -->
       <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 About the Application Controller Project-Level Resources.

14.7.24.2 Displaying Routes in Geographic Map Components

You can display routes between two points, with possible waypoints, in the `geographicMap` component by adding a Route (`route`) child component. Google Maps or Oracle Maps Cloud Service can both be used as a provider to implement this use case.

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 Using Event Listeners.

14.7.25 How to Create a Thematic Map Component

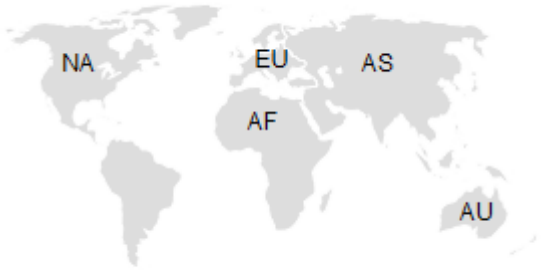
A Thematic Map (`thematicMap`) represents business data as patterns in stylized areas or associated markers. Thematic maps focus on data without the geographic details.

The following example 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 14-95 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 Defining Custom Markers)
- Area isolation (see Defining Isolated Areas)
- Initial zooming (see Enabling Initial Zooming
- Custom base maps (see 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*.

14.7.25.1 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.

14.7.25.2 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.

14.7.25.3 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.

14.7.25.4 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.

14.7.25.5 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: 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>
```

In the preceding example, the base map ID is matched with the `basemap` attribute of the `thematicMap`, and the layer ID is matched with the `layer` attribute of the `areaLayer`. The points are defined through the X and Y coordinates (just like for a predefined base map) to accommodate dynamic points that can change at the time the data are updated.

The following example shows an alternative way to declare a custom area layer with points. In this example, the `pointDataLayer` is a direct child of the `thematicMap`. Despite this variation, it renders the same result as the declaration demonstrated in preceding example.

```
<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="pointXY"
                          pointX="#{row.x}"
                          pointY="#{row.y}" >
         <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
      </dvtm:pointLocation>
   </dvtm:pointDataLayer>
</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>
```

14.7.25.6 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.

14.7.25.7 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 How to Style Data Visualization Components.

The following 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 following example 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. For information on how to add custom CSS and JavaScript files to your application, see Defining the Application Feature Content as a MAF AMX Page or Task Flow.

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 following 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']
   }
};
```

14.7.26 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 Map component. 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:

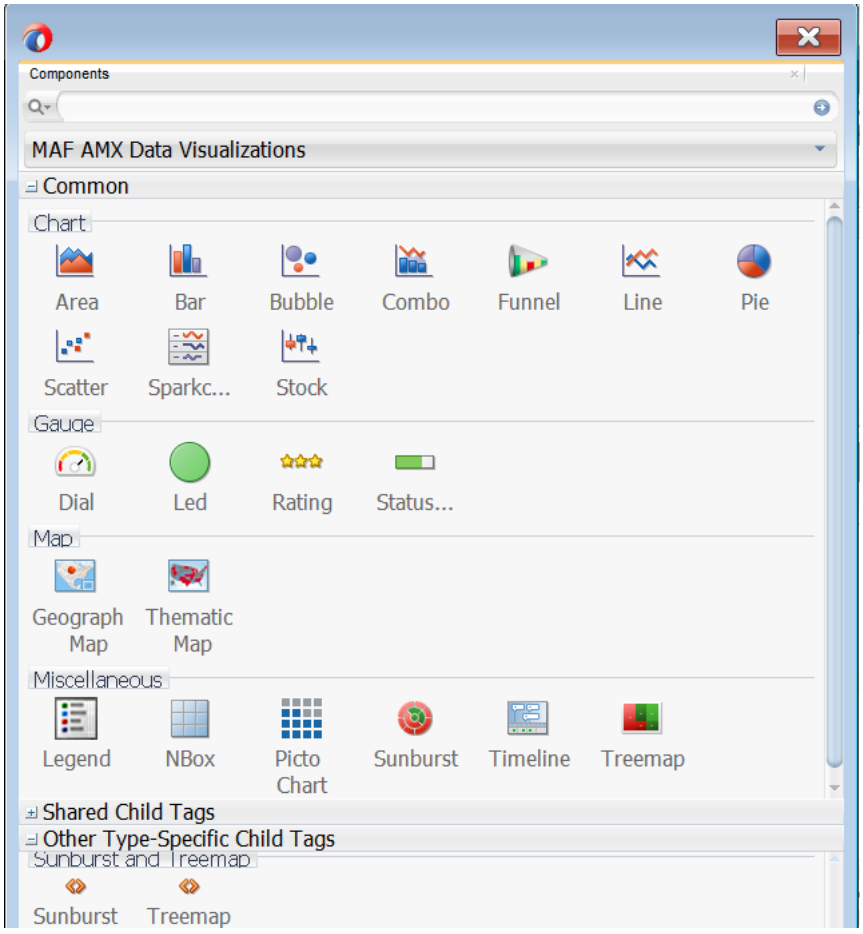
- Using Event Listeners
- *Java API Reference for Oracle Mobile Application Framework*
- *Tag Reference for Oracle Mobile Application Framework*

14.7.27 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 **Components** window, the Treemap is located under **MAF AMX Data Visualizations > Common > Miscellaneous**, and its child components are located under **MAF AMX Data Visualizations > Other Type-Specific Child Tags > Sunburst and Treemap** and **MAF AMX Data Visualizations > Shared Child Tags** (see Figure 14-96).

Figure 14-96 Treemap Component

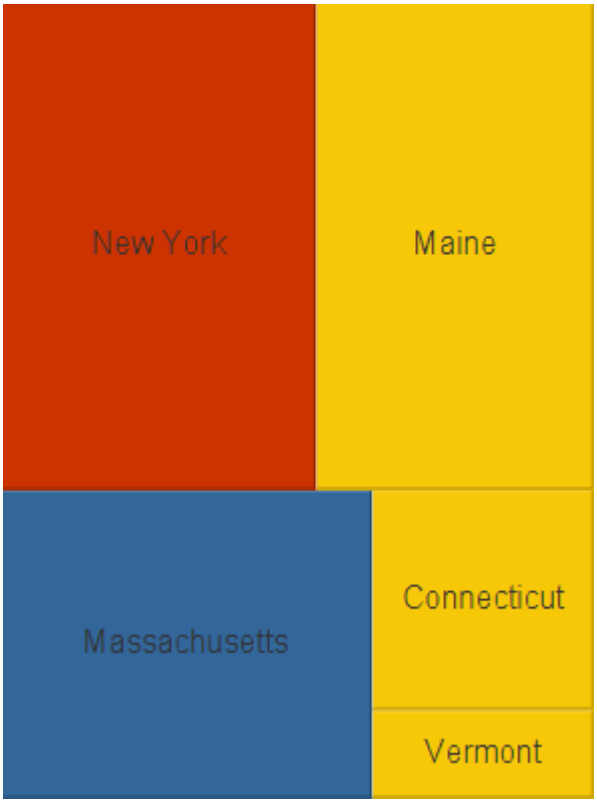


The following example shows the `treemap` element and its children defined in a MAF AMX file.

```
<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 14-97 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*.

14.7.27.1 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 How to Style Data Visualization Components.

The following 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;
   ...
}
```

The following 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;
}
```

The following 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 following 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"
      },
```

```
}
```

14.7.28 How to Create a Sunburst Component

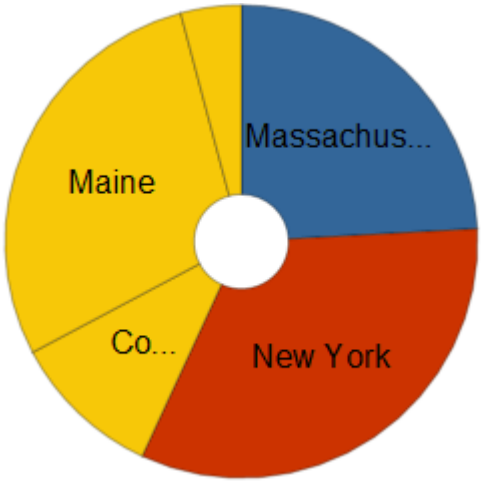
A Sunburst (`sunburst`) displays hierarchical data across two dimensions represented by the size and color of its nodes (`sunburstNode`).

In the Components window, the Sunburst is located under **MAF AMX Data Visualizations > Common > Miscellaneous**, and its child components are located under **MAF AMX Data Visualizations > Other Type-Specific Child Tags > Sunburst and Treemap** and **MAF AMX Data Visualizations > Shared Child Tags**.

The following 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 14-98 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*.

14.7.28.1 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 How to Style Data Visualization Components.

The following 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;
   ...
}
```

The following 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 following 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 following 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"
   }
}
```

14.7.29 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:

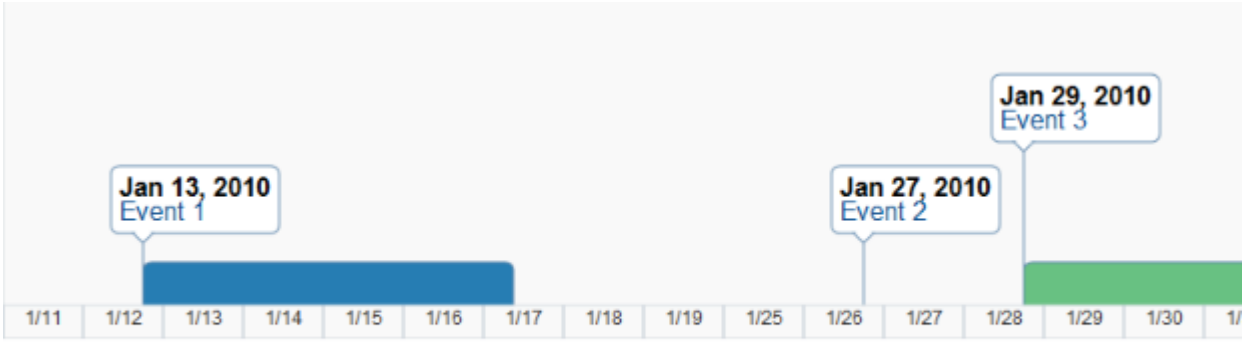
- Nested UI components
- Animation
- Attribute and time range change awareness
- Time fetching
- Custom time scales
- Time currency
- Partial triggers
- Data sorting
- Formatted time ranges
- Time zone
- Visibility

In the Components window, the Timeline is located under **MAF AMX Data Visualizations > Common > Miscellaneous**, and its child components are located under **MAF AMX Data Visualizations > Other Type-Specific Child Tags > Timeline** and **MAF AMX Data Visualizations > Shared Child Tags**.

The following 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 14-99 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*.

14.7.29.1 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 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`

`.timelineSeries-emptyTextStyle`

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`

The following 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]
   }
}
```

14.7.30 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 Components window, the NBox is located under **MAF AMX Data Visualizations > Common > Miscellaneous**, and its child components are located under **MAF AMX Data Visualizations > Other Type-Specific Child Tags > NBox** and **MAF AMX Data Visualizations > Shared Child Tags** (see How to Create a Treemap Component).

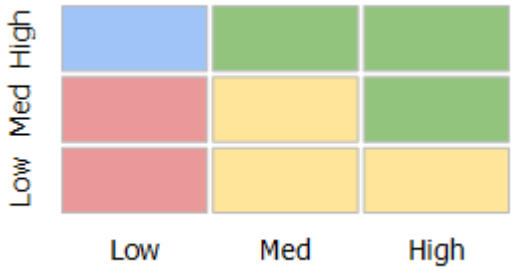
The following example shows the `nBox` element and its children defined in a MAF AMX file.

```
<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 14-100 NBox at Design Time



For information on attributes of the `nBox` element and its child elements, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.31 How to Create a Picto Chart

You use the Picto Chart (`pictoChart`) to define (in a data first scenario) or edit a data binding. A data binding is a way to connect data controls and methods to UI components.

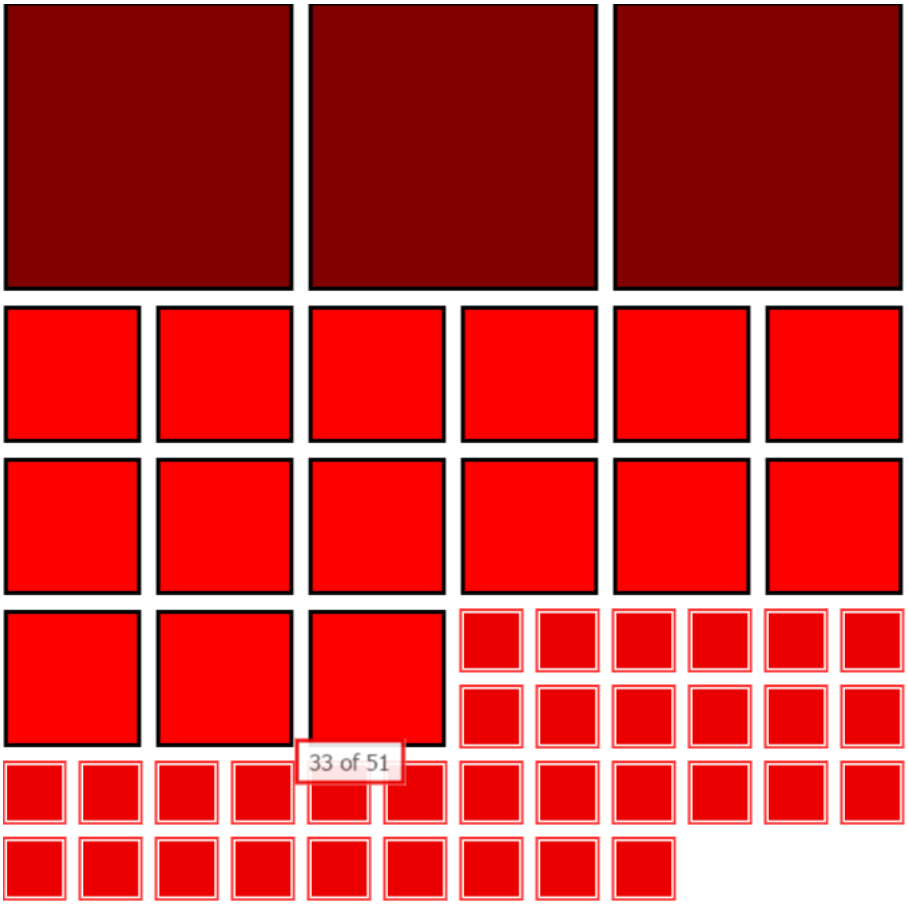
Picto charts use icons to visualize an absolute number, or the relative sizes of the different parts of a population. They are used in infographics as a more effective way to present numerical information than traditional tables and lists. Picto Charts are intended to be used in the default flowing layout. This makes the tool versatile since it can be easily used to visually enhance or support information in text or Data Visualization components.

In the **Components** window, the picto chart is located under **MAF AMX Data Visualization > Common > Miscellaneous**. The following example shows the picto element and its children defined in a MAF AMX file.

```
<dvtm:pictoChart id="pictoChart2"

animationDuration="#{pageFlowScope.animationDuration}"

  animationOnDataChange="#{pageFlowScope.animationOnDataChange}"
  animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
  columnCount="#{pageFlowScope.columnCount}"
  columnWidth="#{pageFlowScope.columnWidth}"
  emptyText="#{pageFlowScope.emptyText}"
  inlineStyle="#{pageFlowScope.inlineStyle}"
  layout="#{pageFlowScope.layout}"
  layoutOrigin="#{pageFlowScope.layoutOrigin}"
  rendered="#{pageFlowScope.rendered}"
  rolloverBehavior="#{pageFlowScope.rolloverBehavior}"
```

```
  rolloverBehaviorDelay="#{pageFlowScope.rolloverBehaviorDelay}"
  rowCount="#{pageFlowScope.rowCount}"
  rowHeight="#{pageFlowScope.rowHeight}"
  selectedRowKeys="#{pageFlowScope.selectedRowKeys}"
  shortDesc="#{pageFlowScope.shortDesc}"
  styleClass="#{pageFlowScope.styleClass}">

<dvtm:pictoChartItem id="pci21"
borderColor="#{pageFlowScope.borderColor}"
borderWidth="#{pageFlowScope.borderWidth}"
color="#{pageFlowScope.color1}"
columnSpan="#{pageFlowScope.span1}"
count="#{pageFlowScope.count1}"
rowSpan="#{pageFlowScope.span1}"="#{pageFlowScope.shape}"/>

<dvtm:pictoChartItem id="pci22"
borderColor="#{pageFlowScope.borderColor}"
borderWidth="#{pageFlowScope.borderWidth}"
color="#{pageFlowScope.color2}"
columnSpan="#{pageFlowScope.span2}"
count="#{pageFlowScope.count2}"
rowSpan="#{pageFlowScope.span2}"
shape="#{pageFlowScope.shape}"/>

<dvtm:pictoChartItem id="pci23"
borderColor="#{pageFlowScope.borderColor}"
borderWidth="#{pageFlowScope.borderWidth}"
color="#{pageFlowScope.color3}"
columnSpan="#{pageFlowScope.span3}"
count="#{pageFlowScope.count3}"
rowSpan="#{pageFlowScope.span3}"
shape="#{pageFlowScope.shape}"/>

</dvtm:pictoChart>
```

Figure 14-101 Picto Chart at Design Time

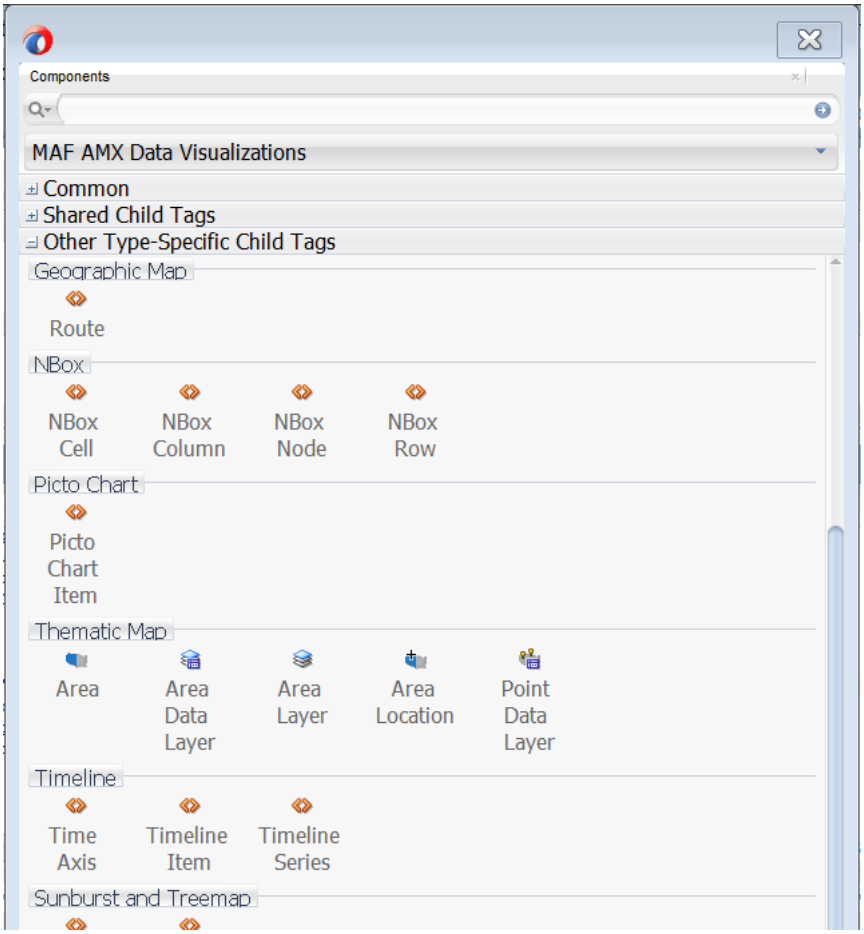


14.7.32 How to Define Child Elements for Map Components, Sunburst, Treemap, Timeline, Picto Chart, and NBox

You can define a variety of child elements for map components, Sunburst, Treemap, Timeline, Picto Chart, and NBox. For information on available child elements and their attributes, see *Tag Reference for Oracle Mobile Application Framework*.

In JDeveloper, the Map, Sunburst, Treemap, Timeline, Picto Chart, and NBox child components are located under **MAF AMX Data Visualizations > Other Type-Specific Child Tags** and **MAF AMX Data Visualizations > Shared Child Tags** in the **Components** window (see Figure 14-102).

Figure 14-102 Creating Map, Sunburst, Treemap, Timeline, Picto Chart, and NBox Child Components



14.7.33 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 How to Add Data Controls to a MAF AMX Page). The **Component Gallery** dialog that Figure 14-116 allows you to choose from a number of data visualization component categories, types, and layout options.

Figure 14-103 Component Gallery to Create Chart Components

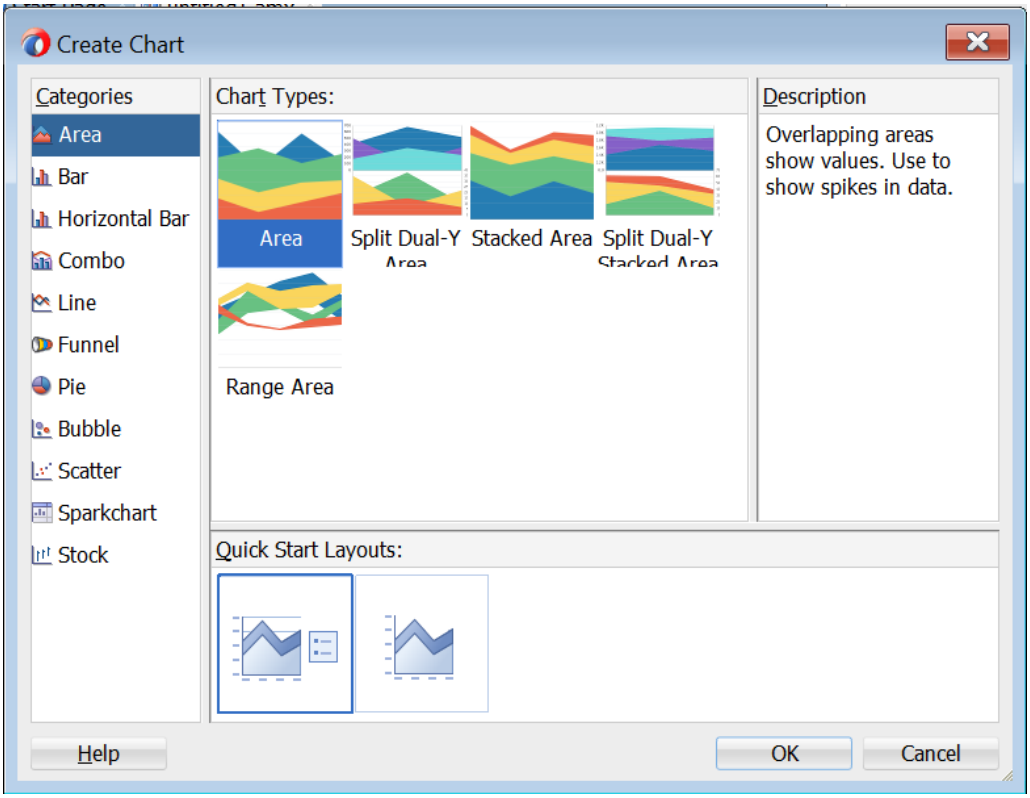


Figure 14-104 Component Gallery to Create Gauge Components

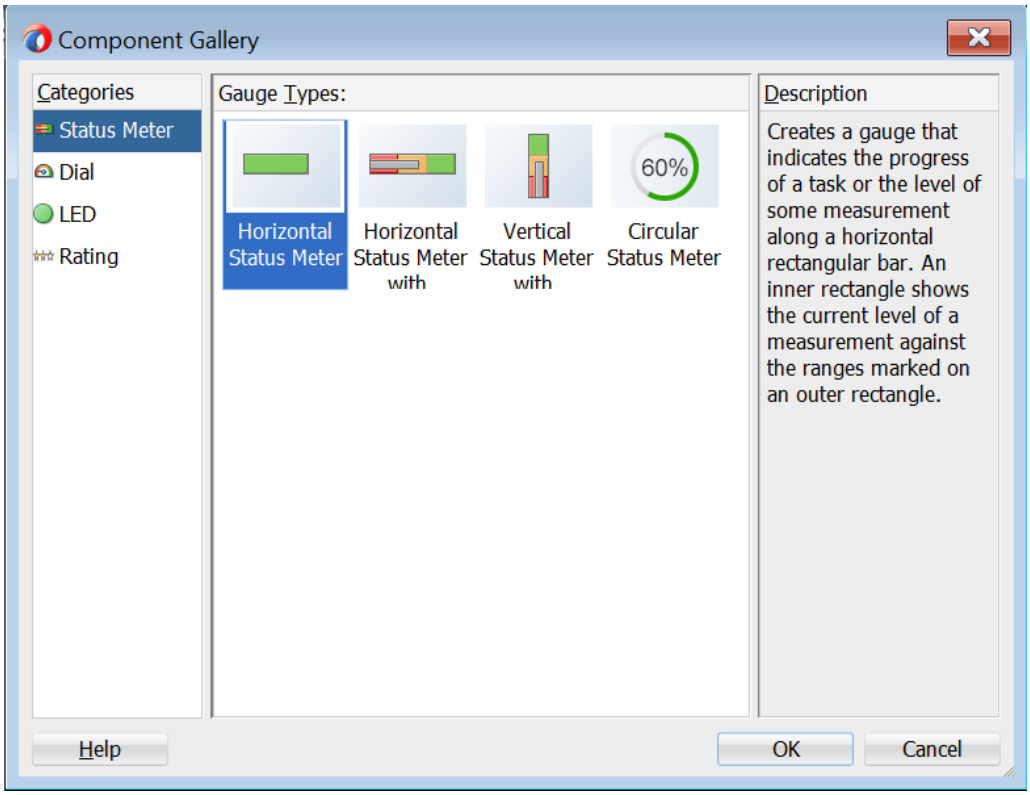
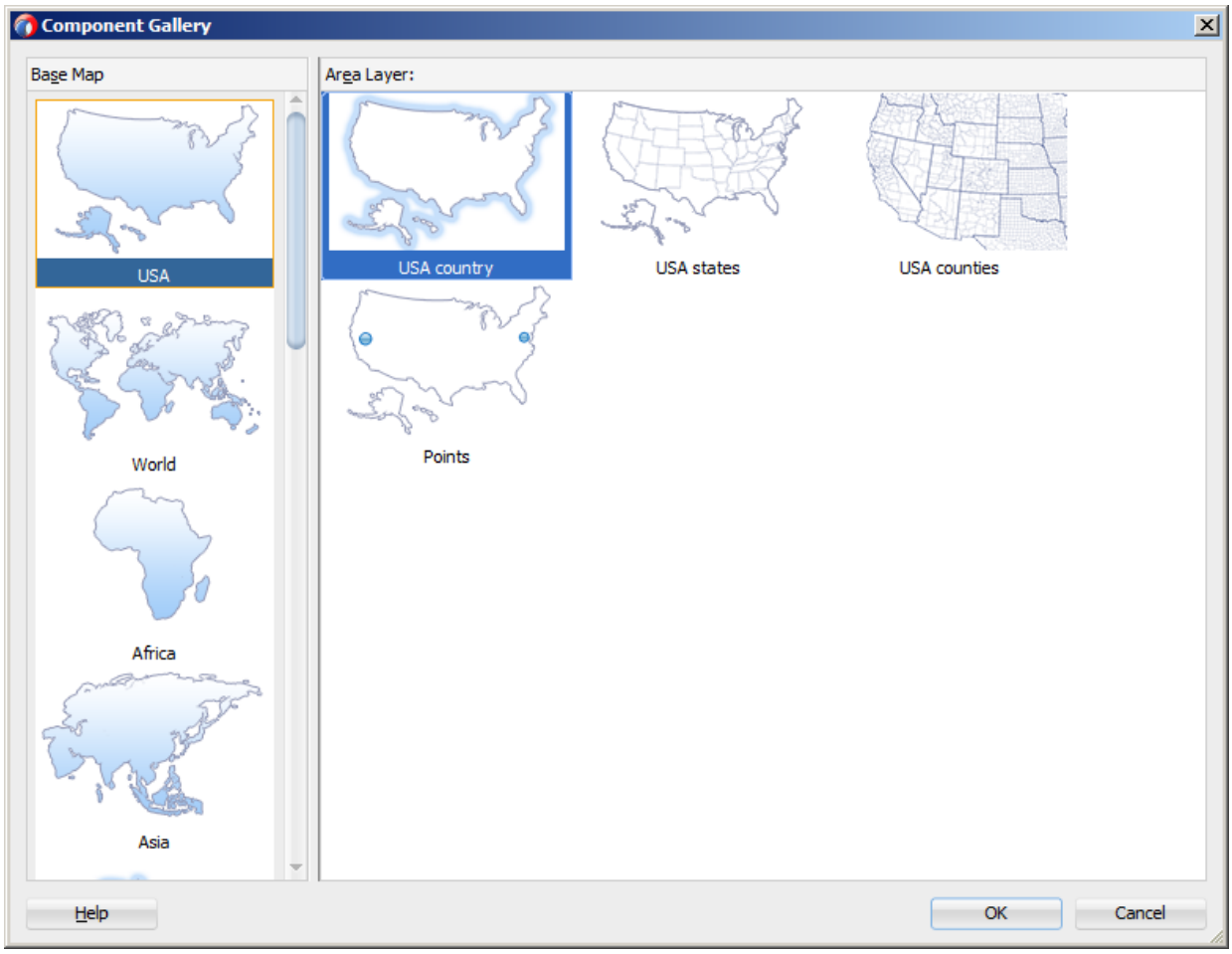


Figure 14-105 Component Gallery to Create Thematic Map Component

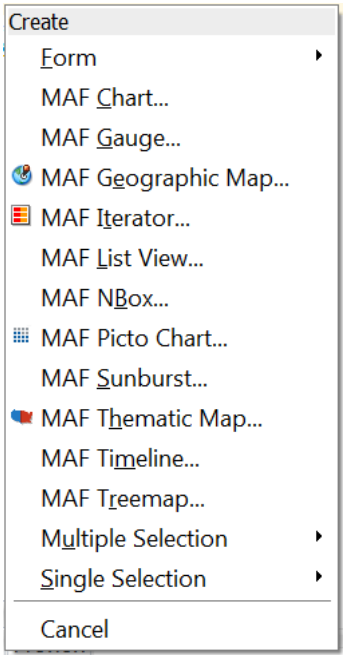


Note:

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.

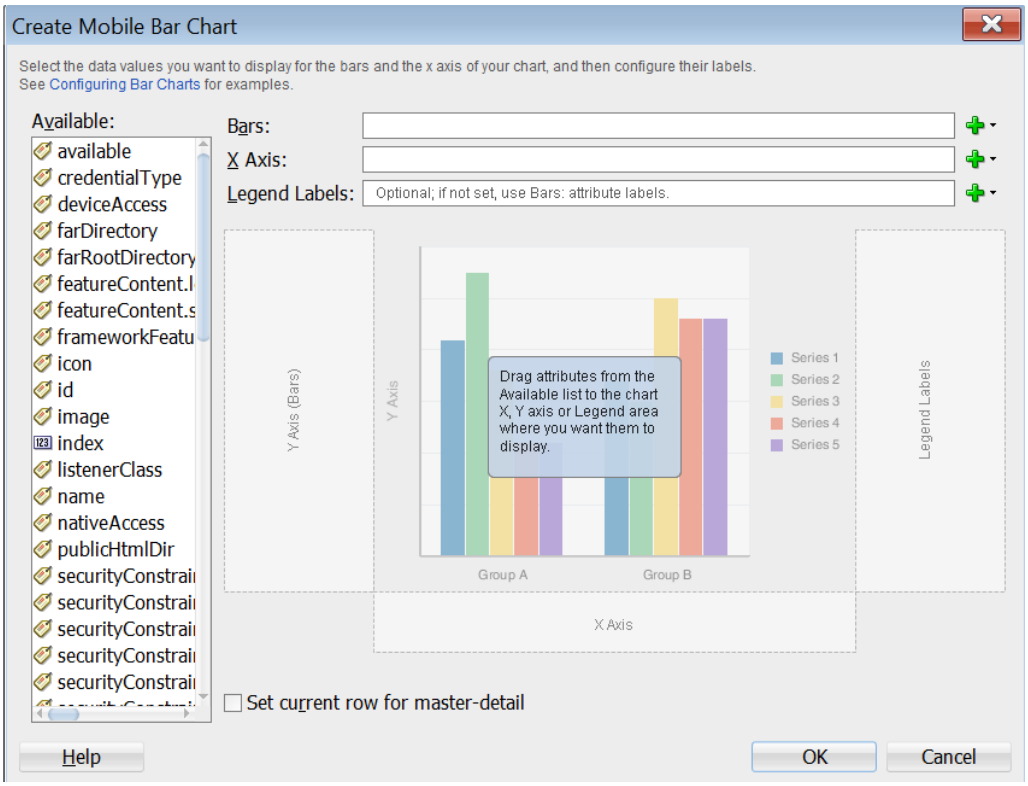
To trigger the display of the **Component Gallery**, you drag and drop a collection from the **Data Controls** window onto a MAF AMX page, and then select either **MAF Chart**, **MAF Gauge**, or **MAF Thematic Map** from the context menu that appears (see Figure 14-106).

Figure 14-106 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 binding the data collection attributes in the data visualization component using data binding dialogs. The name of 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 select Bar as the category and Bar as the type, then the name of the dialog that appears is **Create Mobile Bar Chart**, and the input field is labeled **Bars**, **X Axis**, and **Legend Labels** as Figure 14-107 shows. A dynamic sample of the chart's look and style based on the current mapping of attributes to the different fields appears in the center of the dialog. The areas around the sample are interactive and linked to the **Areas**, **Bars** or **Lines**, **X Axis**, and **Legend Labels** fields, and can be used as another target for drag and drop of attributes from the **Available** list.

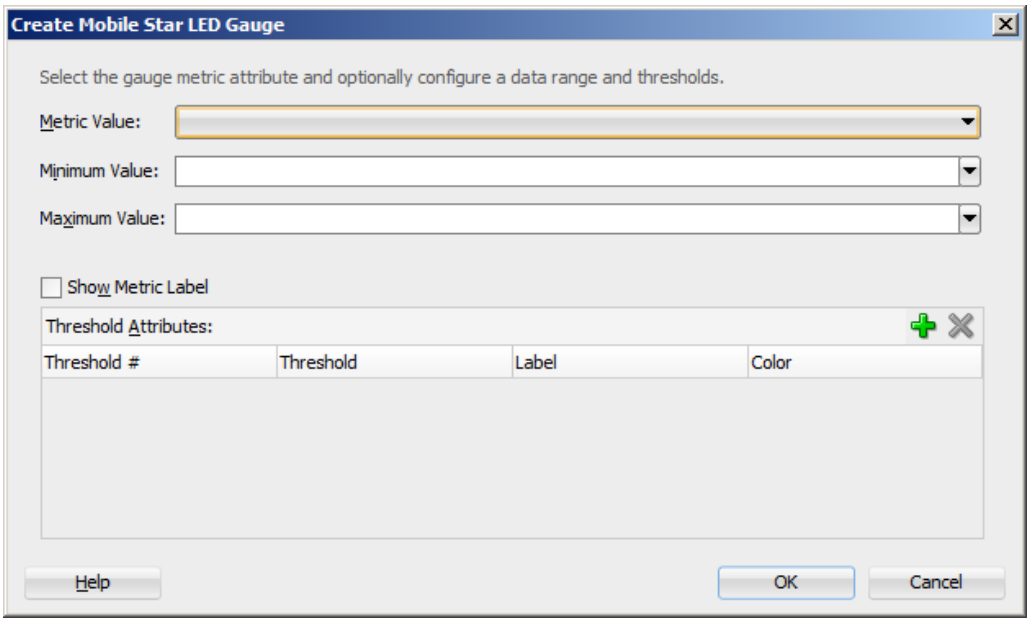
Figure 14-107 Specifying Data Values for Databound Chart



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).

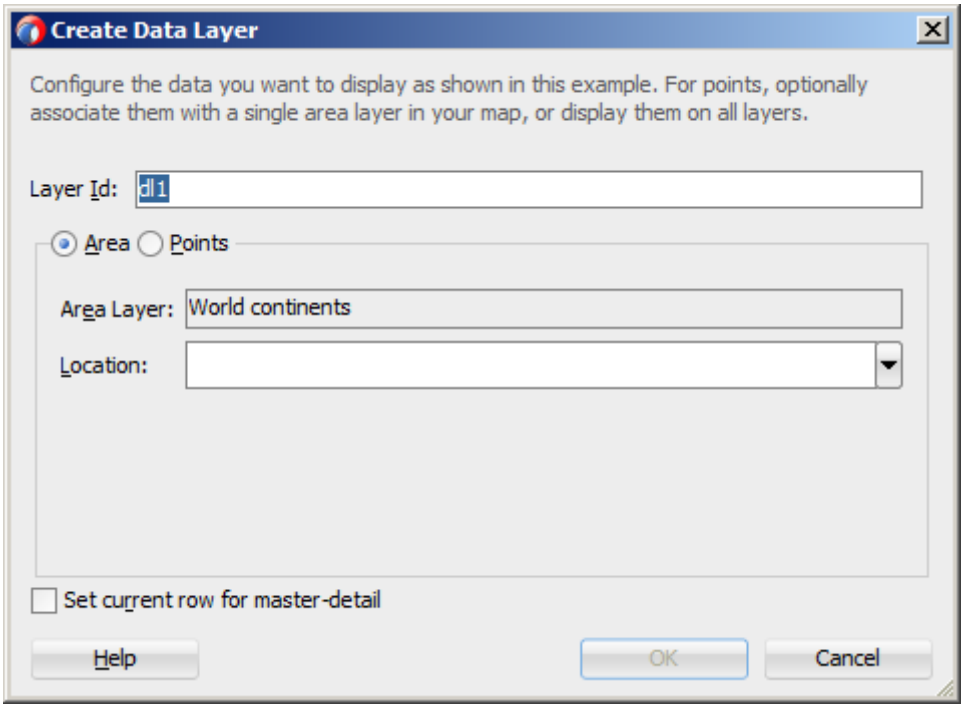
If you use the **Component Gallery** to create a databound gauge component, and then you select LED as the category and Star LED as the type, then the name of the dialog that appears is **Create Mobile Star LED Gauge**, as Figure 14-108 shows.

Figure 14-108 Specifying Data Values for Databound Gauge



If you use the Component Gallery to create a databound thematic map component, then regardless of your selection, the name of the dialog that appears is Create Data Layer, but the fields that are displayed depend on the selection you made in the Component Gallery. For example, if you select World as the base map and World continents as the area layer, the dialog show in Figure 14-108 opens.

Figure 14-109 Create Data Layer 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 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 14-110 Creating Databound Geographic Map

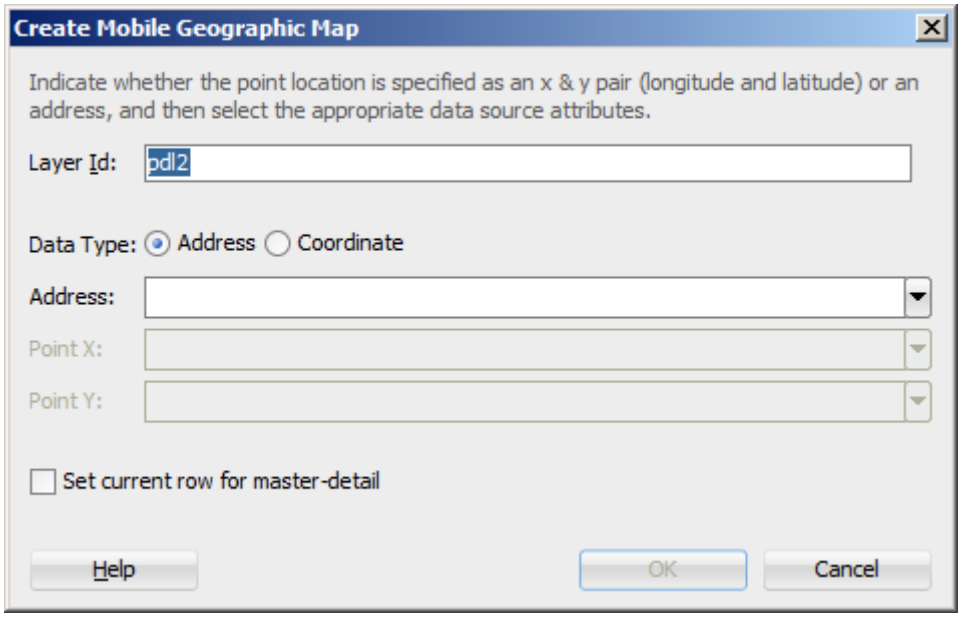


Figure 14-111 Creating Databound Sunburst

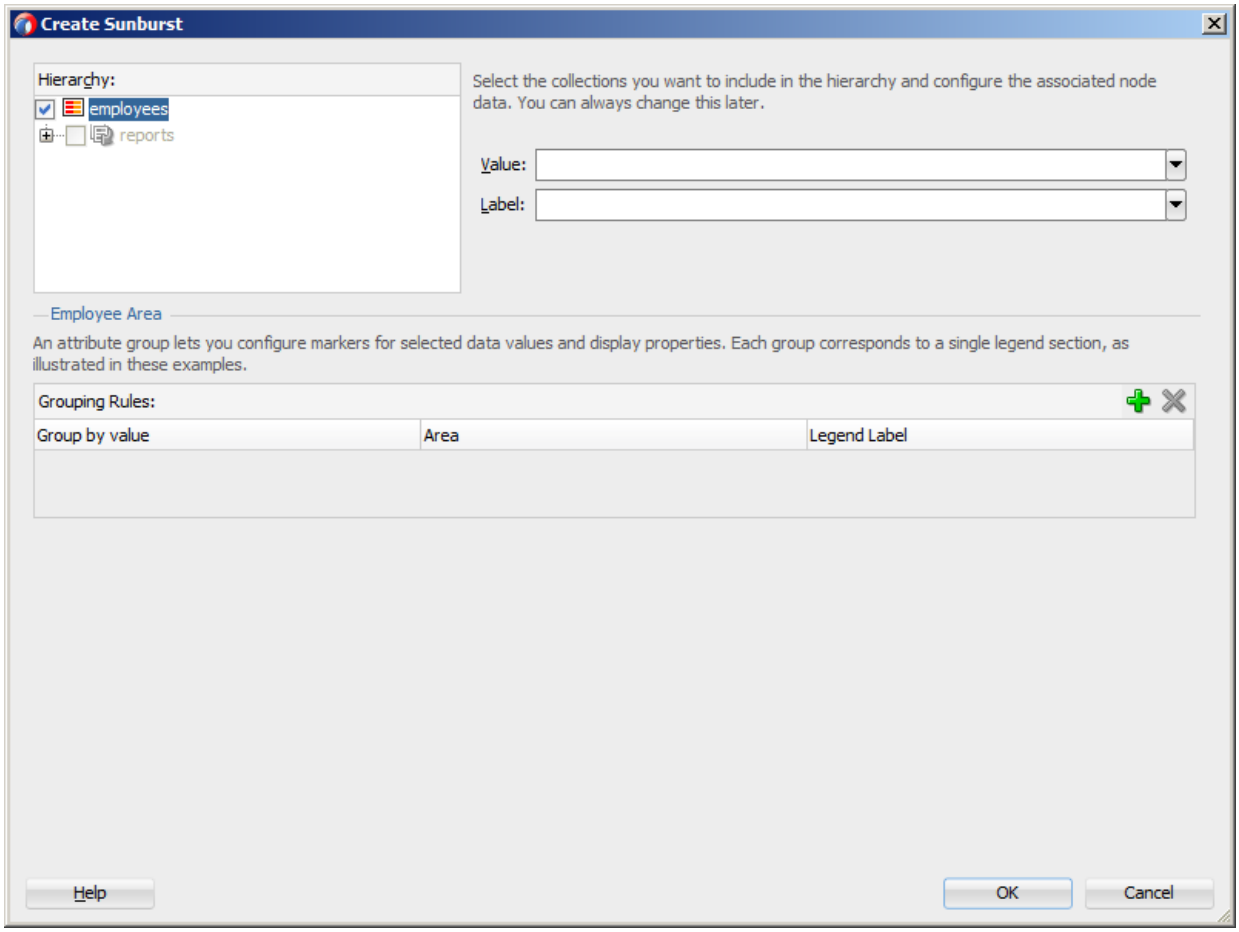


Figure 14-112 Creating Databound Timeline

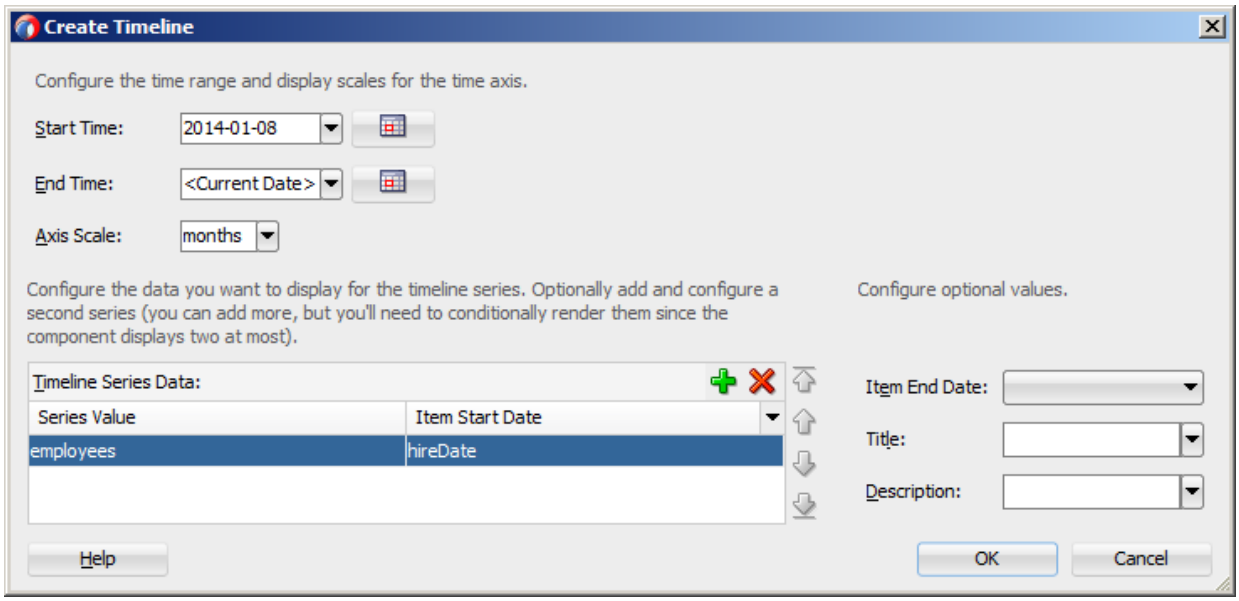


Figure 14-113 Creating Databound Treemap

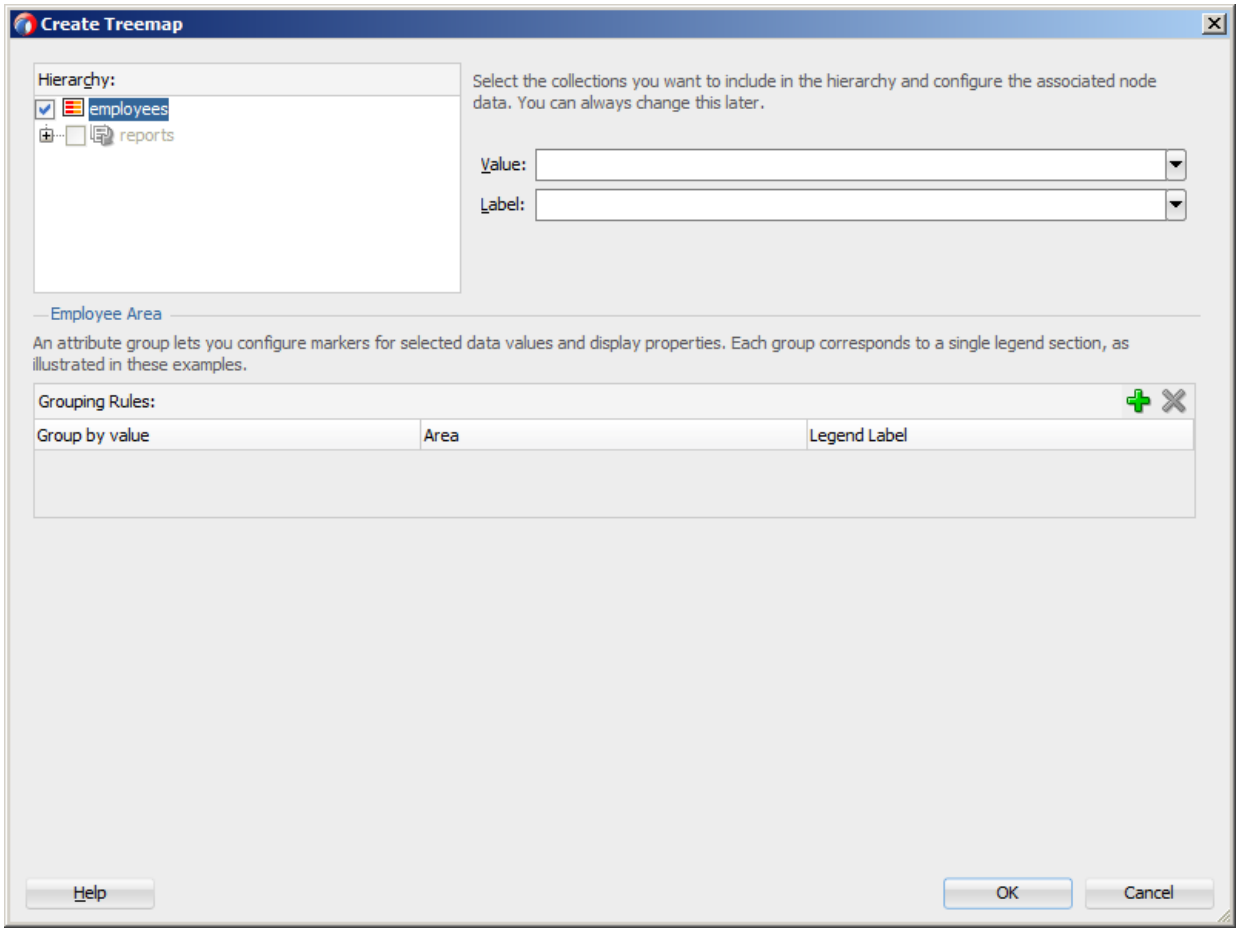
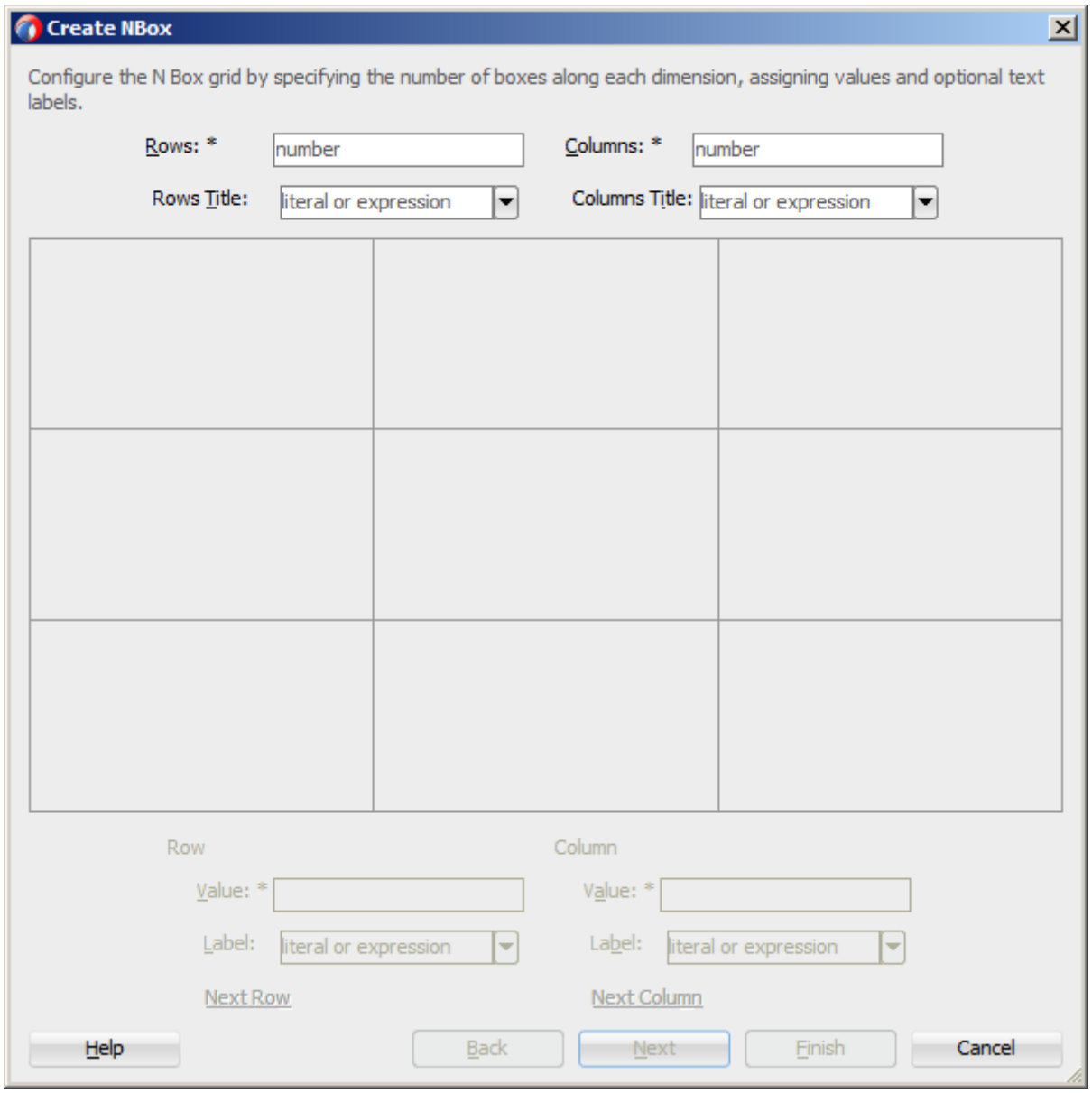


Figure 14-114 Creating Databound NBox



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 14-115 shows.

Figure 14-115 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**.

For details on values for each field of each dialog, consult the online help by clicking **Help** or pressing F1.

14.7.33.1 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 a list for the `series` attribute of the `seriesStyle` element. This list is already populated with the values of the `series` attribute based on the values of the `chartDataItem` elements within the `dataStamp` facet.

14.7.34 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"/>
```

14.7.35 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, 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 information, see *Tag Reference for Oracle Mobile Application Framework*.

14.7.36 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
- Combo Chart
- Bubble Chart
- Line Chart
- Scatter Chart

When the polar setting is applied to any of the preceding charts except the Bar Chart, 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, and the tangential axis are customized through the X Axis child component.

For information, see *Tag Reference for Oracle Mobile Application Framework*.

14.8 Styling UI Components

MAF enables you to employ CSS to apply style to UI components.

14.8.1 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 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 JDeveloper. 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 `Web Content` 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.

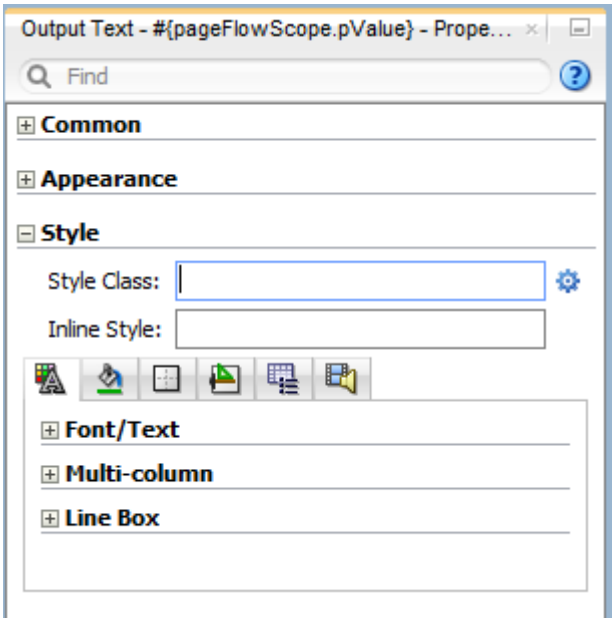
  Note:

  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 How to Enable Debugging of Java Code and JavaScript.

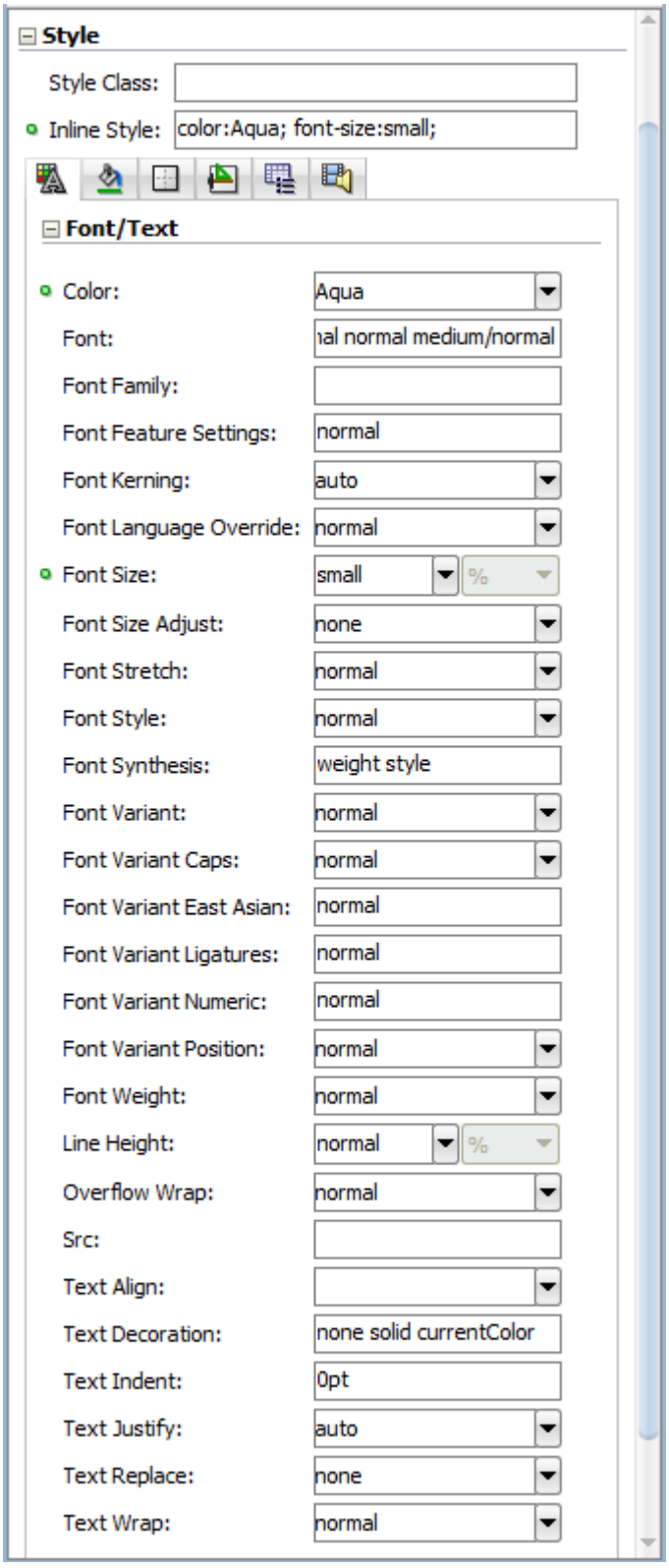
These attributes are displayed in the **Style** section in the **Properties** window, as Figure 14-116 shows.

Figure 14-116 Style Section of the Properties Window



Within the **Properties** window MAF AMX provides a drop-down editor that you can use to set various properties of the `inlineStyle` attribute (see Figure 14-117).

Figure 14-117 Inline Style Editor



For more information, see *Tag Reference for Oracle Mobile Application Framework*.

14.8.2 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 Skinning MAF Applications.

14.8.3 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:

```
#tb1 {
   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;
}
```

14.8.4 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 (see Overriding the Default Skin Styles).

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 How to Create a Dial Gauge).
- Duration bars color palette for the Timeline component (see How to Create a Timeline Component).
- Base maps for the Thematic Map component (see Defining a Custom Base Map).
- Style properties of the Geographic Map component (see How to Create a Geographic Map Component).

- Style properties of the Thematic Map component (see Applying Custom Styling to the Thematic Map Component).
- Selected and unselected states of the Rating Gauge component (see Applying Custom Styling to the Rating Gauge Component).

You should specify these custom JavaScript files in the Includes section at the application feature level (see Defining the Application Feature Content as a MAF AMX Page or Task Flow). 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 following example 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>
```

14.9 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`
  - What You May Need to Know About the Basic WAI-ARIA Terms

- The Oracle Global HTML Accessibility Guidelines (OGHAG).

  For more information, see 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, use the WAI-ARIA live region (see 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

14.9.1 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 14-120).

AMX UI components, such as `commandButton`, `tableLayout` expose one or more of the following attributes to support accessibility:

- Shortdesc.
- Summary
- HintText
- Landmark

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 appears as a value of the `alt` attribute of the image element. The value of the `shortDesc` attribute can be localized.

For the `summary` and `shortDesc` attributes that the `tableLayout` component exposes, you do not need to specify values for either attribute if you use the `tableLayout` component to lay out other components. In this scenario, MAF adds a presentation role to the `tableLayout` component in the HTML that renders at runtime. However, if you use the `tableLayout` component to present data, configure each attribute with different values. At runtime, MAF uses the value you specify for the `shortDesc` attribute as the `title` attribute in the HTML table. The value you specify for the `summary` attribute renders as the summary in the HTML table.

You can set the attributes of the component through the Accessibility section of the **Properties** window.

For more information about these attributes and the components, see *Tag Reference for Oracle Mobile Application Framework*

For the Panel Group Layout and Deck components, you define the landmark role type (see Table 14-14) 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

AMX UI components, such as `inputDate` and `inputNumberSlider`, have `Label` and `Value` accessibility attributes defined by the WAI-ARIA specification. These attribute values are automatically applied at run time and cannot be modified. For more information, see *Tag Reference for Oracle Mobile Application Framework*.

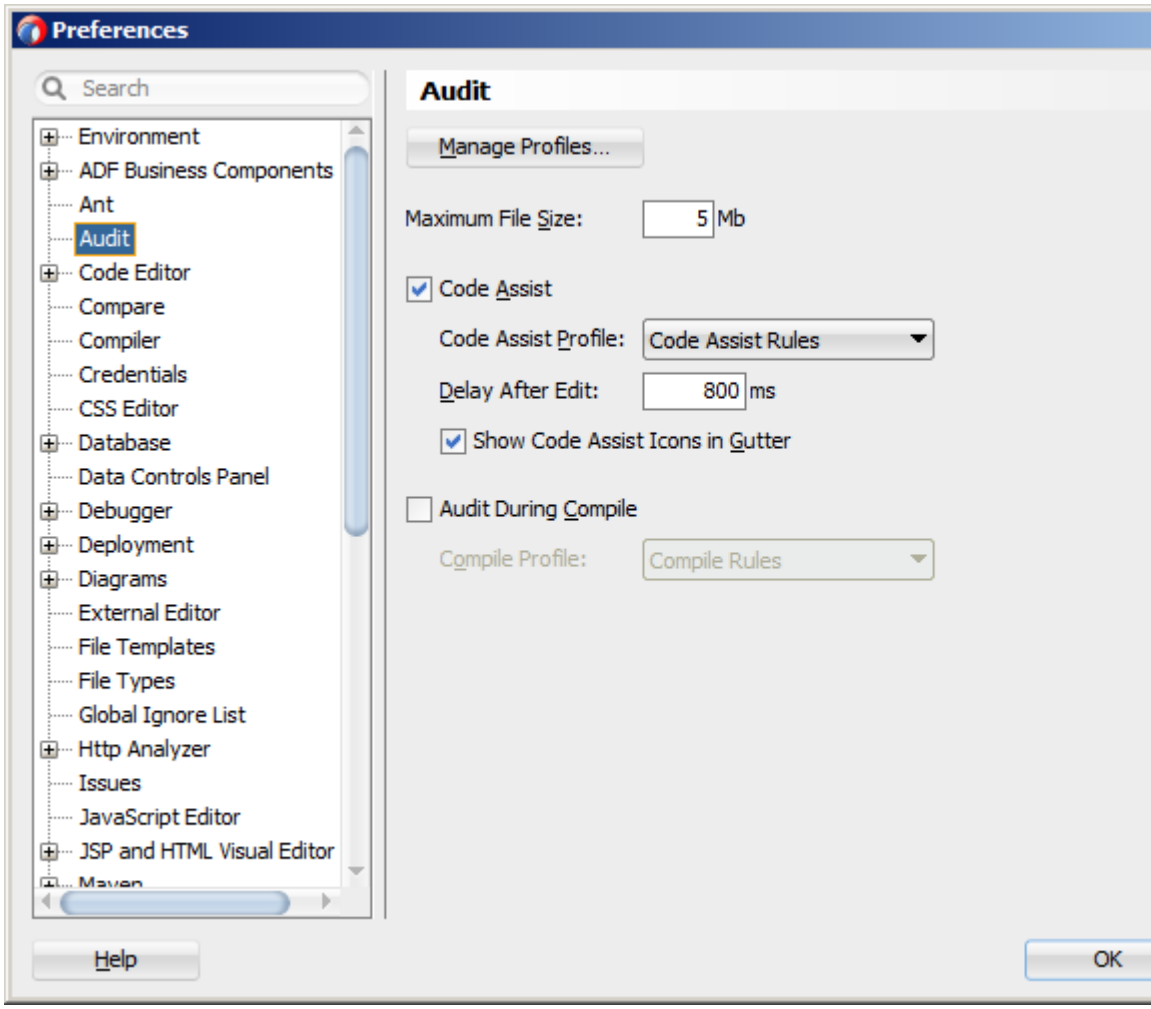
For information on how to configure the accessibility audit rules, see Configuring the Accessibility Audit Rules.

14.9.1.1 Configuring the Accessibility Audit Rules

You can configure the accessibility audit rules using JDeveloper's Preferences dialog as follows:

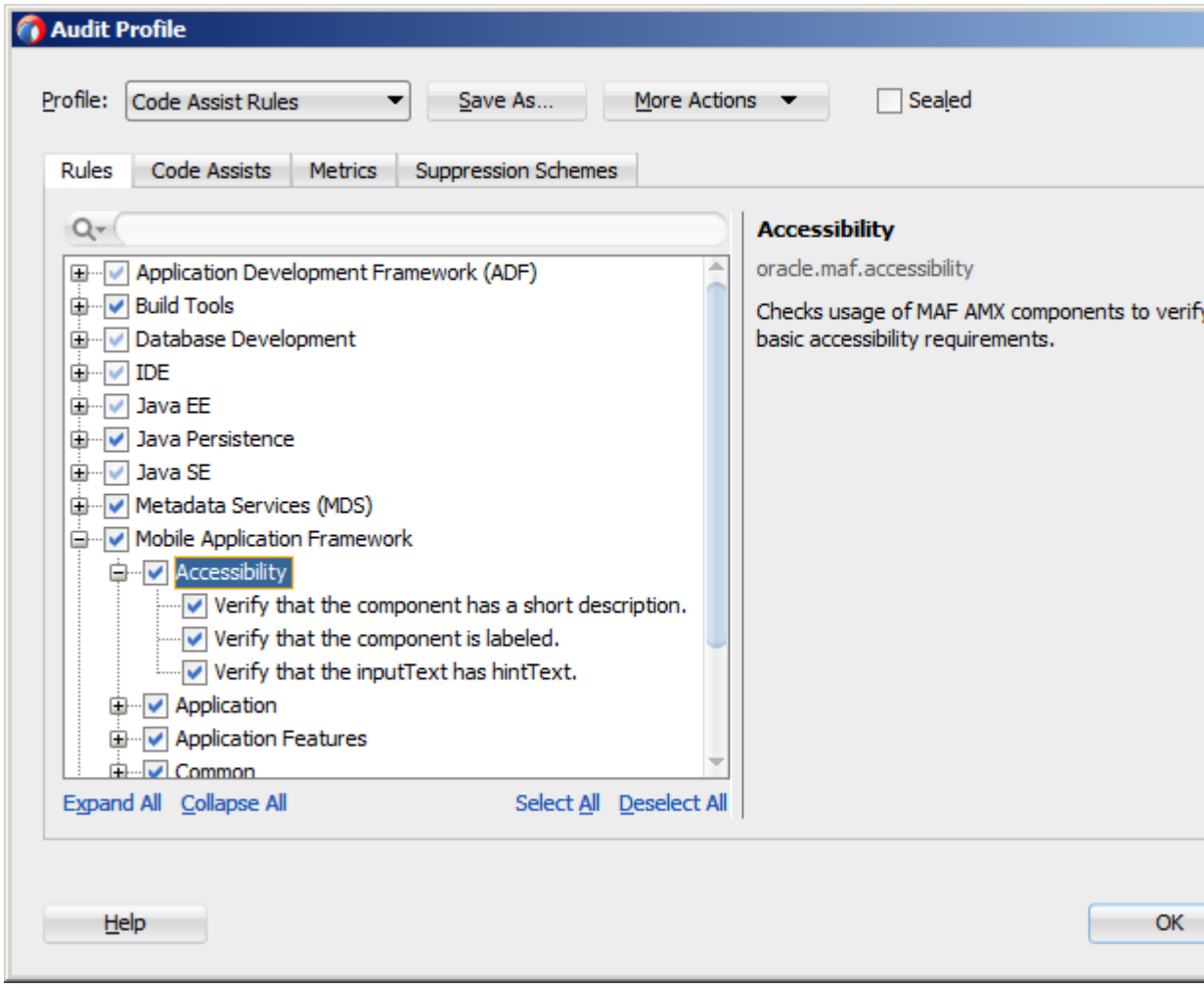
1. In JDeveloper, select **Tools > Preferences** from the main menu (on a Windows computer).
2. From the list of preferences, select **Audit** (see Figure 14-118).
3. On the **Audit** pane that Figure 14-118 shows, click **Manage Profiles** to open the **Audit Profile** dialog.

Figure 14-118 Setting Accessibility Audit Rules



4. In the **Audit Profile** dialog that Figure 14-119 shows, expand the **Mobile Application Framework** node from the tree of rules, and then expand **Accessibility**.

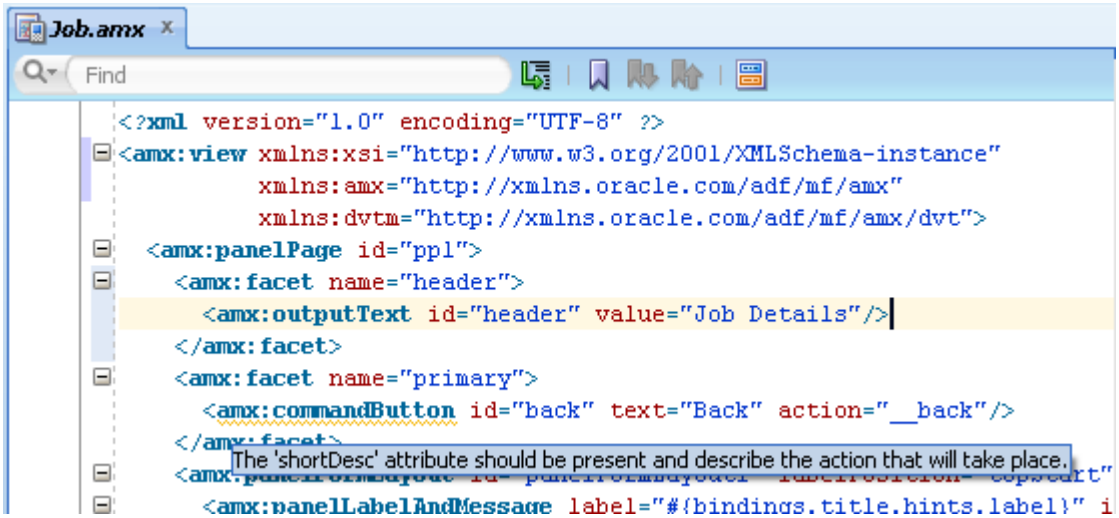
Figure 14-119 Audit Profile Dialog



5. Select the accessibility audit rules to apply to your application, as Figure 14-119 shows.

Figure 14-120 shows the accessibility audit warning displayed in JDeveloper.

Figure 14-120 Accessibility Audit Warning



For information on how to test your accessible MAF AMX application feature, see 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.

14.9.2 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 cannot 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 14-11 lists abstract roles that are used to support the WAI-ARIA role taxonomy for the purpose of defining general role concepts.

Table 14-11 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 14-12 lists widget roles that act as standalone user interface widgets or as part of larger, composite widgets.

Table 14-12 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

Table 14-12 (Cont.) Widget Roles

Widget Role	Description	Widget Required States
textbox	Input that allows free-form text as its value.	aria-labelledby, aria-readonly, aria-required, aria-multiline, aria-disabled (state)

Table 14-13 lists document structure roles that describe structures that organize content in a page. Typically, document structures are not interactive.

Table 14-13 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 14-14 lists landmark roles that represent regions of the page intended as navigational landmarks.

Table 14-14 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 How to Configure UI and Data Visualization Components for Accessibility.

14.9.3 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`.

14.10 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 following example 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>
```

The following example shows how to define a validation message displayed in a popup in a MAF AMX file.

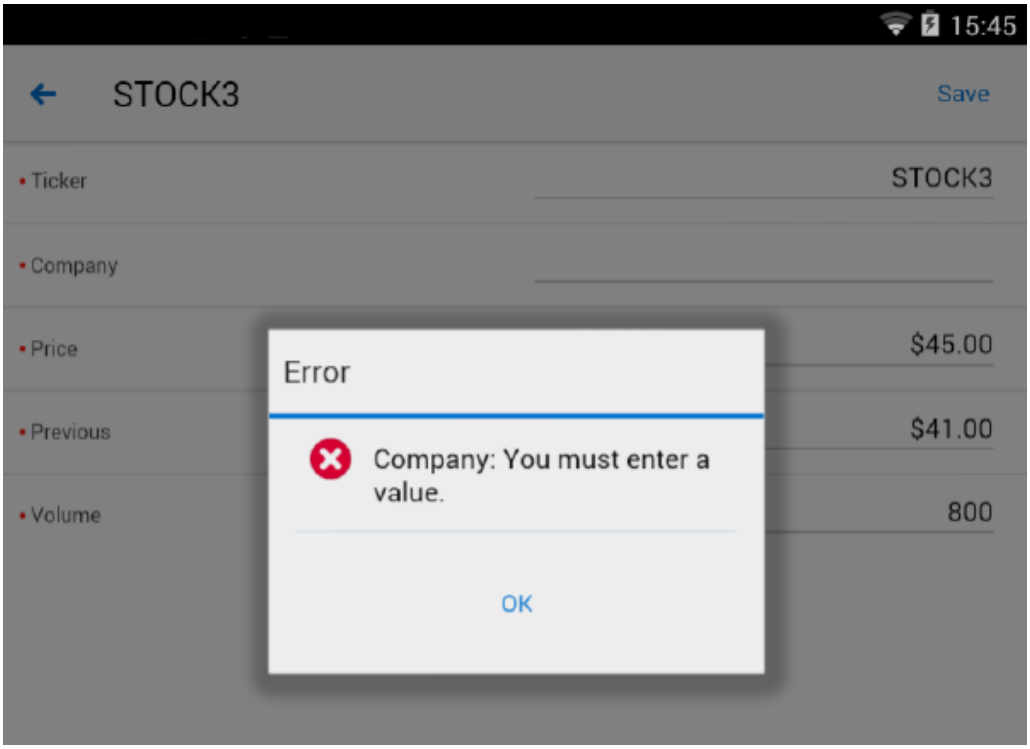
```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="outputText1" value="Login Demo"/>
   </amx:facet>
   <amx:facet name="secondary">
      <amx:commandButton id="btnBack" action="__back" text="Back"/>
   </amx:facet>
   <amx:panelGroupLayout id="panelGroupLayout1">
      <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 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 14-121 shows a popup validation message that appears in the `StockTracker` sample application when you fail to enter a company name. For more information about this and other sample applications, see MAF Sample Applications.

Figure 14-121 Validation Message in StockTracker Sample Application



14.11 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.

You can define listeners as attributes of UI components in one of the following ways:

- Manually in the source of your MAF AMX file.
- From the **Properties** window for 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.String` representing the ID of the drilled object
  - `java.lang.String` representing the rowkey of the drill data item
  - `java.lang.String` representing the group name of the drilled object
  - `java.lang.String` 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 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*

MAF allows you to create managed bean methods for listeners so that your managed bean methods use MAF AMX-specific event classes. The following three examples 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.

The following example shows how to call 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>
```

The following example shows how to use the `AMXEvent`.

```
private void actionListenerMethod(AMXEvent amxEvent) {
   // Some Java handling
}
```

The following example shows how to invoke 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 located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer. This sample demonstrates how to call listeners from Java beans.

14.11.1 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.

For information about the supported event listeners and event types for AMX UI components, see the *Tag Reference for Oracle Mobile Application Framework*.

The `type` attribute (see Figure 14-122) of each of the child event listeners has a base set of values that match the listener events. These values are filtered to show only the events that the parent component supports. 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 14-122 shows values available in the constrained Type list of the Set Property Listener for a parent List Item component.

Figure 14-122 Selecting Event Type

15

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) within a MAF AMX application feature. In addition, object scope lifecycles, managed beans, UI hints, validation, and data change events are also discussed.

This chapter includes the following sections:

- Introduction to Bindings and Data Controls
- About Object Scope Lifecycles
- Creating EL Expressions
- Creating and Using Managed Beans
- Exposing Business Services with Data Controls
- Creating Databound UI Components from the Data Controls Panel
- What Happens at Runtime: How the Binding Context Works
- Configuring Data Controls
- Working with Attributes
- Creating and Using Bean Data Controls
- Sharing Instances of Data Controls Across Application Features
- Using the DeviceFeatures Data Control
- Validating Attributes
- Working with Data Change Events

15.1 Introduction to Bindings and Data Controls

Data controls use standard metadata interfaces to describe the operations and data collections of a business service while declarative bindings contain information about accessing data from data controls. The model layer reads relevant XML files for information about data controls and bindings, and connects the user interface with the business service.

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 operations

and data collections of the service, including information about the properties, methods, and types involved. Using JDeveloper, 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 bindings of the page. 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 Data Controls panel and dropping it on a page as a specific UI component. When you use data controls to create a UI component, JDeveloper 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 Data Controls panel in JDeveloper, 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 a `DataControls.dcx` file, which points at the data control source, and the page bindings link the reference of the specific page to the data control. For information about the ApplicationFeatures data control, see What You May Need to Know About Custom Springboard Application Features with MAF AMX Content.

For more information about data controls and bindings, see the following:

- Exposing Business Services with Data Controls
- Creating Databound UI Components from the Data Controls Panel
- What Happens at Runtime: How the Binding Context Works
- Configuring Data Controls
- Working with Attributes
- Creating and Using Bean Data Controls
- Sharing Instances of Data Controls Across Application Features
- Using the DeviceFeatures Data Control

15.2 About Object Scope Lifecycles

A scope persists those objects in memory that an application may access at runtime using EL expressions. Scopes determine the lifespan and availability of an object, such as an application or a task flow.

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 lifetime and visibility of the variable. 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) or task flow, depending on where the page flow-scoped managed bean is defined. If the bean is defined in an unbounded task flow, its scope is the feature. If the bean is defined in a bounded task flow, its scope is limited to the task flow.
- 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. See About the Managed Beans Category and 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 What You May Need to Know About Serialization of Bean Class Variables.

15.2.1 What You May Need to Know About Object Scopes and Task Flows

Use page flow scope to pass data values between activities within a task flow, and use view scope for variables that are only needed within the current view activity.

When determining what scope to use for variables within a task flow, you must only use view or page flow scopes. The application scope persists objects in memory beyond the life of the task flow and therefore compromises 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 must use page flow scope. View scope must be used for variables that are needed only within the current view activity, not across view activities.

15.3 Creating EL Expressions

EL expressions are used in MAF applications to bind attributes to object values determined at runtime. Build EL expressions using the page definition files to automate access to individual objects and their properties without employing code.

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 is 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 `rendered` attribute of the UI component. 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. See 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;
}
```

15.3.1 About Data Binding EL Expressions

MAF builds data binding expressions when you use the Data Controls panel to create UI components. You can also manually add or edit MAF data binding expressions.

When you use the Data Controls panel 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. See 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 binding container of the current page. 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 Data Controls panel to create UI components, these expressions are built for you. However, you can also manually create them if you need to. The JDeveloper 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. See What You May Need to Know About MAF Binding Properties.

15.3.2 How to Create an EL Expression

Use the procedure to build expressions, and add logical or mathematical operators to them using the Expression Builder.

You can create EL expressions declaratively using the JDeveloper Expression Builder. You can access the Expression Builder from the **Properties** window.

Before you begin:

It may be helpful to have an understanding of EL expressions. See Creating EL Expressions.

To use the Expression Builder:

1. In the **Properties** window, locate the attribute you wish to modify and use the right-most drop-down menu to select **Expression Builder**.
2. Create expressions using the following features:

   - Use the **Variables** drop-down 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 either use the drop-down filter or enter search criteria in the search field. The EL accessible objects exposed by MAF are located under the **Mobile Application Framework Objects** node, which is under the **ADF Managed Beans** node.

     Tip:

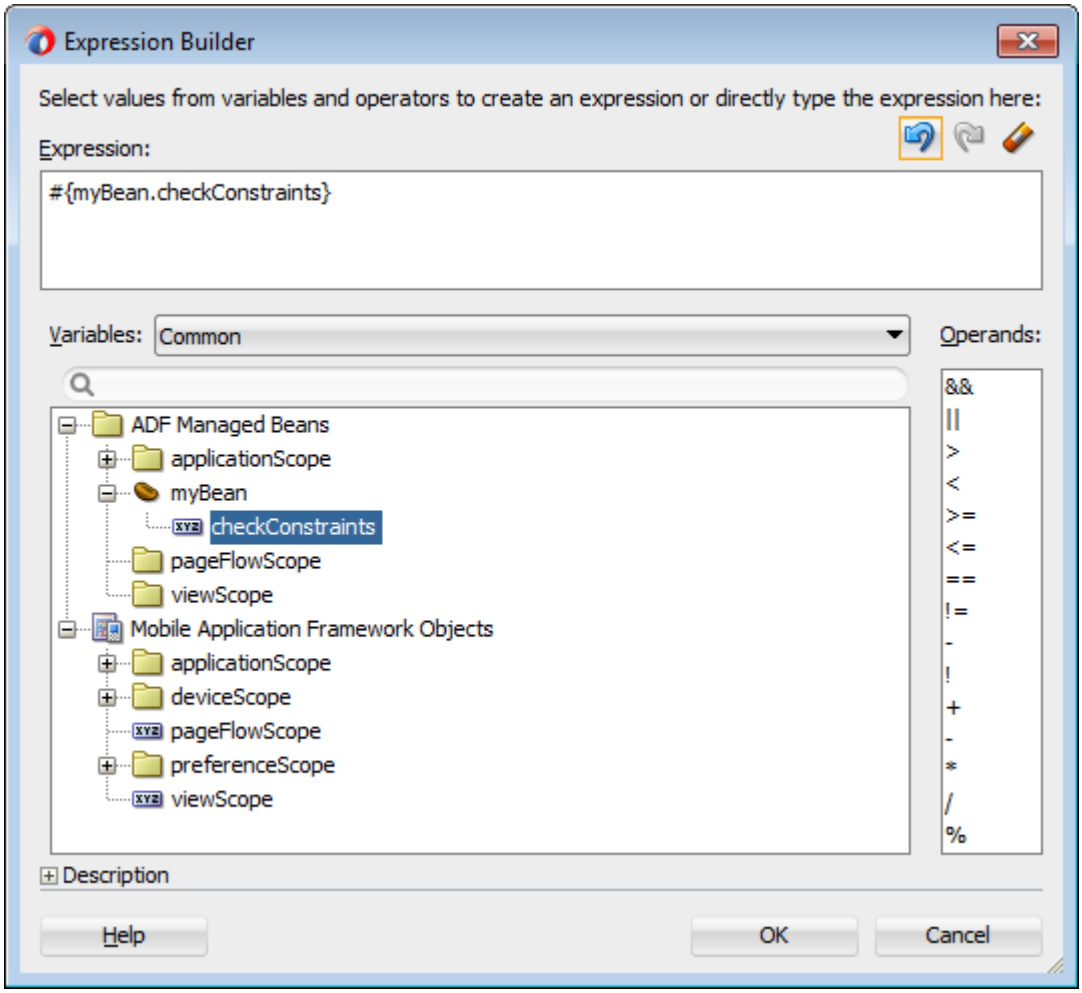
     For more information about these objects, see the MAF Javadoc. See also About the Categories in the Expression Builder.

     Selecting an item in the tree causes it to be moved 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.

Figure 15-1 shows an example of how to create an EL expression from the ADF Managed Beans category. However, you can create EL expressions from any of the categories described in About the Categories in the Expression Builder.

Figure 15-1 The Expression Builder Dialog



Tip:

For information about using proper syntax to create EL expressions, see the Java EE tutorial at `http://www.oracle.com/technetwork/java/index.html`.

Table 15-1 Icons Under the Bindings Node of the Expression Builder

Icon	Description
bindings	Represents the `bindings` container variable, which references the binding container of the current page. Opening the `bindings` node exposes all the binding objects for the current page.

Table 15-1 (Cont.) Icons Under the Bindings Node of the Expression Builder

Icon	Description
	Represents the `data` binding variable, which references the entire binding context (created from all the `.cpx` files in the application). Opening the `data` node exposes all the page definition files in the application.
	Represents an action binding object. Opening a node that uses this icon exposes a list of valid action binding properties.
	Represents an iterator binding object. Opening a node that uses this icon exposes a list of valid iterator binding properties.
	Represents an attribute binding object. Opening a node that uses this icon exposes a list of valid attribute binding properties.
	Represents a list binding object. Opening a node that uses this icon exposes a list of valid list binding properties.
	Represents a table or tree binding object. Opening a node that uses this icon exposes a list of valid table and tree binding properties.
	Represents a MAF binding object property. See What You May Need to Know About MAF Binding Properties.
	Represents a parameter binding object.
	Represents a bean class.
	Represents a method.

15.3.2.1 About the Method Expression Builder

The Method Expression Builder in the Properties window is akin to the Expression Builder except that the Method Expression Builder filters the managed beans depending on the selected property.

Table 15-2 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 15-2 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

Table 15-2 (Cont.) Properties for the Method Expression Builder

Property	Element
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
viewportChangeListener	dvtm:areaChart
viewportChangeListener	dvtm:barChart
viewportChangeListener	dvtm:comboChart
viewportChangeListener	dvtm:horizontalBarChart
viewportChangeListener	dvtm:lineChart

15.3.2.2 About Non EL-Properties

Properties that are not EL-enabled do not have the **EL Expression Builder** option available in the Properties window.

Table 15-3 shows the properties that do not have the **EL Expression Builder** option available in the **Properties** window, because they are not EL-enabled.

Table 15-3 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
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

15.3.3 What You May Need to Know About MAF Binding Properties

When a databound component is created using the Expression Builder, the EL expression might reference specific MAF binding properties, which are defined by Oracle APIs.

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 binding properties are defined by Oracle APIs. For a full list of the available properties for each binding type, see Table 15-4

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 in the following example.

```
<iterator Binds="ItemsForOrder" RangeSize="25"
              DataControl="BackOfficeAppModuleDataControl"
              id="ItemsForOrderIterator" ChangeEventPolicy="ppr"/>
```

15.3.4 How to Enable Retention of Data Provider State Across Iterators

By specifying the same `RSIName` attribute on the top-level iterator of each page that is to access a data collection instance, a data control can have multiple instances of a data provider created for it.

To use this functionality, you specify the same `RSIName` attribute on the top-level iterator (`iterator`) of each page that is to access the same data collection instance.

The following example shows a hierarchy of iterators in a `pageDef` file. The last two accessor iterators enable iteration over a second collection of Employee objects as well as the phone numbers of those employees.

```
<iterator Binds="root"
          RangeSize="25"
          DataControl="BusinessManager"
          id="BusinessManagerIterator"
          RSIName="secondCollection" />
<accessorIterator id="companyIterator"/
                  MasterBinding="BusinessManagerIterator"
                  Binds="company"
                  RangeSize="25"
                  DataControl="BusinessManager"
                  BeanClass="mobile.Company"/>
<accessorIterator id="employeesIterator"
                  MasterBinding="companyIterator"
                  Binds="employees"
                  RangeSize="25"
                  DataControl="BusinessManager"
                  BeanClass="mobile.Employee"/>
<accessorIterator id="phoneNumbersIterator"
                  MasterBinding="employeesIterator"
                  Binds="phoneNumbers"
                  RangeSize="25"
                  DataControl="BusinessManager"
                  BeanClass="mobile.PhoneNumber"/>
<accessorIterator id="employeesIterator2"
                  MasterBinding="companyIterator"
```

```
                    Binds="employees"
                    RangeSize="25"
                    DataControl="BusinessManager"
                    BeanClass="mobile.Employee"
                    RSIName="secondCollection" />
<accessorIterator id="phoneNumbersIterator2"
                    MasterBinding="employeesIterator2"
                    Binds="phoneNumbers"
                    RangeSize="25"
                    DataControl="BusinessManager"
                    BeanClass="mobile.PhoneNumber" />
```

15.3.5 How to Reference Binding Containers

MAF uses different root EL expressions to reference the binding containers of an active screen and an inactive screen. Review the listed properties that can be used in EL expressions to access MAF AMX binding objects.

You can reference the binding container of the active screen by the root EL expression `"#{bindings}"` and you can reference binding container of another screen through the expression `"#{data.PageDefName}"`. The MAF AMX binding objects are referenced by name from the binding container `"#{bindings.Name}"`.

Table 15-4 shows a partial list of the properties that you can use in EL expressions to access values of the MAF AMX binding objects at runtime. The properties appear in alphabetical order.

Table 15-4 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

Table 15-4 (Cont.) Runtime EL Properties of MAF Bindings

Runtime Property	Description	Iterator	Action	attributeValues	Tree
`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
`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

Table 15-4 (Cont.) Runtime EL Properties of MAF Bindings

Runtime Property	Description	Iterator	Action	attributeValues	Tree
`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}`.

15.3.6 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

15.3.6.1 About the Bindings Category

You can configure the properties that are supported for different binding types. Review the supported bindings and properties in 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 15-5 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 15-5 Supported Binding Types

Binding Type	Properties

`accessorIterator`	`class`
	`currentRow: dataProvider`
	`name`
	`rangeSize`

Table 15-5 (Cont.) Supported Binding Types

Binding Type	Properties
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 15-5 (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 15-5 (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 15-5 (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

15.3.6.2 About the Managed Beans Category

Review the options available in the Managed Beans category.

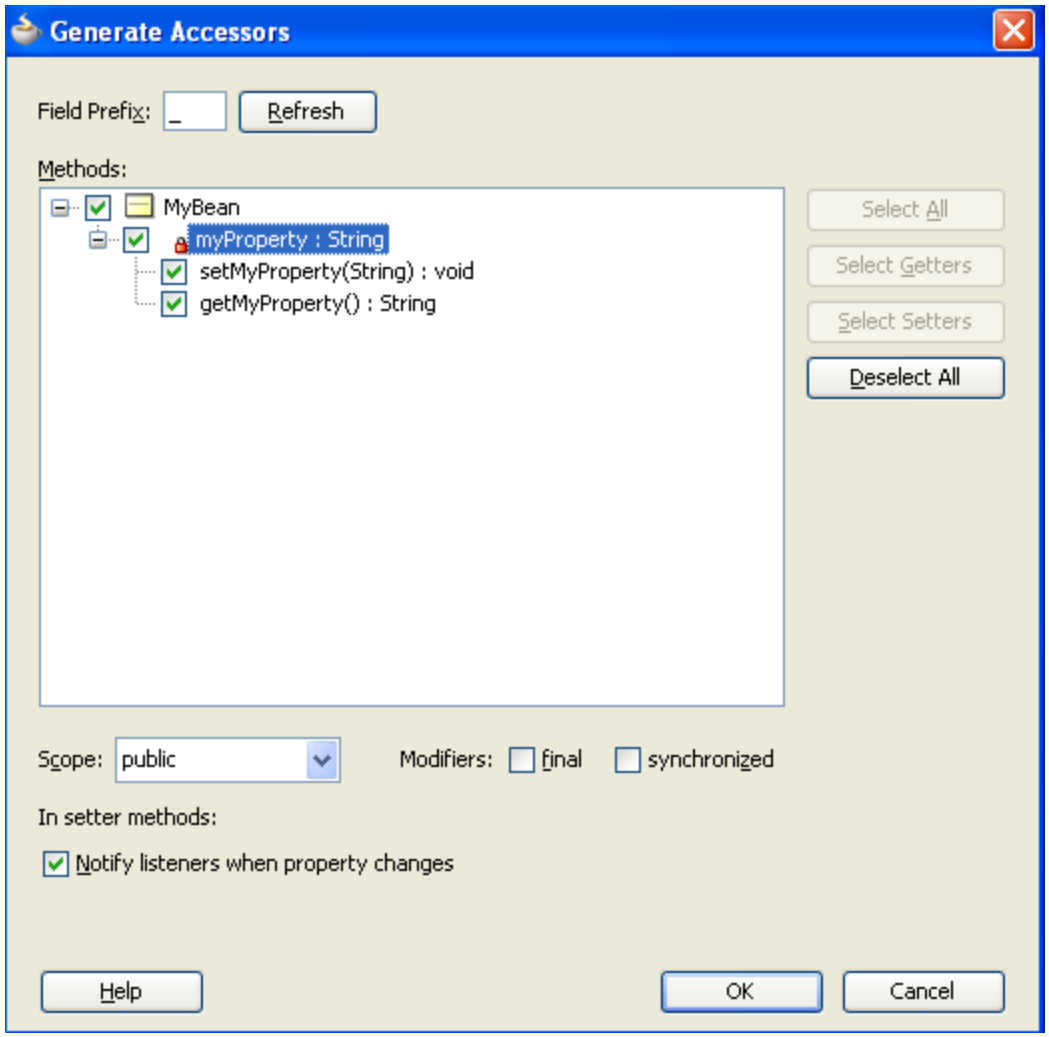
This section lists the options available under the Managed Beans category.

- `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 property accessors of the beans. To automatically generate the necessary code to source notifications from the property accessors of the beans, select the **Notify listeners when property changes** checkbox in the **Generate Accessors** dialog (see Figure 15-2).

Figure 15-2 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 About Data Change Events.

The following example illustrates how to retrieve a value bound to another managed bean attribute programmatically.

```
public void someMethod() {
   Object value = AdfmfJavaUtilities.evaluateELExpression(
                       "#{applicationScope.MyManagedBean.someProperty}");
   ...
}
```

The following 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.

15.3.6.3 About the Mobile Application Framework Objects Category

The Mobile Application Framework Objects category includes those objects that are defined in MAF, and can be referenced using EL.

The Mobile Application Framework Objects category lists various objects defined in MAF 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 lifetime and visibility of the variable. In order of decreasing visibility, they are application scope, page flow scope, and view scope. See 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.
- `deviceScope`: The `deviceScope` node exposes information about device properties. The `deviceScope` has application-level lifetime and visibility.
- `feature`: The `feature` node exposes feature-level data. The feature object exposes the `dataControlContextDepth`and `maximumDataControlContextDepth`properties. You can obtain values for these properties using `#{feature.dataControlContextDepth}`and `#{feature.maximumDataControlContextDepth}`. These two properties are read only.
- `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.feature.oracle.hello.SampleGroup1.label}"

  "#{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.

  The following example shows an example of AMX code for `viewControllerBundle`.

  ```
  <amx:loadBundle basename="mobile.ViewControllerBundle"
  var="viewcontrollerBundle"/>
  ```

15.3.7 About EL Events

EL events determine how the MAF AMX UI functions because they synchronize the updating of expressions with common terms. Any change in the value of an underlying expression generates an event to all listeners for that value.

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 following example 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 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. See 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 following 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.

15.3.8 How to Use EL Expressions Within Managed Beans

Use the EL expressions that JDeveloper provides. Create new EL expressions, access, set, or invoke EL expressions within a managed bean.

While JDeveloper 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.

The following 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);
}
```

The following 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,
                                                                        returnTy
pe,
                                                                        argTypes
);
  return methodExpression.invoke(AdfmfJavaUtilities.getAdfELContext(), argValues);
}
```

The following 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);
}
```

15.4 Creating and Using Managed Beans

Managed beans are Java classes registered with applications by means of configuration files, which when parsed, provide access to the properties and methods of the 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 properties and methods of the beans. 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}`.

15.4.1 How to Create a Managed Bean in JDeveloper

Use the procedure to create a managed bean, and register it with the MAF application.

You can create a managed bean and register it with the MAF application at the same time using the **Overview** editor for the `adfc-mobile-config.xml` file.

Before you begin:

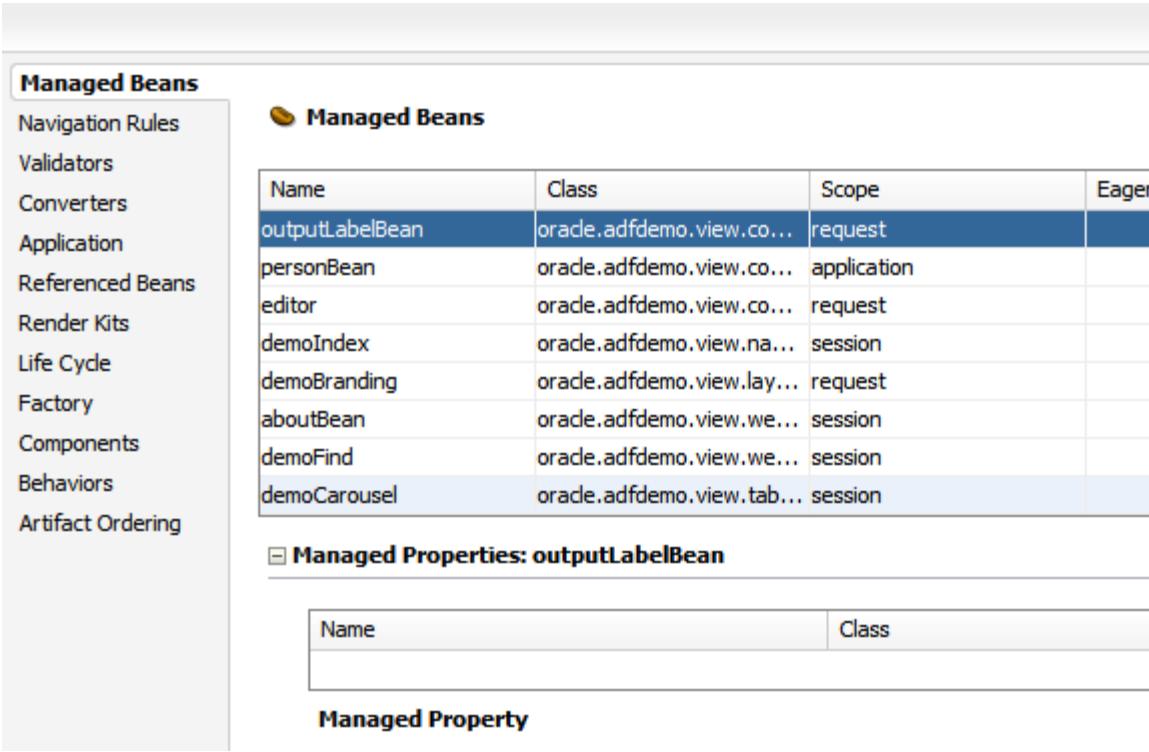
It may be helpful to have an understanding of managed beans. See Creating and Using Managed Beans.

To create and register a managed bean:

1. In the **Applications** window, 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 15-3 shows the editor for the `adfc-mobile-config.xml` file.

   Figure 15-3 Managed Beans in the adfc-mobile-config.xml File



4. Click the **Add** icon to add a row to the Managed Bean table.
5. In the **Create Managed Bean** dialog, enter values. Click **Help** for more information about using the dialog. Select the **Generate Class If It Does Not Exist** option if you want JDeveloper to create the class file for you. You can also open the **Create Managed Bean** dialog from the **Properties** window, by selecting one of the listener properties and clicking the **Edit** button. From there you can create a new managed bean and corresponding method.

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. See 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 the **New** icon 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 **Generate Accessors** menu item on the context menu of the code editor to generate the corresponding `get` and `set` methods for these bean properties.

15.4.2 What Happens When You Use JDeveloper to Create a Managed Bean

Creating a managed bean and generating the Java file in JDeveloper causes the IDE to create a stub class. Add logic to the page so that an EL expression can reference it using the given name of the stub class.

When you create a managed bean and elect to generate the Java file, JDeveloper creates a stub class with the given name and a default constructor. The following 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}
```

JDeveloper also adds a `managed-bean` element to the `adfc-mobile-config.xml` file (or to the task flow file that is being edited). The following 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>
```

15.5 Exposing Business Services with Data Controls

Create data controls to declaratively bind UI components to business services. Use the **Create Data Control** menu option to generate data controls.

Once you have the services of the application in place, use JDeveloper to create data controls that provide the information needed to declaratively bind UI components to those services.

You generate data controls with the Create Data Control menu item. 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.

15.5.1 How to Create Data Controls

Create an application workspace, add the business services, and then use the procedure to create a data control. You can also use the context menu of the class or object on which the data control is based to create a data control.

You create adapter-based data controls from within the Applications window of JDeveloper.

Before you begin:

It may be helpful to have a general understanding of using data controls. See 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 Creating a MAF Application.

To create a data control:

1. Right-click the top-level node for the data model project in the application workspace and select **New** and then **From Gallery**.
2. In the New Gallery, expand **Business Tier**, select **Data Controls**, select the type of data control that you want to create, and click **OK**.
3. 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 selecting Create Data Control.

15.5.2 What Happens in Your Project When You Create a Data Control

JDeveloper creates the definition file for the data control which opens in the overview editor, and the hierarchy of the file is displayed in the Data Controls panel.

When you create a data control, JDeveloper creates the data control definition file (`DataControls.dcx`), opens the file in the overview editor, and displays the hierarchy of the file in the Data Controls panel. 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.

15.5.2.1 DataControls.dcx Overview Editor

View the hierarchy of data control objects and the exposed methods of a data model in the overview editor for the `DataControls.dcx` file. Use the editor to edit the settings for a data control object.

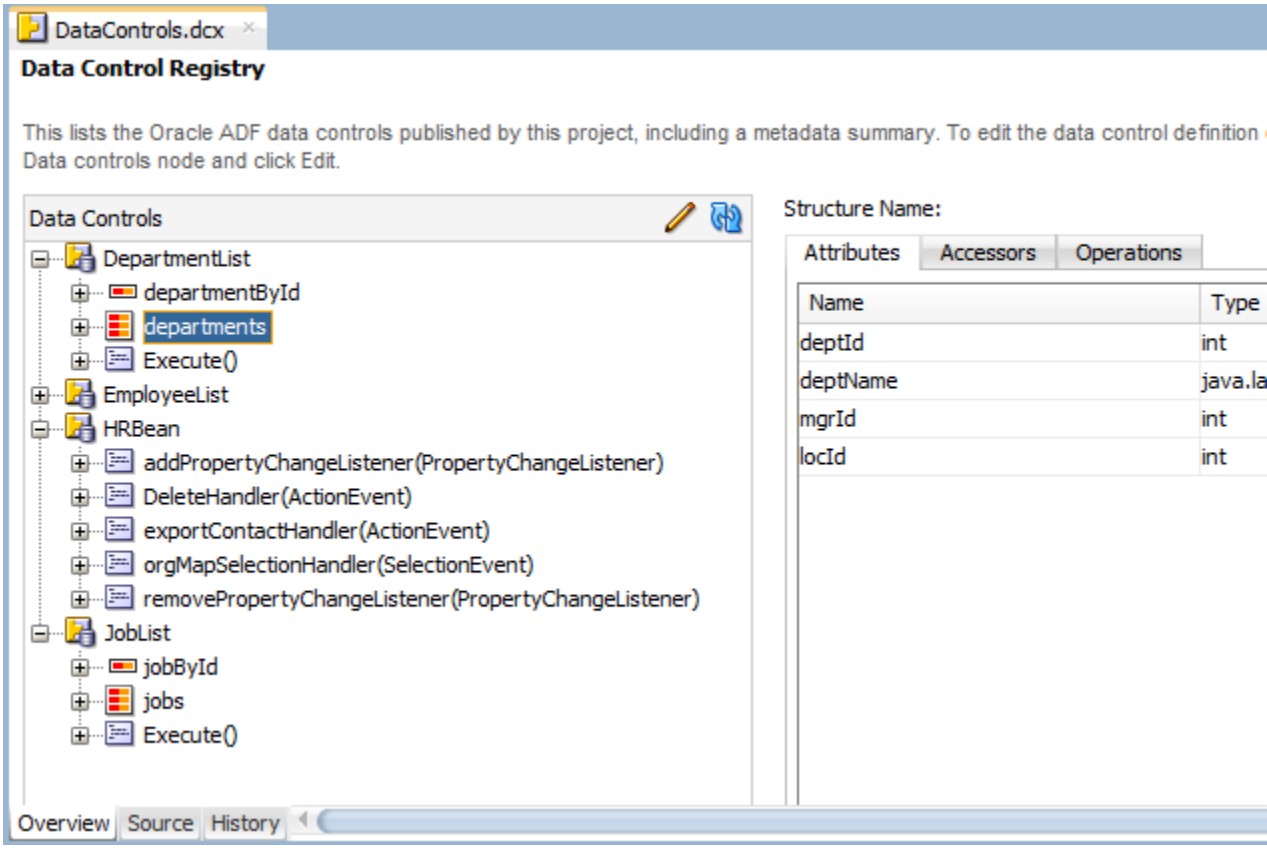
The overview editor for the `DataControls.dcx` file provides a view of the hierarchies of data control objects and exposed methods of your data model.

See Table 15-6 for a description of the icons that are used in the overview editor and Data Controls panel.

You can change the settings for a data control object by selecting the object and clicking the **Edit** icon. See How to Edit a Data Control.

Figure 15-4 shows the `DataControls.dcx` file in the overview editor.

Figure 15-4 DataControls.dcx File in the Overview Editor



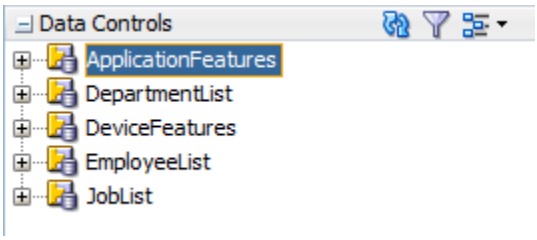
15.5.2.2 Data Controls Panel

The Applications window displays the Data Controls panel after a data control has been created. Drag nodes from the Data Controls panel to create databound UI components for a page.

The Data Controls panel serves as a palette, from which you can create databound UI components by dragging nodes from the Data Controls panel to the design editor for a page. The Data Controls panel appears in the Applications window once you have

created a data control. Figure 15-5 shows the Data Controls panel for a sample application.

Figure 15-5 Data Controls Panel



15.5.3 Data Control Built-in Operations

The data control framework defines a standard set of operations for data controls that are implemented by business services. The type of the data control and the functionality of the business service differ in the available 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 iterator of the bean collection.

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.

15.5.3.1 addXXX and removeXXX Methods of Data Control

Although most built-in operations operate automatically, built-in operations such as `Create` and `Delete` that use the `addXXX` and `removeXXX` methods require the addition of some method handlers to work. The `addXXX` and `removeXXX` methods automatically refresh the collection and trigger data change events.

Most of the built-in operations operate on the collection automatically. There are several operations that require the developer to write some method handlers in order to have these operations work. In order to use the `Create` operation, it is necessary for the developer to write method handler for `addXXX`. The `Create` operation also does the operation of `CreateInsert` because it inserts the record into the current collection. The `CreateInsert` operation is not used for MAF collections. Similarly, for `Delete` operation, the developer will have to write method handler for `removeXXX`.

The `addXXX` and `removeXXX` methods automatically refresh the collection and fire data change events, and the developer does not have to exclusively refresh the provider. The data objects are used by the built-in operations of the data control, such as `addXXX` and `removeXXX` methods, which are used by the `Create` and `Delete` built-in operations.

- `addXXX`: This method returns the unique Id to the Data Control framework. For example,

```
public void addDeptBean(DeptBean dept)
{
deptCollection.add(dept);
}
```

  where `DeptBean` is the class name and `dept` is the object.

- `removeXXX`: This method removes the object. For example,

```
public void removeDeptBean(DeptBean dept)
{
deptCollection.remove(dept);
}
```

  where `dept` is the object of class `DeptBean`.

The `CRUDDemo` sample application is located in the `PublicSamples.zip` file within the `jdev_install/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

15.6 Creating Databound UI Components from the Data Controls Panel

Drag and drop a data control object from the Data Controls panel onto the editor for a page to create a specific UI component. JDeveloper automatically creates the code and objects needed to bind the component to the data control that you selected.

In the Data Controls panel, each data control object is represented by a specific icon. Table 15-6 describes what each icon represents, where it appears in the Data Controls panel hierarchy, and what components it can be used to create.

Table 15-6 Data Controls Panel 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.
	Key Attribute	Represents an object attribute that has been declared as a primary key attribute, either in data control structure file or in the business service itself.	Label, text field, date, list of values, and selection list components.
	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.

Table 15-6 (Cont.) Data Controls Panel Icons and Object Hierarchy

Icon	Name	Description	Used to Create...
	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.

15.6.1 How to Use the Data Controls Panel

The Data Controls Panel provides a predefined set of UI components to build an application. Use the procedure to create UI components using the context menu that is displayed when you select a data collection from the panel.

JDeveloper provides you with a predefined set of UI components from which to select 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 Data Controls panel. See Creating Databound UI Components from the Data Controls Panel.

You will need to complete these tasks:

- Create a data control as described in How to Create Data Controls.
- Create a a MAF AMX page as described in Creating MAF AMX Pages.

To use the Data Controls panel to create UI components:

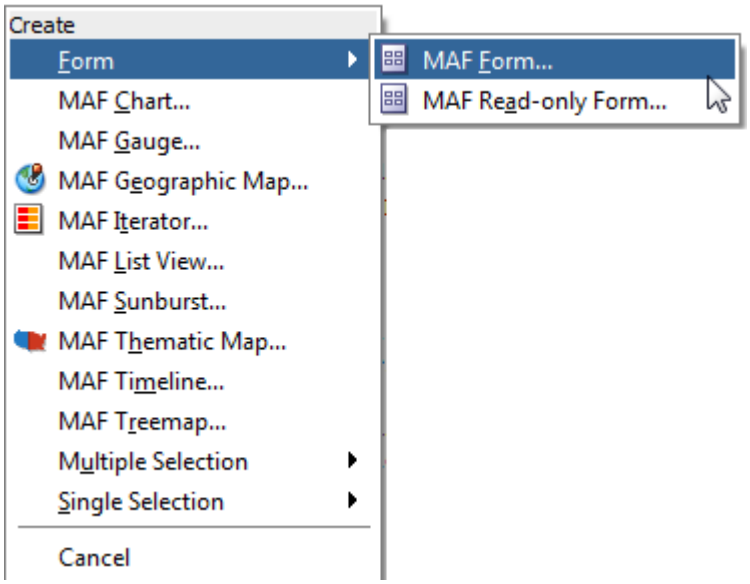
1. Select an item in the Data Controls panel and drag it onto the visual editor for your page. For a definition of each item in the panel, see Table 15-6.

2. From the ensuing context menu, select a UI component.

   When you drag an item from the Data Controls panel and drop it on a page, JDeveloper 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 15-6 shows the context menu displayed when a data collection from the Data Controls panel is dropped on a page.

Figure 15-6 Dropping Component From Data Controls Panel



Depending on the component you select from the context menu, JDeveloper 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, JDeveloper 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 JDeveloper 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 Data Controls panel. 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, JDeveloper automatically deletes those binding objects for you (automatic deletion of binding objects will not happen if you use the source editor).

15.6.2 What Happens When You Use the Data Controls Panel

JDeveloper creates files to define the binding context for the application, a registry for the bindings file, and the binding container for every page in the application. Also added are components of the page, libraries, files, and configuration elements required by the UI components.

When an application is built using the Data Controls panel, JDeveloper does the following:

- Creates a `DataBindings.cpx` file in the default 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. See 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.

  Tip:

  You can set the package configuration (such as name and location) in the ADF Model settings page of the Project Properties dialog (accessible by double-clicking the project node).

  The page definition file (*`pageName`*`PageDef.xml`) defines the binding container for each page in the view layer of an application. The binding container provides runtime access to all the binding objects for a page. For more information about the page definition file, see 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. See 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 What Happens When You Create a MAF Application.

15.7 What Happens at Runtime: How the Binding Context Works

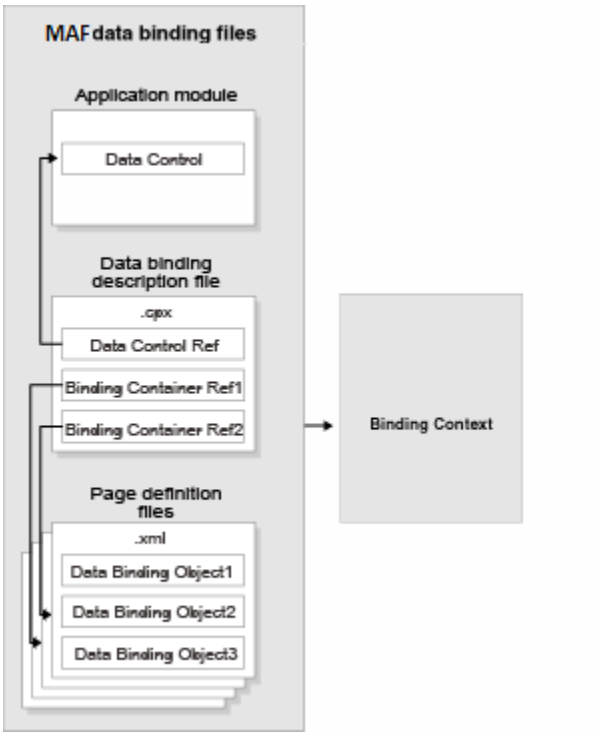
At runtime, a binding context manages the interactions between MAF bindings on a page and the business services. Created from the `DataBindings.cpx` file and the page definition files, the binding context is a map that contains references for data control or binding container objects on demand.

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 15-7. 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 What You May Need to Know About Generated Drag and Drop Artifacts.

Figure 15-7 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 is not directly linked to the UI.

For more information on working with bindings in MAF, see the following:

- What You May Need to Know About Generated Bindings
- Using the MAF AMX Editor Bindings Tab
- What You May Need to Know About Removal of Unused Bindings

15.8 Configuring Data Controls

Create a data control, and then create and modify data control structure files that contain its elements. Use the overview editor for the `.dcx` file to generate a data control structure file.

When you create a data control, a standard set of values and behaviors are assumed for the data control. For example, the data control determines how the label for an attribute will display in a client. You can configure these values and behaviors by creating and modifying data control structure files that correspond to the elements of the data control. You first generate a data control structure file using the overview editor for the `.dcx` file.

15.8.1 How to Edit a Data Control

Create a data control, and then use the procedure to edit the data control.

You can make a data control configurable by using the overview editor for the `DataControls.dcx` file to create data control structure files that correspond to objects encompassed by the data control. You can then edit the individual data control structure files.

Before you begin:

It may be helpful to have a general understanding of data control configuration. See Configuring Data Controls.

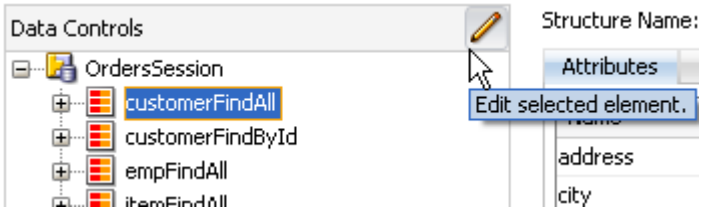
You will need to complete this task:

Create a data control, as described in How to Create Data Controls.

To edit a data control:

1. In the Applications window, double-click **DataControls.dcx**.
2. In the overview editor, select the object that you would like to configure and click the **Edit** icon to generate a data control structure file, as shown in Figure 15-8.

Figure 15-8 Edit Button in Data Controls Registry



3. In the overview editor of the data control structure file, make the desired modifications.

15.8.2 What Happens When You Edit a Data Control

When a data control is edited, JDeveloper creates a data control structure file for the relevant data collection, and opens the file in the overview editor. You can then add or edit configurations for the data control.

When you edit a data control, JDeveloper creates a data control structure file that contains metadata for the affected collection and opens that file in the overview editor. This file stores configuration data for the data control that is specific to that collection, such as any UI hints or validators that you have specified for the data object.

A data control structure file has the same base name as the data object with which it corresponds. For example, if you click the **Edit** icon when you have a collection node selected that corresponds with the `Customer.java` entity bean, the data control structure file is named `Customer.xml`. The data control structure file is generated in a package that corresponds to the package of the bean class, but with `persdef` prepended to the package name. For example, if the `Customer.java` bean is in the `model` package, the `Customer.xml` data control definition file is generated in the `persdef.model` package. Once a data control structure file has been generated, you can use the overview editor for that file to make further configurations.

A data control structure file contains the following information:

- Attributes: Describes all of the attributes on the service. For example, for entity beans, there is an attribute for each bean property that is mapped to a database column. You can also add transient attributes. You can set UI hints that define how these attributes will display in the UI. You can also set other properties, such as whether the attribute value is required, whether it must be unique, and whether it is visible. For more information, see Working with Attributes.

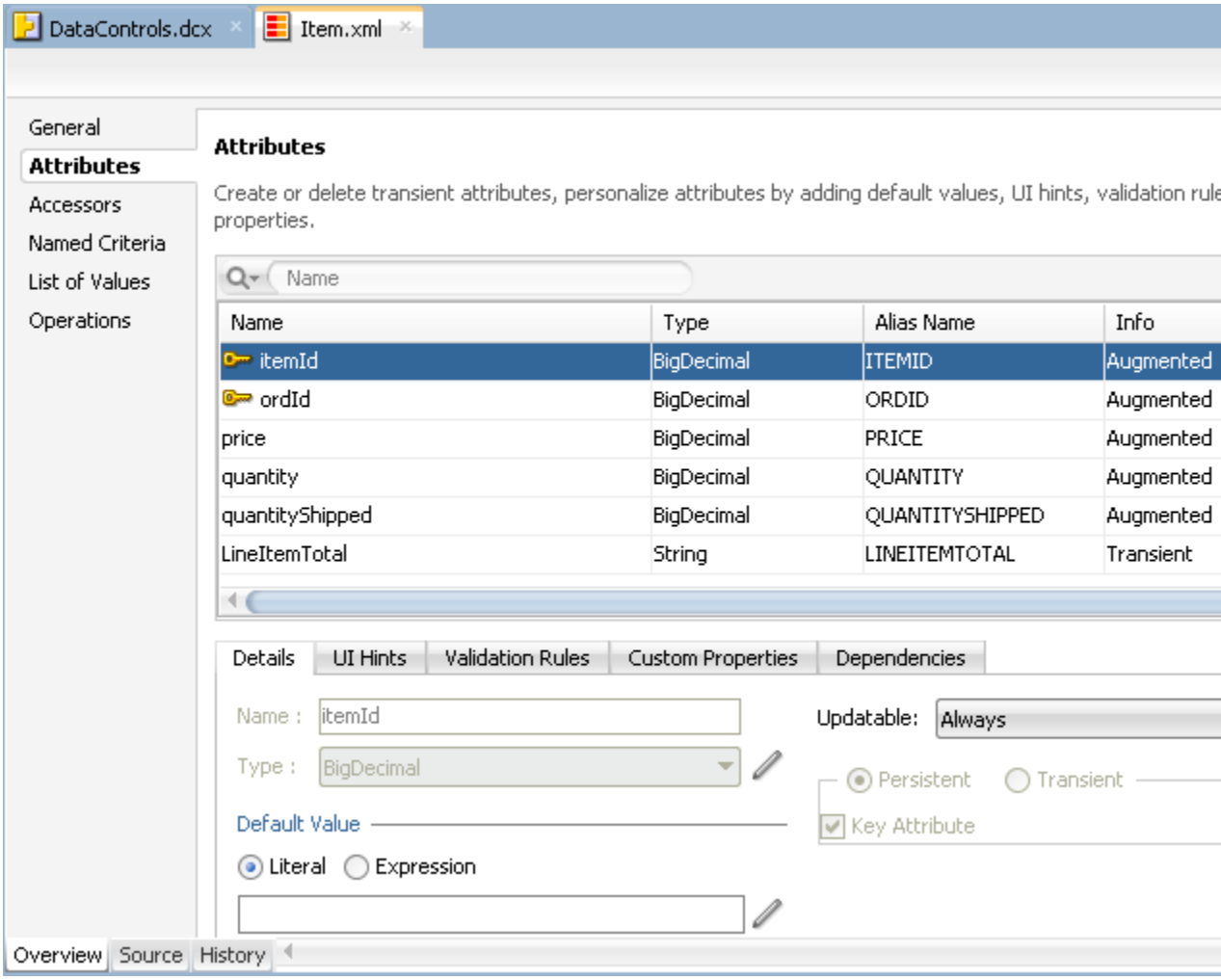
  You can also set validation for an attribute and create custom properties. For more information on validation, see Validating Attributes.

- Accessors: Describes data control elements that return result sets.

- Operations: Describes methods on the data object that are used by the built-in operations of the data control, such as `add` and `remove` methods, which are used by the Create and Delete built-in operations, respectively.

Figure 15-9 shows the data control structure file for the `Item` bean.

Figure 15-9 Data Control Structure File in the Overview Editor



Note:

The overview editor of a data control structure file shows all of the attributes, accessors, and operations that are associated with the data object. However, the XML source of the data control structure file only contains definitions for elements that you have edited. The base elements are introspected from the data object. Also, when you make changes to the underlying data object, the data control inherits those changes.

15.8.3 What You May Need to Know About MDS Customization of Data Controls

Data control structure files are necessary for objects within a data control if those objects are to be made available to Oracle Metadata Services customization. Edit and add metadata to every data control object so that a data control structure file is generated for it.

If you wish for all of the objects that are encompassed by the data control to be available for Oracle Metadata Services (MDS) customization, the packaged application must contain data control structure files for those objects.

When you create a data control based on the adapter framework, data control structure files are not generated by default, since they are not needed by the data control if you do not add metadata to a given object. Typically, a data control structure file is only generated for a data control object once you edit the data control to add declarative metadata (such as UI hints or validators) to that object, as described in How to Edit a Data Control. To create data control structure files for each data control object, you need to repeat that procedure for each data control object.

See Customizing MAF Application Artifacts with MDS .

15.9 Working with Attributes

When a data control is created and a data control structure file generated for objects, you can configure the functionality of the persistent attributes of the data objects. Use the Attributes page of the overview editor of the data control structure file to configure properties.

When you create a data control for your business services, you can create a data control structure file for an individual data object in which you can declaratively augment the functionality of the persistent attributes of the data object. For example, you can create validation rules and set UI hints to control the default presentation of attributes in UI components.

You set these properties on the Attributes page of the overview editor of the data control structure file. For information on creating a data control structure file, see How to Edit a Data Control.

15.9.1 How to Designate an Attribute as Primary Key

In the data control structure file of a data object, you can specify an attribute as a primary key for the data object. Use the procedure to set an attribute as a primary key for a data object that has not inherited a set primary key.

In the overview editor for the data control structure file of a data object, you can designate an attribute as a primary key for that data object if you have not already done so in the underlying class of the data object.

Before you begin:

It may be helpful to have an understanding of how you set attribute properties. See Working with Attributes.

You will need to complete this task:

Create the desired data control structure files as described in How to Edit a Data Control.

To set an attribute as a primary key:

1. In the Applications window, double-click the desired data control structure file.
2. In the overview editor, click the **Attributes** navigation tab.
3. On the Attributes page, select the attribute you want to designate as the primary key and then click the **Details** tab.
4. On the Details page, set the **Key Attribute** property.

Note:

If the attribute has already been designated as the primary key in the class, the data control inherits that setting and the **Key Attribute** checkbox will be selected. However, in this case, you can not deselect the **Key Attribute** option.

15.9.2 How to Define a Static Default Value for an Attribute

In the overview editor of the data control structure file for a data object, you can specify a static default value for an attribute. Use the procedure to set the value type to **Literal,** and define a static default value for an attribute.

The **Value** field in the **Details** section allows you to specify a static default value for the attribute when the value type is set to **Literal**. For example, you might set the default value of a `Status` attribute of a `ServiceRequest` entity bean to `Open`, or set the default value of a `UserRole` attribute of a `User` bean to `user`.

Before you begin:

It may be helpful to have an understanding of how you set attribute properties. See Working with Attributes.

To define a static default value for an attribute:

1. In the Applications window, double-click the desired data control structure file.
2. In the overview editor, click the **Attributes** navigation tab.
3. On the Attributes page, select the attribute you want to edit, and then click the **Details** tab.
4. On the Details page, select the **Literal** option.
5. In the text field below the **Literal** option, enter the default value for the attribute.

15.9.3 How to Set UI Hints on Attributes

Setting UI hints on attributes displays and labels them consistently, and makes the attributes more usable and localizable for the UI components that use them. Use the procedure to set a UI hint using the **Attributes** option in the Applications window.

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 to use 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 overview editor for the data control structure file of the data object, which is accessible from the Applications window.

Before you begin:

It may be helpful to have an understanding of how you set attribute properties. See Working with Attributes.

You will need to complete this task:

Create the desired data control structure files as described in How to Edit a Data Control.

To set a UI hint:

1. In the Applications window, double-click the desired data control structure file.
2. In the overview editor, click the **Attributes** navigation tab.
3. On the Attributes page, select the attribute you want to edit, and then click the **UI Hints** tab.
4. In the **UI Hints** section, set the desired UI hints.

15.9.4 What Happens When You Set UI Hints on Attributes

UI hints on an attribute are stored as properties to which tags are added. Values for properties are stored in a resource bundle file, which is generated if it is not found.

When you set UI hints on an attribute, those hints are stored as properties. Tags for the properties are added to the data control structure file of the data object and the values for the properties are stored in a resource bundle file. If the resource bundle file does not already exist, it is generated in the package of the data control and named according to the project name when you first set a UI hint.

The following example shows the code for the `price` attribute in the `Item.xml` data control structure file, including tags for the Label and Format Type hints which have been set for the attribute.

```
<PDefAttribute
  Name="price">
  <Properties>
    <SchemaBasedProperties>
      <LABEL
        ResId="${adfBundle['model.ModelBundle']['model.Item.price_LABEL']}"/>
      <FMT_FORMATTER ResId="${adfBundle['model.ModelBundle']
                                       ['model.Item.price_FMT_FORMATTER']}"/>
    </SchemaBasedProperties>
  </Properties>
</PDefAttribute>
```

The following example shows the corresponding entries for the Label and Format Type hints in the `ModelBundle.properties` resource bundle file, which contains the values for all of the localizable properties of the project.

```
model.Item.price_LABEL=Price
. . .
model.Item.price_FMT_FORMATTER=oracle.jbo.format.DefaultCurrencyFormatter
```

15.9.5 How to Access UI Hints Using EL Expressions

Use EL expressions to display hint values as data on a page so that you can access UI hints. Drop databound components onto a page to create binding instances that provide access to UI hints.

You can access UI hints using EL expressions to display the hint values as data in a page. You access UI hints through the binding instances that you create after dropping databound components onto your pages.

The following example was produced using the DeviceFeatures data control. It shows the EL expression that is produced by dragging and dropping Contact as a MAF form and only keeping the `displayName` and `nickname` fields. The labels in bold are examples of the retrieval of UI hints using EL.

```
<amx:panelFormLayout id="pfl2">
      <amx:inputText value="#{row.bindings.displayName.inputValue}"
                     label="#{bindings.Contact.hints.displayName.label}"
id="it9"/>
      <amx:inputText value="#{row.bindings.nickname.inputValue}"
                     label="#{bindings.Contact.hints.nickname.label}"
                     id="it10"/>
</amx:panelFormLayout>af:panelHeader id="ph1"
```

15.10 Creating and Using Bean Data Controls

A bean data control serves as a metadata wrapper for a bean class, and exposes the code elements of the bean as data control objects that are used to bind code elements to UI components. Create a Java bean data control by using the **Create Data Control** option in the Applications window.

A bean data control serves as a metadata wrapper for a bean class and exposes the code elements of the bean 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). To create a Java bean data control, right-click a Java class file (in the Applications window), and select Create Data Control. You create Java bean data controls from within the Applications window of JDeveloper.

Before you begin:

It may be helpful to have a general understanding of data controls. See How to Create Data Controls.

Note:

If the Java bean 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 About Data Change Events.

For a sample of to build CRUD operations using the local SQLite database and Java bean data controls, see the MAF sample application called CRUDDemo located in the

`PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

15.10.1 What You May Need to Know About Serialization of Bean Class Variables

Define a chain of nested objects as transient to prevent serialization and the creation of cyclic objects from object nesting. You can serialize and deserialize Java objects into JSON objects by using the `JSONBeanSerializationHelper` class.

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. See the `oracle.adfmf.framework.api.JSONBeanSerializationHelper` class in *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();
```

15.11 Sharing Instances of Data Controls Across Application Features

You can share an instance of a data control across application features so that a data control accessed by multiple application features is loaded into memory once for all application features that access the shared instance.

Doing this improves the performance of your MAF application. Shared instance data controls are loaded once into memory rather than for each individual application feature that accesses the data control.

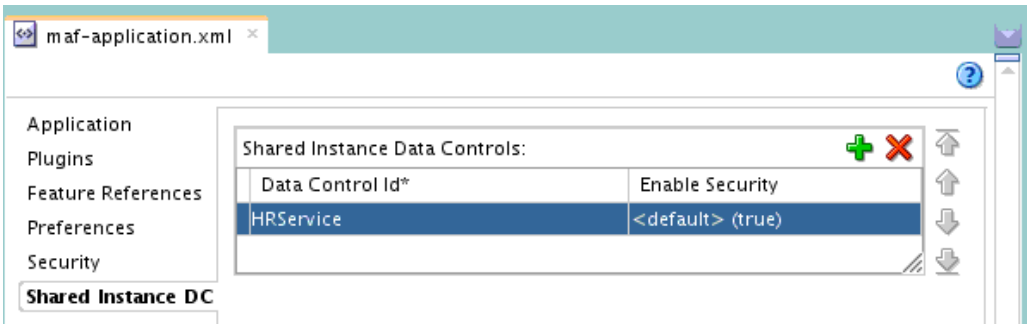
MAF propagates data changes that end users make in one application feature that uses a shared instance data control to other application features that use the shared instance data control.

By default, data controls are not shared across application features in your MAF application. You must add each data control that you want to share to the Shared Instance DC page of the overview editor for the `maf-application.xml` file. Figure 15-10 shows the Shared Instance DC page where the `HRService` data control has been designated as a shared instance data control.

By default, MAF only permits access to the shared instance data control from application features that have security enabled. You can disable this restriction and

allow application features used by unauthenticated users to access a shared instance data control by setting the **Enable Security** property to `false`.

Figure 15-10 Shared Instance Data Control Page in maf-application.xml's Overview Editor



Data controls that your MAF application accesses from a task flow with a `data-control-context` value of `isolated` will not be shared, even if you have designated the data control as a shared instance data control.

15.11.1 How to Share Instances of Data Controls Across Application Features

You share instances of data controls by adding each data control that you want to share to the Shared Instance DC page of the overview editor for the `maf-application.xml` file.

To share instances of data controls across application features:

1. In the Applications window, expand the **Application Resources** panel.
2. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the `maf-application.xml` file and in the overview editor that appears, click the **Shared Instance DC** navigation tab.
4. In the Shared Instance DC page, click the **Add** icon.
5. In the Insert sharedInstanceDataControl dialog, select the ID of the data control from the dropdown list.
6. Click **OK**.

15.11.2 What Happens When You Share Instances of Data Controls Across Application Features

At runtime, one instance of a data provider is created for a data control that you designate as a shared instance data control. This means, for example, that a Java class is instantiated one time only when the associated shared instance data control is first referenced by a feature.

All application features that reference the shared instance data control share this instance of the Java class. Updates to the shared instance data control are delivered to all application features if the security setting permits it. If, for example, the shared instance data control is only available to secured application features (the default setting), updates in the form of data change or data provider events will be sent to

secured application features and MAF throws an error when an unsecured application feature attempts to access the shared instance data control.

One exception to the just-described behavior is where your MAF application accesses the shared instance data control from a task flow with a `data-control-context` value of `isolated`. In this scenario, more than one instance of a data provider is created.

JDeveloper adds entries to the `maf-application.xml` file that identifies each data control that you designated as shared. The following example shows entries for two shared instance data controls. The first shared instance data control (`HRService`) can only be accessed by application features that have security enabled (the default setting) while the second (`CustomerService`) can be accessed by unsecured application features.

Example 15-1 Shared Instance Data Controls Defined in maf-application.xml File

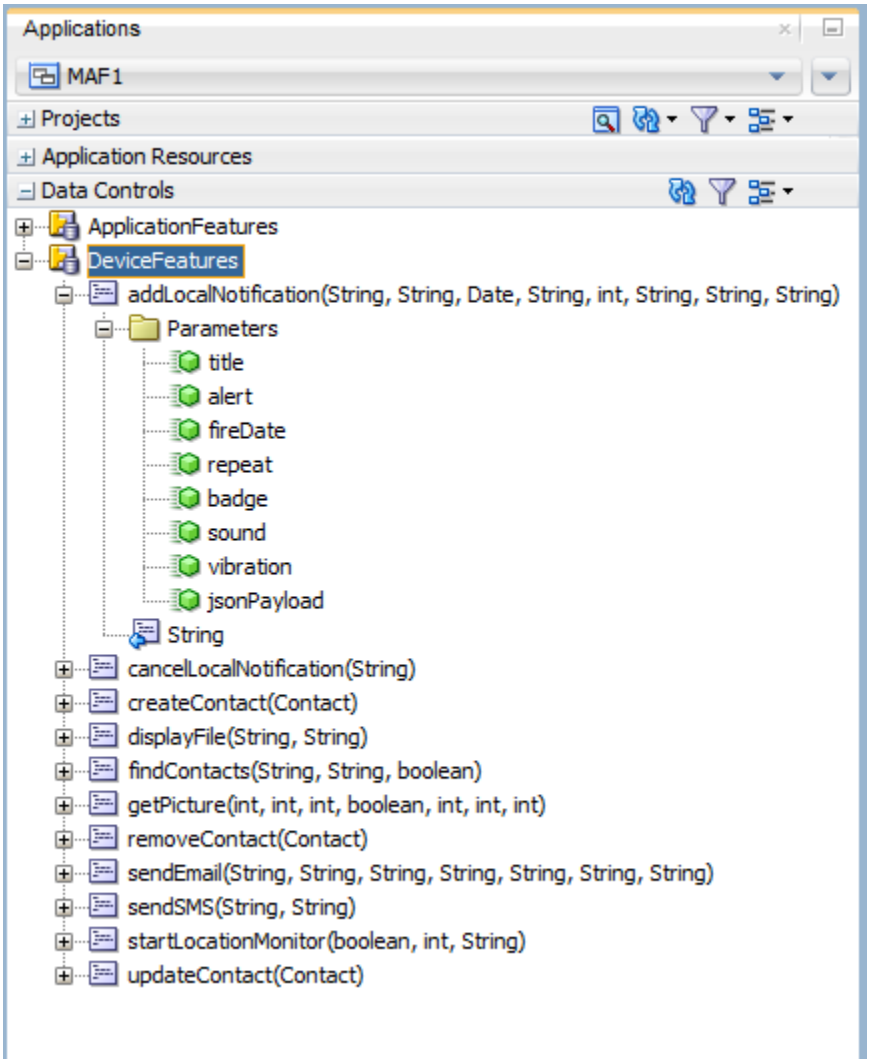
```
<adfmf:application ...>
  ...
  <adfmf:sharedInstanceDataControls>
    <adfmf:sharedInstanceDataControl dataControlId="HRService" id="sidc1"/>
    <adfmf:sharedInstanceDataControl dataControlId="CustomerService" id="sidc2"
securityEnabled="false"/>
  </adfmf:sharedInstanceDataControls>
</adfmf:application>
```

15.12 Using the DeviceFeatures Data Control

When you create a new MAF application, MAF presents the`DeviceFeatures` data control in the Data Controls panel. Use the available operations in the data control on a MAF AMX page to manage user contacts on a device, use email and SMS text messages, ascertain the location of a device, use a device camera, or retrieve the images stored on a device.

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 panel 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 camera, and retrieve the images stored in a file system of a device. 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 15-11 MAF DeviceFeatures Data Control in the Overview Editor



The DeviceFeatures data control appears in the Data Controls panel automatically when you create an application using the MAF application template. Figure 15-11 shows the DeviceFeatures data control in the overview editor. The following methods are available:

- `addLocalNotification`
- `cancelLocalNotification`
- `createContact`
- `displayFile`
- `findContacts`
- `getPicture`
- `removeContact`
- `sendEmail`
- `sendSMS`

- `startLocationMonitor`
- `updateContact`

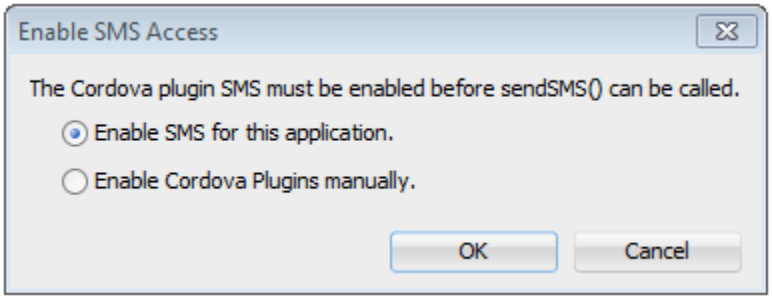
After you create a page, you can drag DeviceFeatures data control methods (or other objects nested within those methods) from the Data Controls panel to a MAF AMX view to create command buttons and other components that are bound to the associated functionality. You can accept the default bindings or modify the bindings using EL. You can also use JavaScript or Java to implement or configure functionality. See How to Add Data Controls to a 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.

[[dev ER 19938192 addressed below]]

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 select the appropriate option in the dialog that JDeveloper displays when you drag and drop an operation that does not have the associated plugin enabled in the MAF application. For example, JDeveloper displays the dialog in Figure 15-12 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 15-12 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 15-13 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 Using Plugins in MAF Applications .

Figure 15-13 DeviceFeatures Data Control Operation Requires Plugin

15.12.1 How to Use the getPicture Method to Enable Taking Pictures

The `DeviceFeatures` data control provides the `getPicture` method to take a photo with the device camera or retrieve an existing image. Use the procedure to customize a `getPicture` operation using parameters such as `sourceType`, `destinationType`, `quality`, and `allowEdit`.

The DeviceFeatures data control includes the `getPicture` method, which enables MAF applications to leverage a device camera and photo library so end users can take a photo or retrieve an existing image. At the end of this section there are examples that show:

- JavaScript code that enables an end user to take a picture with a device camera. Java code that allows the user to take a picture with a device camera.
- Java code that allows the user to retrieve a previously-saved image.

For information about the `getPicture` method, see the DeviceDataControl class in the MAF Javadoc 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 the device 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 DeviceFeatures data control in the Data Controls panel and drop it on the page as a **Button**.

   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.

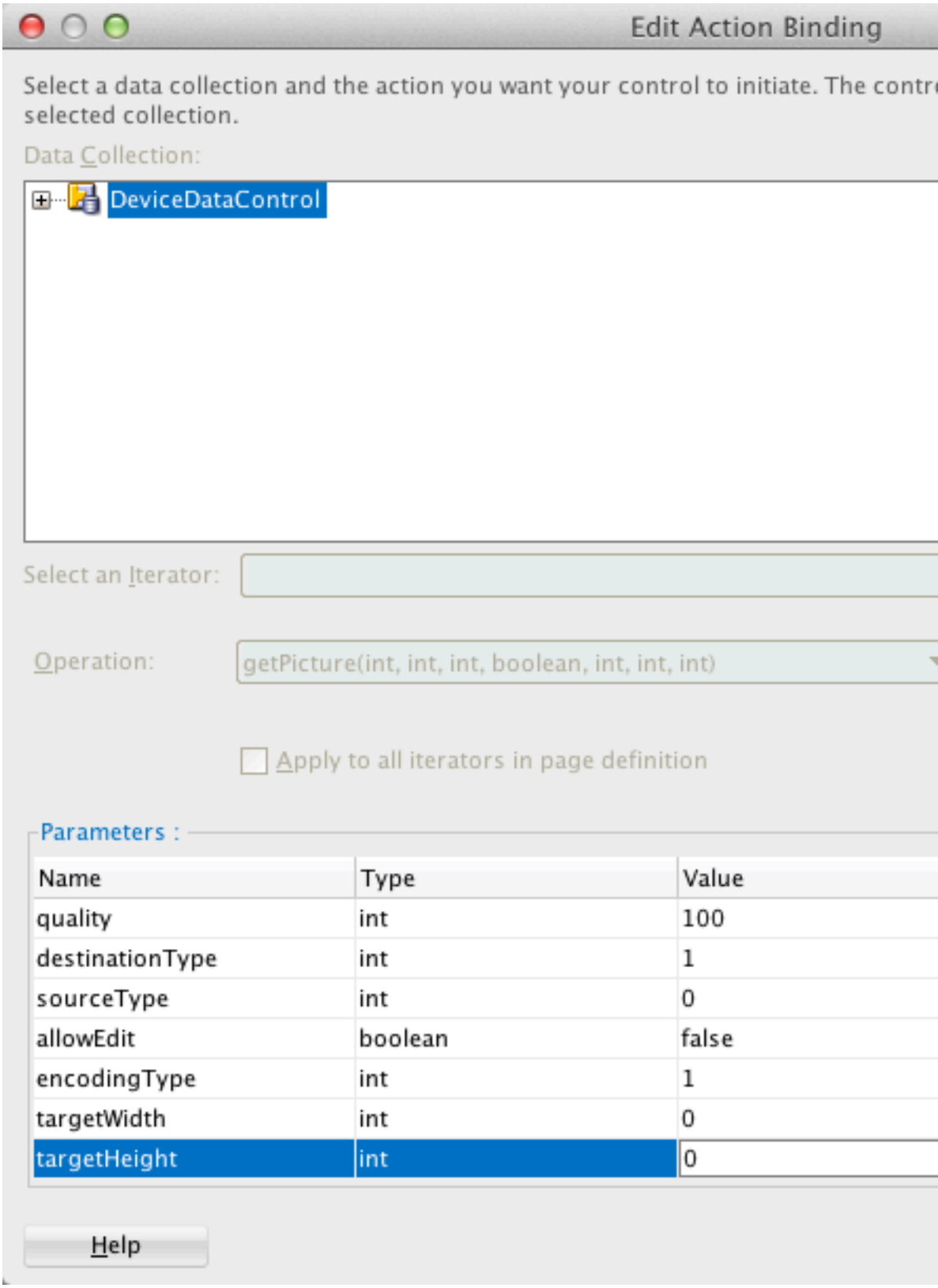
3. Drag the return value of **getPicture** and drop it on the page as an **Output Text**.

4. From the Common Components panel, drag an **Image** from the Component Palette and drop it on the page.

5. Set the `source` attribute of the Image to the return value of the `getPicture` operation. The bindings expression should be: `#{bindings.Return.inputValue}`.

Figure 15-14 shows the bindings for displaying an image from the user's photo library:

Figure 15-14 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 temporary directory of the application.
  - The contents of the temporary directory of the application 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.

The following example shows JavaScript code that allows the user to take a picture with a device 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);
}
```

The following example shows Java code that allows the user to take a picture with a device 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);
```

The following 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);
```

15.12.2 How to Use the SendSMS Method to Enable Text Messaging

Use the `sendSMS` method in the `DeviceFeatures` data control to send and receive SMS messages with the Short Message Service (SMS) text messaging interface of a device. The `sendSMS` operation is customizable.

The DeviceFeatures data control includes the `sendSMS` method, which enables MAF applications to leverage the Short Message Service (SMS) text messaging interface of a device so users can send and receive SMS messages. MAF enables you to display the SMS interface of a device 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 user can select 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 user can actually send the SMS.

The timeout value for the `sendSMS` method is set to 5 minutes. If the operation of the device takes longer than the timeout allowed, a timeout error appears.

In Android, if an 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.

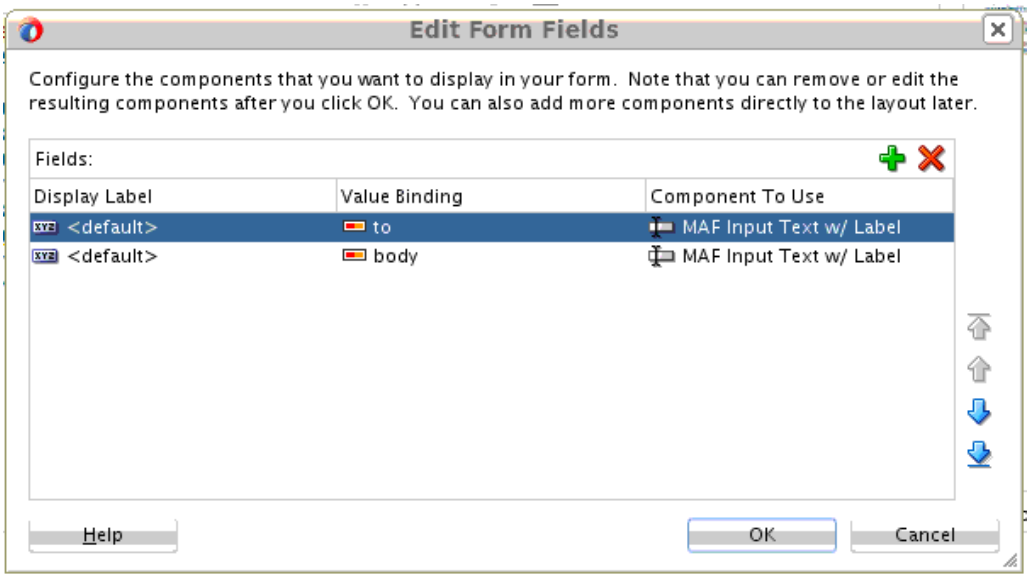
MAF applications on the Universal Windows Platform do not support the use of SMS text messaging at present.

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 Data Controls panel 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 SMS interface of the device, which will display an SMS that is ready to send with all of the specified fields pre-populated.

Figure 15-15 shows the bindings for sending an SMS using an editable form on the page.

Figure 15-15 Bindings for Sending an SMS Using an Editable Form at Design Time



The following examples show code examples that allow the user to send an SMS message with the text messaging interface of a device.

For information about the `sendSMS` method, see the `DeviceDataControl` class in the MAF Javadoc and refer to the Cordova documentation (`http://cordova.apache.org/`).

The following example shows JavaScript code that allows the user to send an SMS message:

```
adf.mf.api.sendSMS({to: "5551234567", body: "This is a test message"});
```

The following example shows Java code that allows the user to send an SMS message:

```
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");
```

15.12.3 How to Use the sendEmail Method to Enable Email

The `DeviceFeatures` data control provides the `sendEmail` method to send and receive email messages from a configured email account with the email messaging interface of a device. The `sendEmail` operation is customizable.

The DeviceFeatures data control includes the `sendEmail` method, which enables MAF applications to leverage the email messaging interface of a device so users can send and receive email messages. MAF enables you to display the email interface of a device 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 examples at the end of this section.

After the email interface of a device is displayed, the user can select 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 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 operation of the device takes longer than the timeout allowed, a timeout error is displayed.

Note:

In Android, if an 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.

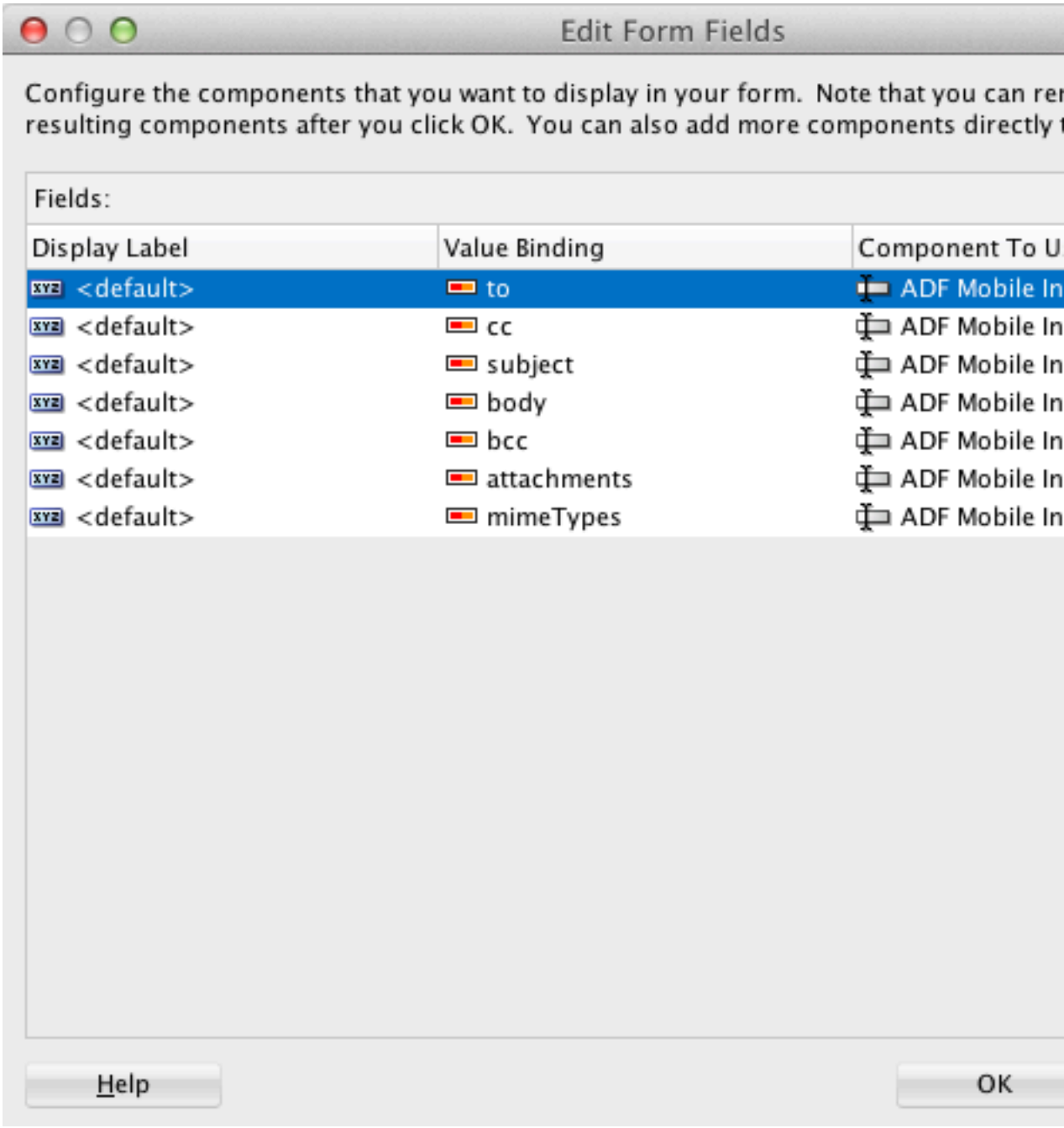
To customize a sendEmail operation using the DeviceFeatures data control:

In JDeveloper, drag the `sendEmail` operation from the DeviceFeatures data control in the Data Controls panel to 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 email interface of a device, which will display an email ready to send with all of the specified fields pre-populated.

Figure 15-16 shows the bindings for sending an email using an editable form on the page.

Figure 15-16 Bindings for Sending an Email Using an Editable Form at Design Time



Following are code examples that allow the user to send an email message with the email interface of the device

For information about the `sendEmail` method, see the `DeviceDataControl` class in the MAF Javadoc and refer to the Cordova documentation (http://cordova.apache.org/).

The following example shows JavaScript code that allows the user to send an email message:

```
// 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.
```

The following example shows Java code that allows the user to send an email message:

```
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.co
m",
                                        "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.co
m",
                                        "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.co
m",
                                        "path/to/file1.txt,path/to/file2.png",
                                        ",image/png");
```

15.12.4 How to Use the createContact Method to Enable Creating Contacts

The `DeviceFeatures` data control provides the `createContact` method to manage contacts that you can add and save in an address book. Use the procedure to customize a `createContact` operation.

The DeviceFeatures data control includes the `createContact` method, which enables MAF applications to leverage the interface of a device and file system for managing contacts so users can create new contacts to save in the address book of the device. MAF enables you to display the interface of the device 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 examples at the end of this section.

For more information about the `createContact` method and the `Contact` object, see the `DeviceDataControl` class in the MAF Javadoc and refer to the Cordova documentation (`http://cordova.apache.org/`). Also see 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 operation of the device 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 JDeveloper, drag the `createContact` operation from the DeviceFeatures data control in the Data Controls panel 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 Java bean 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 following code 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 Data Controls panel 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.

```
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;
}
```

The following examples show code examples that allow the user to create contacts on devices.

The following 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);
}
```

The following 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);
```

15.12.5 How to Use the findContacts Method to Enable Finding Contacts

The `DeviceFeatures` data control provides the `findContacts` method to help you locate contacts in an address book using properties such as id, name, birthday, photos, and address. Use the procedures to customize a `findContacts` operation.

The DeviceFeatures data control includes the `findContacts` method, which enables MAF applications to leverage the interface of a device and file system for managing contacts so users can find one or more contacts from the address book of the device . MAF enables you to display the interface of the device 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 MAF Javadoc 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 address book application of the operating system 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 operation of the device takes longer than the timeout allowed, a timeout error is displayed.

To customize a findContacts operation using the DeviceFeatures data control:

1. In JDeveloper, drag the `findContacts` operation from the DeviceFeatures data control in the Data Controls panel 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 Data Controls panel 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 following example shows possible argument values for the `findContacts` method.

```
// 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.

The following JavaScript example shows how to find a contact by family name and get the contact's name, phone numbers, email, addresses, and note.

```
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);
}
```

The following Java example shows how to find a contact by family name and get the contact's name, phone numbers, email, addresses, and note.

```
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);
```

15.12.6 How to Use the updateContact Method to Enable Updating Contacts

The `DeviceFeatures` data control provides the `updateContact` method to help you add or update contact information such as phone number, email type, and postal code in the address book of the device. Use the procedures to customize an `updateContact` operation.

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 users can update contacts in the address book of the device. MAF enables you to display the interface of the device 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 as shown in the example at the end of this section.

For more information about the `updateContact` method and the `Contact` object, see the `DeviceDataControl` class in the MAF Javadoc and refer to the Cordova documentation (`http://cordova.apache.org/`). Also see 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 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 JDeveloper, drag the **updateContact** operation from the DeviceFeatures data control in the Data Controls panel 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 Java bean 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. 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 Data Controls panel 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 following examples show how to update and add a contact's phone number.

The following JavaScript example shows how to use `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);
}
```

The following JavaScript example shows how to use `updateContact` to add a phone number to 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);
}
```

The following Java code example shows how to use `updateContact` 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]);
```

The following Java example shows how to use `updateContact` to add a phone number to 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 operation of the device takes longer than the timeout allowed, a timeout error is displayed.

15.12.7 How to Use the removeContact Method to Enable Removing Contacts

The `DeviceFeatures` data control provides the `removeContact` method to remove contacts, found using `findContacts`, from an address book on a device. Use the procedures to customize a `removeContact` operation.

The DeviceFeatures data control includes the `removeContact` method, which enables MAF applications to leverage the interface of a device and file system for

managing contacts so users can remove contacts from the address book of the device. MAF enables you to display the interface of the device and optionally pre-populate the `removeContact` fields. The `removeContact` method takes in a `Contact` object as a parameter, as shown in the examples at the end of this section.

Note:

The `Contact` object that is needed as the input parameter can be found using the `findContacts` method as described in How to Use the findContacts Method to Enable Finding Contacts.

To customize a removeContact operation using the DeviceFeatures data control:

1. In JDeveloper, drag the **removeContact** operation from the DeviceFeatures data control in the Data Controls panel 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 Data Controls panel 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.

The examples at the end of this section 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 MAF Javadoc 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.

The following JavaScript code example shows how to use `removeContact`.

```
// Remove the contact from the device
contact.remove(onSuccess,onError);

function onSuccess()
{
  alert("Removal Success");
}

function onError(contactError)'
{
```

```
  alert("Error = " + contactError.code);
}
```

The following Java code example shows how to use `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 operation of the device takes longer than the timeout allowed, a timeout error is displayed.

15.12.8 How to Use the startLocationMonitor Method to Enable Geolocation

The `DeviceFeatures` data control includes the `startLocationMonitor` method to track device location at a point in time, or periodically. Listen for changes in the location of a device using parameters such as `enableHighAccuracy`, `updateInterval`, and `locationListener`.

The DeviceFeatures data control includes the `startLocationMonitor` method, which enables MAF applications to use the geolocation services of a device in order to obtain and track the location of the device. MAF enables you to display the interface of a device and optionally pre-populate the `startLocationMonitor` fields.

MAF exposes APIs that enable you to acquire the current position of a device, allowing you to retrieve the current location of the device for one instant in time or to subscribe to it on a periodic basis. The examples at the end of this section show how to use geolocation to subscribe to changes in the location of a device and how to obtain the location of a device. For information about the `startLocationMonitor` method, see the `DeviceDataControl` class in *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 the location of a device using the DeviceFeatures data control:

In JDeveloper, drag the `startLocationMonitor` operation from the DeviceFeatures data control in the Data Controls panel 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 s `startLocationMonitor` method of the `DeviceDataControl` class 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 position of the device 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 location of the device 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 position of the device using JavaScript, refer to the Cordova documentation (`http://cordova.apache.org/`).

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 *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 the current location of a device (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());
```

15.12.9 How to Use the displayFile Method to Enable Displaying Files

The `DeviceFeatures` data control provides the `displayFile` method to display files that are local to the device, such as PDFs, image files, or Microsoft Office documents, using either the platform functionality on iOS devices, or using third-party applications on Android devices. The `displayFile` operation is customizable.

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 in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.maf/Samples`

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 15-7 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/`).	MAF 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.

Table 15-7 (Cont.) Supported File Types

iOS	Android
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 JDeveloper, drag the **displayFile** operation from the DeviceFeatures data control in the Data Controls panel 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 following example shows you how to view files using `the displayFile` method. For information about the `displayFile` method, see the `DeviceDataControl` class in the MAF Javadoc).

```
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());
        }
```

15.12.10 How to Use the addLocalNotification and cancelLocalNotification Methods to Manage Local Notifications

The `DeviceFeatures` data control provides the `addLocalNotification` and `cancelLocalNotification` methods to schedule or cancel local notifications. Use

the procedure to customize an `addLocalNotification` or `cancelLocalNotification` operation.

The DeviceFeatures data control includes the `addLocalNotification` and `cancelLocalNotification` methods, which enable MAF applications to leverage the interface of a device for managing notifications so users can schedule or cancel local notifications.

To customize an addLocalNotification or cancelLocalNotification operation using the DeviceFeatures data control:

1. In JDeveloper, drag the **addLocalNotification** or cancelLocalNotification operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as a **Button**, **Link**, **List Item**, or **Parameter Form**.

   Button, Link, or List Item: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `addLocalNotification` or `cancelLocalNotification` operation. For more information on this dialog, see the online help for Oracle JDeveloper. At runtime, a button, link, or list item will be displayed on the page, which will use the entered values to perform the `addLocalNotification` or `cancelLocalNotification` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. For more information on this dialog, see the online help for Oracle JDeveloper. 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 the `addLocalNotification` or `cancelLocalNotification` operation when pressed.

Figure 15-17 shows the Edit Action Binding dialog, which you use to configure the parameters of the selected operation. In this example, the notificationID of the `cancelLocalNotification` operation is bound to the result of the `addLocalNotification` operation.

Figure 15-17 Setting Bindings for Scheduling Local Notifications

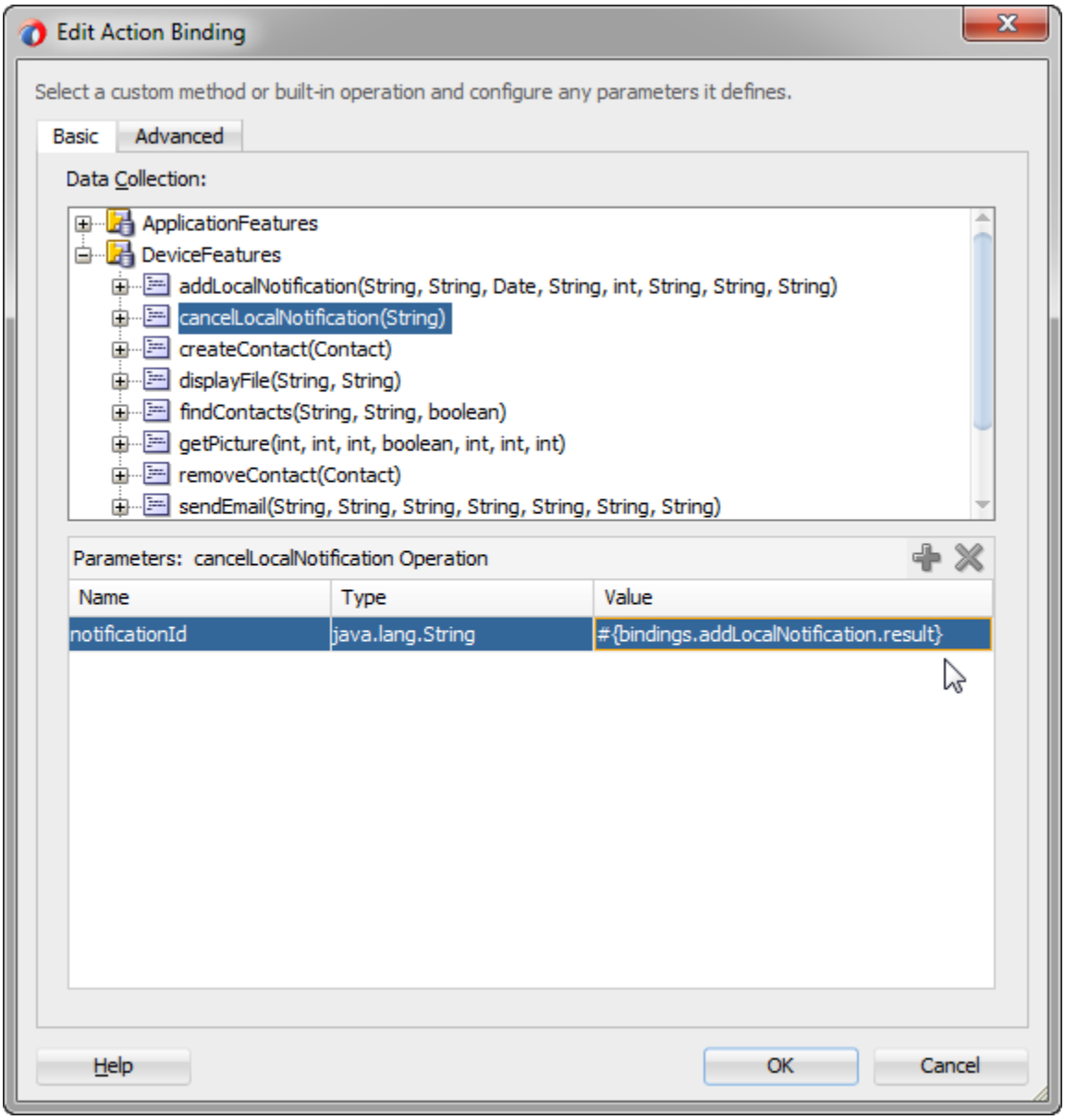
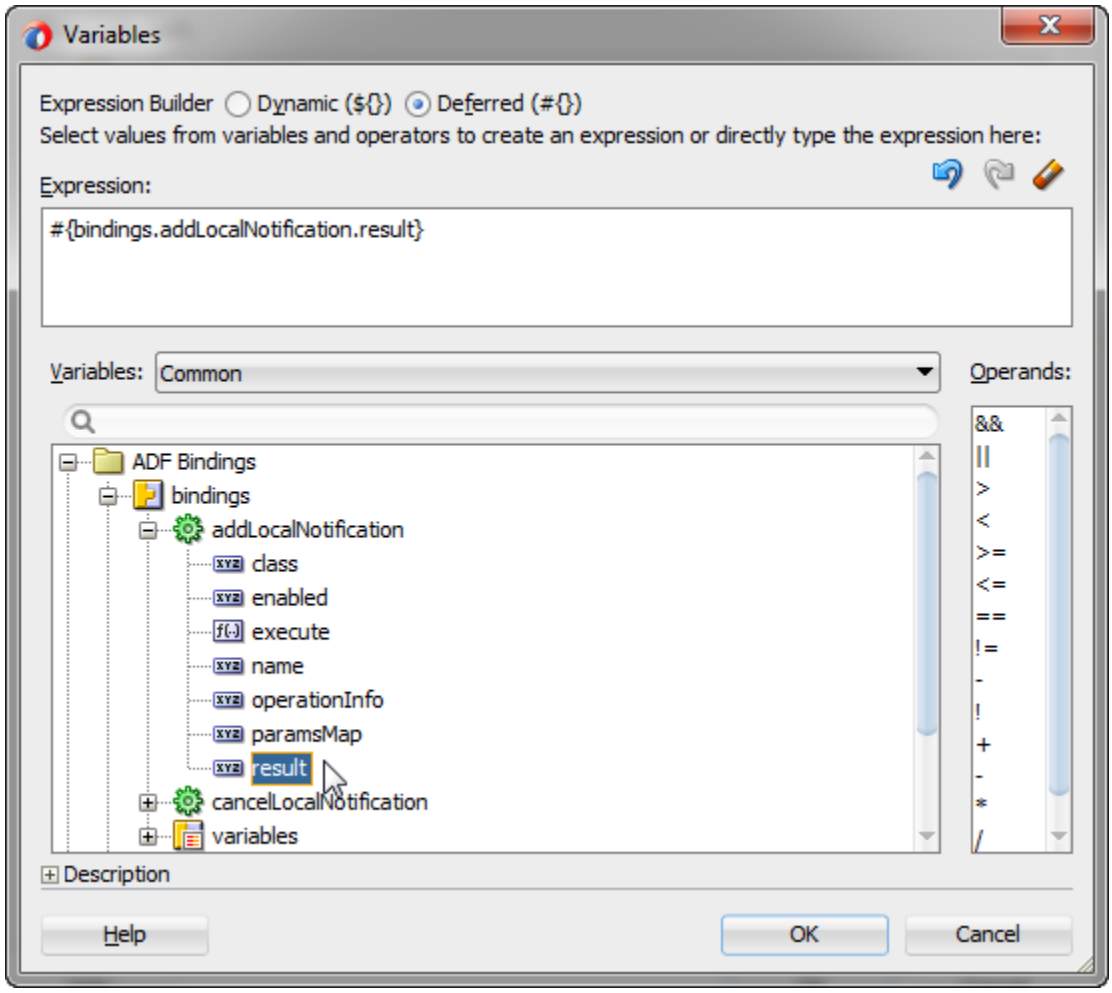


Figure 15-18 shows how you can use the expression builder to bind the result of the `addLocalNotification` operation to the `cancelLocalNotification` operation.

Figure 15-18 Binding cancelLocalNotification to the result of addLocalNotification



For information about the `addLocalNotification` and `cancelLocalNotification` methods, see the `DeviceDataControl` class in the MAF Javadoc). See Managing Local Notifications. Also see Introduction to Notifications.

15.12.11 What You May Need to Know About Device Properties

MAF includes properties that are accessed from Java, JavaScript, and EL to support dynamic behavior in applications. Review the list of properties to know about querying them, the expected return values, and property changes during the application lifecycle.

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 screen orientation of the device, 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 15-8 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 lifecycle of the application. The example at the end of this section shows how you can access these properties using JavaScript.

Note:

The timeout value for device properties is set to 1 minute. If the operation of the device takes longer than the timeout allowed, a timeout error is displayed.

Table 15-8 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()

Table 15-8 (Cont.) Device Properties and Corresponding EL Expressions

Property	Static/ Dynamic	EL Expression	Sample Value	Java API
hardware.hasLocalStorage	Static	#{deviceScope.hardware.hasLocalStorage}	"true", "false"	DeviceManager.hasLocalStorage()
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`.

The following example shows 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>
```

Note:

You can declaratively bind a JavaScript function to the `showpagecomplete` event by adding an `amx:clientListener` tag as a direct child of `<amx:view>`, as in the following example:

```
<amx:clientListener type="showpagecomplete"
method="myShowPageCompleteHandler"/>
```

For more information about the Client Listener (`clientListener`) component, see How to Use the Client Listener.

15.13 Validating Attributes

In MAF, validation rules are set on binding attributes, and validation occurs in the data control layer. You can define both validators for attributes exposed by the data controls, and validation message for 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. See Validating Input and How to Add Validation Rules.

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 15-9 lists supported validation combinations for the length validator.

Table 15-9 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 15-10 and Table 15-11 list supported validation combinations for the range validator.

Table 15-10 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 15-11 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 15-12 lists supported validation combinations for the list validator.

Table 15-12 List Validation

Compare type	String
`In`	Supported
`Not In`	Supported

Table 15-13 and Table 15-14 lists supported validation combinations for the compare validator.

Table 15-13 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
`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

Table 15-13 (Cont.) Compare Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short	String
`Greater Than Equal To`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported

Table 15-14 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

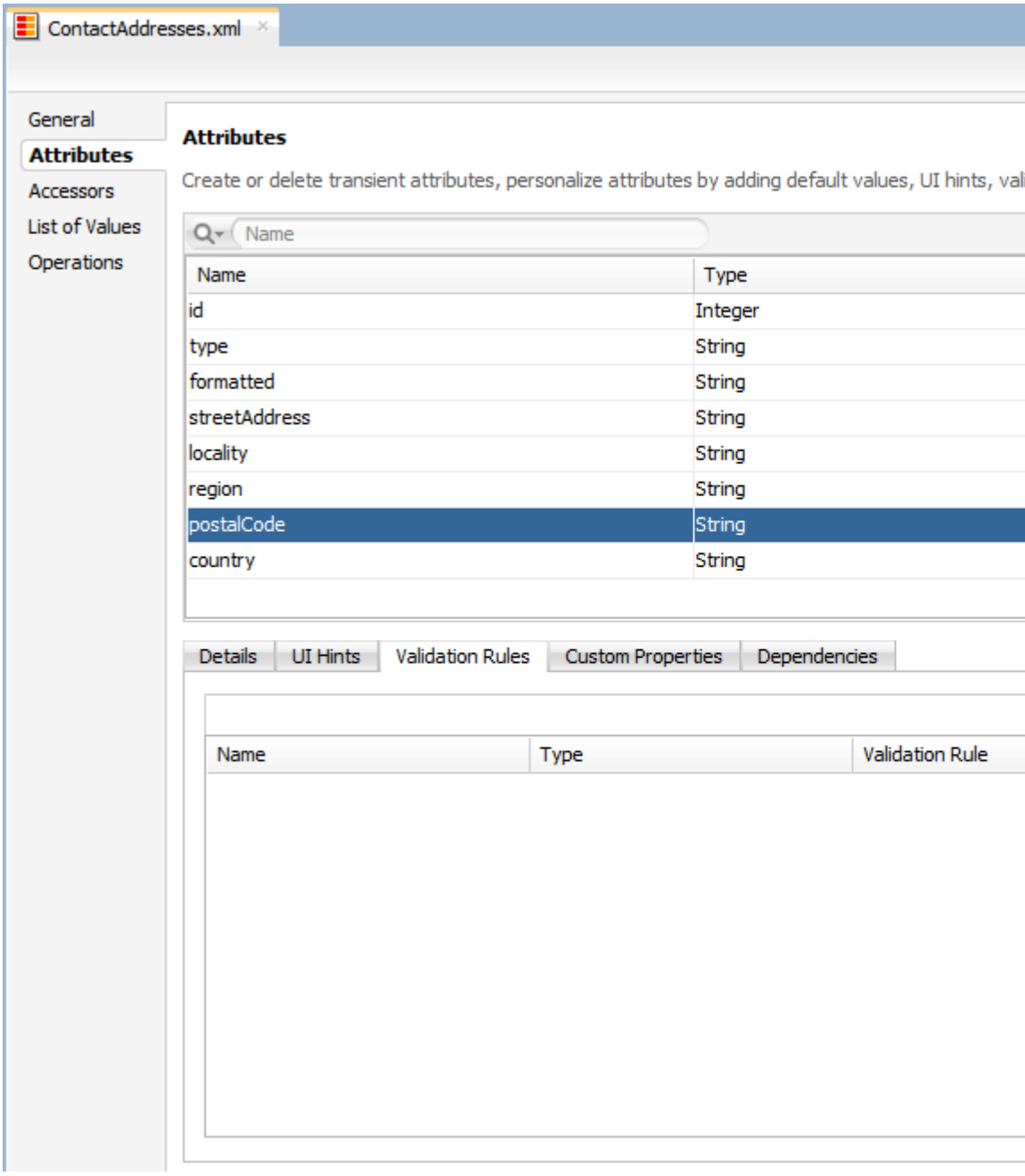
15.13.1 How to Add Validation Rules

MAF users can define validation rules. Use the procedure to add a validation rule to a data controls object using the Overview Editor for Data Control Structure Files - Attributes Page.

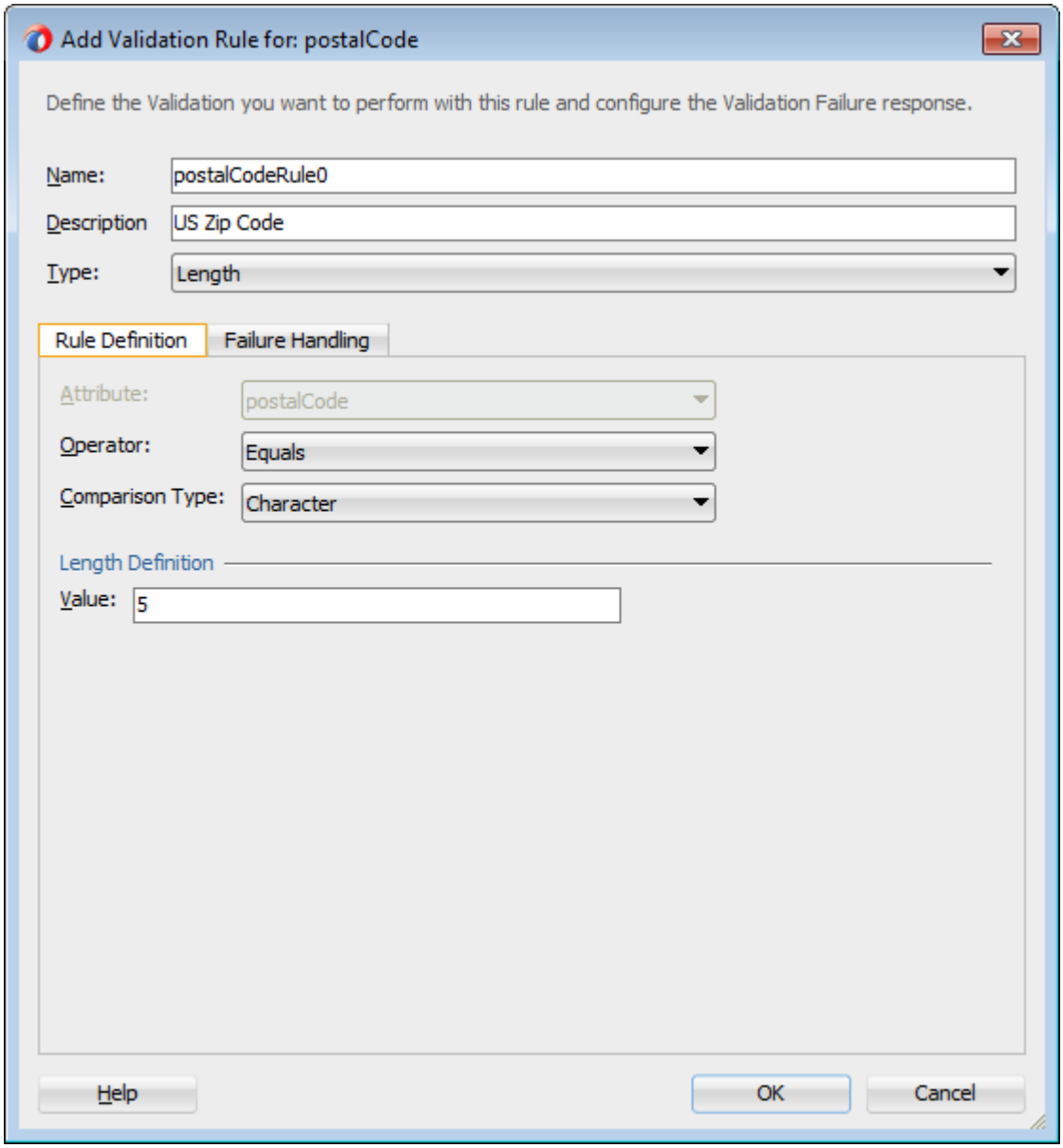
You can define validation rules for a variety of use cases. To add a declarative validation rule to an entity object, use the Overview Editor for Data Control Structure Files - Attributes Page.

To add a validation rule:

1. From the Data Controls panel, right-click on a data controls object and select **Edit Definition**.
2. In the Overview Editor for Data Control Structure Files, select the **Attributes** page.



3. Select the **Validation Rules** tab in the lower part of the page and then click **Add**. In the resulting **Add Validation Rule** dialog, define the validation rule and the failure handling.



15.13.2 What You May Need to Know About the Validator Metadata

The data control structure metadata XML files contain validator metadata.

The validator metadata is placed into the data control structure metadata XML files at design time. The following example shows a sample length validator.

```
<?xml version="1.0" encoding="windows-1252" ?>
<!DOCTYPE PDefViewObject SYSTEM "jbo_03_01.dtd">
<PDefViewObject
   xmlns="http://xmlns.oracle.com/bc4j"
   Name="Product"
   Version="12.1.1.61.36"
   xmlns:validation="http://xmlns.oracle.com/adfm/validation">
   <DesignTime>
      <Attr Name="_DCName" Value="DataControls.ProductListBean"/>
      <Attr Name="_SDName" Value="mobile.Product"/>
   </DesignTime>
```

```
   <PDefAttribute
      Name="name">
      <validation:LengthValidationBean
         Name="nameRule0"
         OnAttribute="name"
         CompareType="GREATERTHAN"
         DataType="BYTE"
         CompareLength="5"
         Inverse="false"/>
   </PDefAttribute>
</PDefViewObject>
```

15.14 Using Background Threads

Data model values can be updated with background Java threads. Update model objects from a background thread by using the `MafExecutorService` API.

A background thread may be useful when fetching data (say from a remote server) or computing values with a complex algorithm. You can also use background threads to fetch or compute data values, but you should not use them to update the application's data model objects directly, because this could result in conflicts with the application's user interface threads.

To update model objects from a background thread, use the `MafExecutorService` API to submit a Java `Runnable` that will perform the model updates. First, obtain the new or updated model values (fetched or computed) and then submit a `Runnable` to update the values in the application's data model objects, as shown in the following example.

```
//  First, fetch/compute new data values.
fetchUpdatedValues();

//  Next, use a Runnable to update the model objects.
MafExecutorService.execute(new Runnable()
{
  public void run()
  {
    doModelUpdates();
    AdfmfJavaUtilities.flushDataChangeEvent();
  }
});
```

Note: To ensure the application does not become unresponsive, the submitted task must be of short duration. Feature locks may be acquired before executing the task, which will not be released until the task completes.

For more information on the `oracle.adfmf.framework.api.MafExecutorService.execute` class, see the MAF Javadoc.

15.15 Working with Data Change Events

The user interface must be updated to reflect underlying data changes. Selecting the **Notify listeners when property changes** option sends notifications for property changes; using the `ProviderChangeSupport` class sends notifications relating to collection elements; and calling the `AdfmfJavaUtilities.flushDataChangeEvent` method forces data changes on AMX pages to the client.

To simplify data change events, JDeveloper uses the property change listener pattern. In most cases you can use JDeveloper to generate the necessary code to source notifications from the property accessors of the beans by selecting the **Notify listeners when property changes** checkbox in the Generate Accessors dialog (see 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:

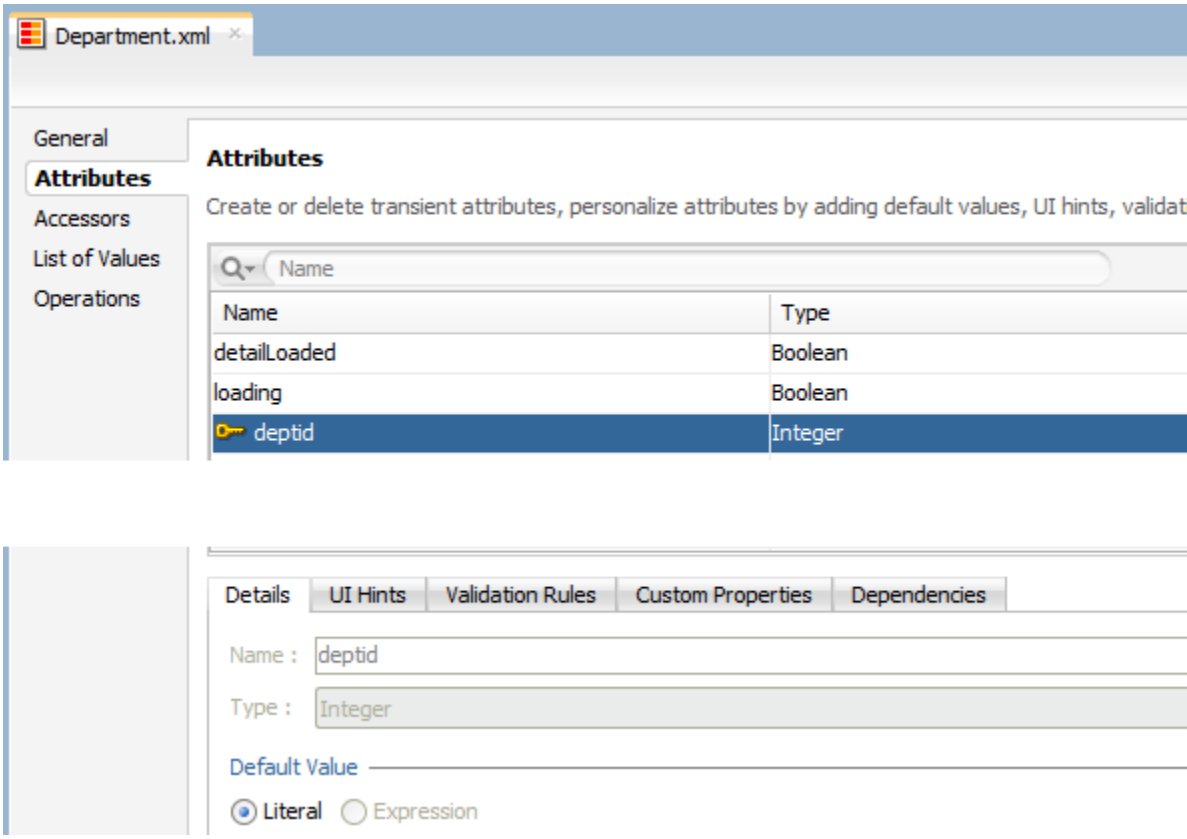
If you are manually adding a `PropertyChangeSupport` object to a class, you must also include the `addPropertyChangeListener` and `removePropertyChangeListener` methods (using these explicit method names).

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. Provider change events are like property change events but apply to the entire provider instead of just an individual property.

Note:

The `ProviderChangeSupport` object is *not* generated automatically—you must manually add it to your class—along with the `addProviderChangeListener`, `removeProviderChangeListener`, and `getKey()` methods (using these explicit method names). The `getKey()` method must return a string that produces a unique value for the provider. As an alternative to adding the `getKey()` method to your class, designate an attribute in the data control as the key attribute in the data control structure file using the overview editor shown in Figure 15-19.

Figure 15-19 Selecting a Key Attribute in the Data Control Structure File of a Data Control



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
- `fireProviderChange`—when a single element is changed in the collection (necessary to prevent the whole list from refreshing)
- `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 Java bean 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 following example 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 as well as the `ProviderChangeEvent` and `ProviderChangeSupport` classes, see the MAF Javadoc.

```
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;

    /* --- JDeveloper 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 JDeveloper 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 following example shows how you can use the `flushDataChangeEvent` method 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 JDeveloper 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;

        /* --- JDeveloper 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;

    /* --- JDeveloper 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. For more information about this and other sample applications, see MAF Sample Applications.

16

Configuring End Points Used in MAF Applications

This chapter describes how to use the Configuration Service to configure end points that a MAF application can use.

This chapter includes the following sections:

- Introduction to Configuring End Points in MAF Applications
- Defining the Configuration Service End Point
- Creating the User Interface for the Configuration Service
- About the URL Construction
- Setting Up the Configuration Service on the Server

16.1 Introduction to Configuring End Points in MAF Applications

MAF provides the Configuration Service to configure end points used by web services, login utilities, and other components of MAF applications.

The Configuration Service is a tool that allows you to configure end points used by web services, login utilities, and other parts of MAF applications.

Note:

MAF applications on the Universal Windows Platform do not support the use of the Configuration Service.

16.2 Defining the Configuration Service End Point

Defining the Configuration Service end point URL and a new connection entry in the `connections.xml` file constitute the definition of a Configuration Service end point.

The end point 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 URL and its `name` set to an arbitrary value which will eventually be referenced in a Java bean code.

The following 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"/>
          <injectCookiesToRestHttpHeader value="true"/>
          <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 user. To address this use case, create a user interface to retrieve the value of the end point URL from the 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 16-1 shows.

Example 16-1 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);
```

16.3 Creating the User Interface for the Configuration Service

The Configuration Service user interface is created using either a new or available application feature. The feature is used to activate methods of the APIs of the `oracle.adfmf.config.client.ConfigurationService` class that is made available in a Java bean.

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 configuration server of the previous release 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 to associate the configuration server connection and the end point URL:

```
_configservice.setDeliveryMechanismConfiguration("connectionName",
                                        <ConfigService_ConnectionName>);
```

- `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 hard-coded value or obtained through the `Application.getApplicationVersion` API:

  ```
  _configservice.stageAndActivateVersion("1.0");

  _configservice.stageAndActivateVersion(Application.getApplicationVersion);
  ```

- `addProgressListener` method registers an update progress listener 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. The ConfigServiceDemo application is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

For more information about the `oracle.adfmf.config.client.ConfigurationService` class, see *Java API Reference for Oracle Mobile Application Framework*.

16.4 About the URL Construction

The endpoint URL that the user provided or that was specified in the`connections.xml file` is used by the Configuration Service to construct the URL to download the `connections.xml` file.

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 download the `connections.xml` file:

```
http://my.server.com:port/SomeLocation/com.mycompany.appname/connections.xml
```

16.5 Setting Up the Configuration Service on the Server

The Configuration Service accepts HTTP `GET` requests and returns the `connections.xml` file.

The Configuration Service can be implemented as a service that accepts HTTP `GET` requests and returns the `connections.xml` file.

The URL used by the Configuration Service client is in the following format:

`url configured in adf-config.xml/application bundle id/``connections.xml`

The Configuration Service end point may be secured using basic authentication (`BASIC_AUTH`) over HTTP and HTTPS.

17

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:

- Introduction to Using Web Services in MAF Applications
- Creating a Rest Service Adapter to Access Web Services
- Accessing Secure Web Services
- Configuring the Browser Proxy Information

17.1 Introduction to Using Web Services in a MAF Application

MAF supports the consumption of REST web services with JSON objects in MAF applications. This type of web service is recommended by MAF over alternative web services as the smaller payloads that it generates 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.

> **Note:** As an alternative to writing a `RESTServiceAdapter`, use the design-time support provided by MAF to generate the client data model that accesses REST web services. See Creating the Client Data Model in a MAF Application.

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 MAF Sample Applications.

17.2 Creating a Rest Service Adapter to Access Web Services

Use `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 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.

17.2.1 Accessing Input and Output Streams

You can use the `RestServiceAdapter` to obtain and customize the `java.net.HttpURLConnection`, as well as access and interact with the input and output streams of the connection which allows you to read data from the `HttpURLConnection` and write to it for further upload to the server.

The following example initializes and returns an `HttpURLConnection` 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.

```
....
// Get the connection
HttpURLConnection httpURLConnection =
   restServiceAdapter.getHttpURLConnection(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);
...
```

17.2.2 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.

```
RestServiceAdapterFactory 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) {
               }
            }
         }
      }
```

17.3 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.

Table 17-1 lists the predefined security policies that you can associate with connections to REST web services. How to Enable Access to Web Services describes how you associate the connection with the predefined security policy.

Table 17-1 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.

Table 17-1 (Cont.) Security Policies Supported for REST-Based Web Services

Authentication Type	REST Policy	Description
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.
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 in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

17.3.1 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 Accessing Secure Web Services. You create the login server connection using the Create MAF Login Connection dialog in the `maf-application.xml` overview editor. For details about creating the login server connection, see How to Create a MAF Login Connection.

To associate a security policy with a web service:

1. In the Applications window, expand the **Descriptors** node and then **ADF META-INF**, and double-click **maf-application.xml**. Then, in the overview editor for the `maf-application.xml` file, expand the **Security - Web Services Security Policies** section and select the web service connection that you created in the **Name** field.

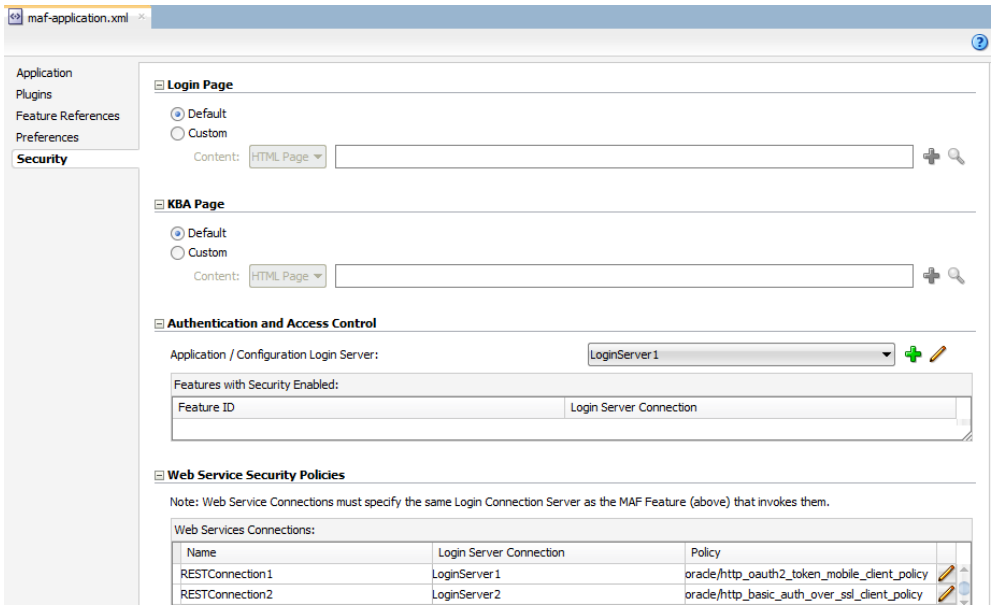
2. In the **Login Server Connection** field, select the login server connection that you defined.

3. In the **Policy** field, double-click the Edit Policy icon button and in the Edit Data Control Policies dialog, select the policy that you want to associate with the service for the current web service connection and click **OK**.

   Figure 17-1 shows the policy associated with RESTConnection1 and RESTConnection2.

   For a list of the security policies that you may select in the Edit Data Control dialog and associate with a REST-based web service, see Accessing Secure Web Services.

Figure 17-1 Associating a Security Policy with a Web Service Connection

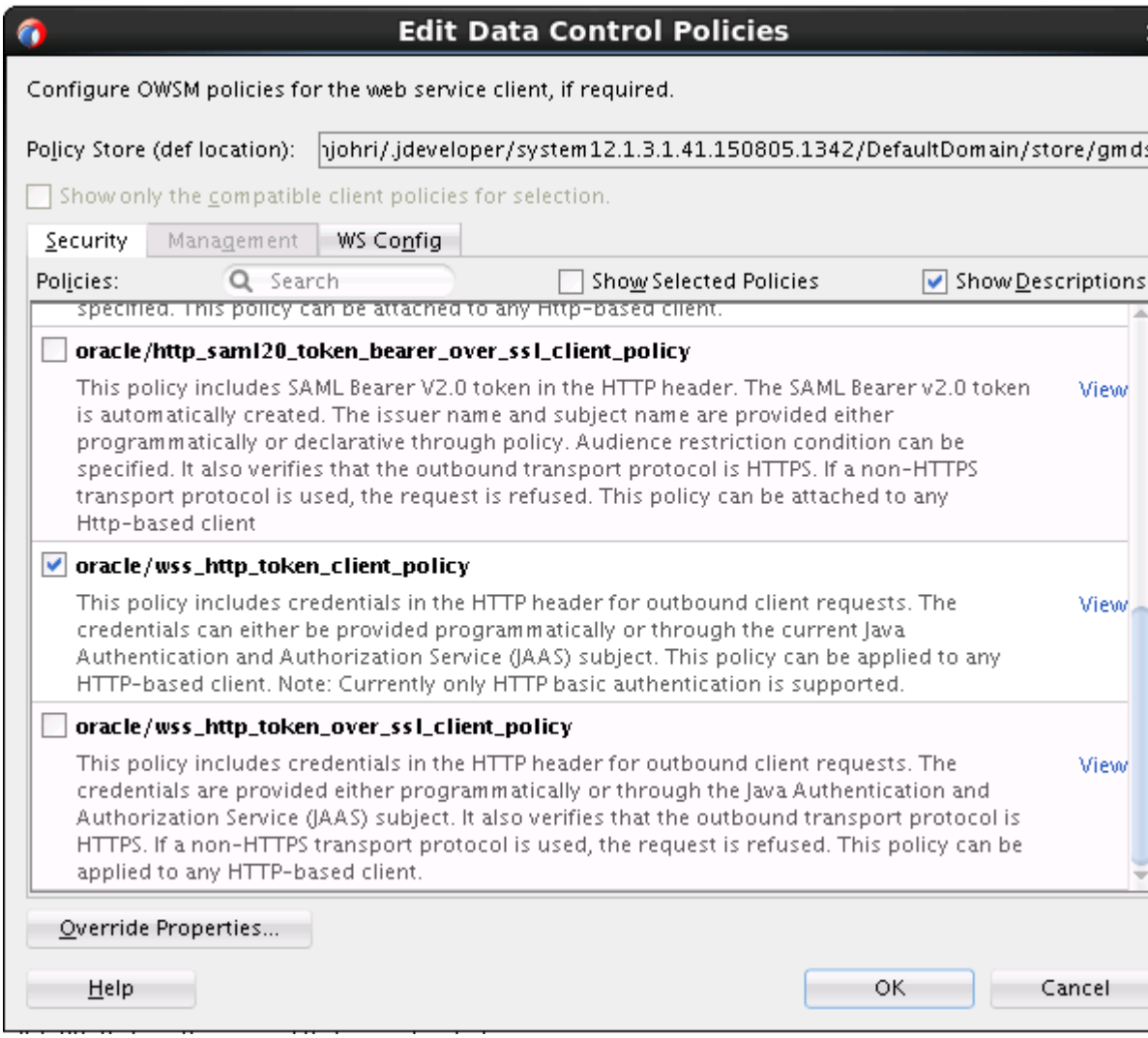


17.3.2 What Happens When You Enable Access to Web Services

JDeveloper stores the web service policy definitions in the `wsm-assembly.xml` file in the `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 Policies dialog shown in Figure 17-2. Click **Override Properties** to invoke a dialog where you can specify alternative values for properties that the selected policy permits you to override.

Figure 17-2 Editing Web Service Data Control Policies



17.3.3 What You May Need to Know About Credential Injection

When a user authenticates to access secured web services, MAF dynamically injects the credentials into the web service request using the Oracle Web Services Manager Mobile Agent.

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 server URL of the authentication provider 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 `adfCredentialStoreKey` attribute value of the login server connection in the `adfCredentialStoreKey`

attribute of the web service connection reference in order to associate the login server to the web service security (see the following two examples).

Note:

Since JDeveloper does not provide an Overview editor for the `connections.xml` file, you can use the Properties window to update the `adfcredentialStoreKey` attribute of the `<Reference>` element with the name configured for the `adfCredentialStoreKey` attribute of the login server connection. Alternatively, you can add or update the attribute using the Source editor.

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 Using and Configuring Logging). 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 Configuring End Points Used in MAF Applications .

The `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.

17.3.4 What You May Need to Know About Cookie Injection

MAF executes a cookie injection for the login connection associated with the URL endpoint of the REST web service for cookie-based authorization.

Each time a MAF application requests a REST web service for cookie-based authorization, the MAF 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.

17.4 Configuring the Browser Proxy Information

Configure Java system properties to use an HTTP proxy server to call web services that reside outside a corporate firewall. The proxy can either be defined by means of a generic connection framework or the java.net API.

By default, MAF determines the proxy information using the system settings on the platform where you deploy the application. For example, if the proxy information is set using the Settings utility on an iOS-powered device, then JVM automatically absorbs it.

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, 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.

18

Using the Local Database in MAF AMX

This chapter describes how to use the local SQLite database within a MAF AMX application feature.

This chapter includes the following sections:

- Introduction to the Local SQLite Database Usage
- Using the Local SQLite Database

18.1 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 sample usage of the local SQLite database, see the MAF sample application called CRUDDemo located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer. The CRUDDemo sample application uses a custom SQLite database file that is packaged within this application. The database file contains a table with records which include information on employees. When the application is activated, it reads data from the table and displays a list of employees. The information about the employees can be subject to CRUD operations: employees can be created, reordered, updated, and deleted through the user interface. All the CRUD operations are updated in the SQLite database.

If you plan to use the SQLite database to provide offline access and synchronization with a REST data service in your MAF application, we recommend that you use the MAF client data model. It provides wizards to facilitate the retrieval of data from REST services, to select what data to persist in the SQLite database when your MAF application is offline, and enables support for offline transaction plus synchronization when the MAF application returns online. It also provides an API (`DBPersistenceManager`) which exposes a variety of methods to interact with the SQLite database. See Creating the Client Data Model in a MAF Application and

Accessing the SQLite Database Using the MAF Client Data Model DBPersistenceManager.

18.1.1 Differences Between SQLite and Other Relational Databases

SQLite is designed for use as an embedded database system, one that is 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 are 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`

18.1.1.1 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. Its coarse-grained locking mechanism does not allow SQLite to support multiple read-write connections to the same database instance.

Due to its coarse-grained locking mechanism, SQLite does not support multiple read-write connections to the same database instance. 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`.

18.1.1.2 SQL Support and Interpretation

Despite SQLite complying with the SQL92 standard, it has a few unsupported constructs.

Although SQLite complies with the SQL92 standard, there are a few unsupported constructs, including the following:

- `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`
- `GRANT`
- `REVOKE`

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.

18.1.1.3 Data Types

SQLite is dynamically typed and any value can be stored in any column, regardless of its declared type.

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. See "Datatypes In SQLite Version 3" available from the Documentation section of the SQLite website at http://www.sqlite.org/datatype3.html.

18.1.1.4 Foreign Keys

SQLite supports foreign keys. It parses and enforces foreign key constraints. See the *SQLite Foreign Key Support* available from the Documentation section of the SQLite site at http://www.sqlite.org/foreignkeys.html.

18.1.1.5 Database Transactions

SQLite is ACID-compliant and hence supports transactions but it neither supports nested transactions, nor does it allow a transaction to be rolled back until all open ResultSets have been closed. SQLite either permits multiple read-only connections or a single read-write connection to any given database.

Although SQLite is ACID-compliant and hence supports transactions, there are some fundamental differences between its transaction support and that of Oracle:

- 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.

See "Distinctive Features of SQLite" available from the Documentation section of the SQLite website at http://www.sqlite.org/different.html.

18.1.1.6 Authentication

SQLite neither supports role-based nor user-based authentication. It allows users access to all the data in a file, and you can use MAF encryption to secure the data.

SQLite does not support any form of role-based or user-based authentication. By default, anyone can access all the data in the file. However, MAF provides encryption routines that you can use to secure the data, and prevent access by users without a valid set of credentials. See How to Encrypt and Decrypt the Database.

18.2 Using the Local SQLite Database

MAF contains an encrypted SQLite 3.8.5 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

18.2.1 How to Connect to the Database

You can connect to the SQLite database using the `java.sql.Connection` object associated with the application. The SQLite JDBC URL must begin with the text `jdbc:sqlite`.

Connecting to the SQLite database differs from opening a connection to an Oracle database. 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:`.

The following example shows how to open a connection to an unencrypted database. Before obtaining the connection, load the JDBC driver.

```
public static Connection getConnection() throws Exception {
   if (conn == null) {
      try {
         // create a database connection
         String Dir = AdfmfJavaUtilities.getDirectoryPathRoot(
                              AdfmfJavaUtilities.ApplicationDirectory);
         String connStr = "jdbc:sqlite:" + Dir + "/portfolio.db";
         // Load the driver
         Class.forName("SQLite.JDBCDriver");
         conn = DriverManager.getConnection(connStr);
      }
      catch (SQLException e) {
         // If the error message is "out of memory", it probably
         // means that no database file is found
         System.err.println(e.getClass().getName() + ": " + e.getMessage() );
         e.printStackTrace();
      }
   }
   return conn;
}
```

The following 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".

18.2.2 How to Use SQL Script to Initialize the Database

When the application starts, an SQL script initializes the database. The script must be added as a resource to the ApplicationController project of the MAF application.

An SQL script is used to initialize the database when the application starts. The following example shows the SQL initialization script that demonstrates some of the supported SQL syntax (described in 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 the script as a resource to the ApplicationController project of your MAF application. Assume that a sample script has been saved as `initialize.sql` in the `META-INF` directory. The following example shows the code that you must 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:

To keep the example simple, the preceding example omits error handling.

Invoke the database initialization code (see the preceding example) from the `start` method of the `LifeCycleListenerImpl`, as the following example shows.

```
public void start() {
   try {
      initializeDatabaseFromScript();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
```

```
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
                e);
      }
   }
```

18.2.3 How to Initialize the Database on a Desktop

A database can be initialized on the iOS, Android, Windows, Linux, and Mac platforms using the same database file. In case of complexities, initialize the database on a desktop using third-party tools, and package the response file as a resource in the application.

To use the database, add the database as a resource to the ApplicationController project of your MAF application. Assume that a database has been saved as `sample.db` in the `META-INF` directory. The following example shows the code that you must add to copy the database from your application to the mobile device's file system to enable access to the database.

```
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:

To keep the example simple, the preceding example omits error handling.

Invoke the database initialization code (see the preceding example) from the `start` method of the `LifeCycleListenerImpl`, as the following example shows.

```
public void start() {
   try {
      initializeDatabase();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
                e);
   }
}
```

18.2.4 What You May Need to Know About Commit Handling

The SQLite database provides an auto-commit functionality that commits statements as they are read from the SQL script.

`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.

18.2.5 Limitations of MAF SQLite JDBC Driver

The `getByte` method of the ResultSet and the `execute` method of Statement from the `java.sql` package have limited or no support in MAF.

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 throws 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`).

18.2.6 How to Use the VACUUM Command

The size of the SQLite database remains constant even after records deletion leading to fragmentation and degraded performance. Use the `VACUUM` command to prevent performance degradation.

When records are deleted from an SQLite database, its size remains unchanged. This constant size leads to fragmentation and, ultimately, results in degraded performance. To prevent performance degradation, run the `VACUUM` command, periodically.

Note:

The `VACUUM` command uses long periods of time when run on large databases (approximately 0.5 seconds per megabyte on the Linux computer on which SQLite is developed). It can also use up to twice as much temporary disk space as the original file while it is running.

The `VACUUM` command must be run from a properly registered `LifeCycleListener` implementation (see Using Lifecycle Listeners in MAF Applications.).

18.2.7 How to Encrypt and Decrypt the Database

You not only can encrypt the SQLite database using APIs, but can also specify a password for the encryption. Use the procedures to encrypt the database with your own password, decrypt the database encrypted with your own password, encrypt the database using the MAF-generated password, and to decrypt the database and delete the MAF-generated password

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.

18.2.7.1 Encrypting the Database with Your Own Password

Use the procedure to encrypt the database with your own password.

To encrypt the database with your own password:

1. Establish the database connection (see How to Connect to the Database).
2. Use the following utility method to encrypt the database with a new key:

```
AdfmfJavaUtilities.encryptDatabase(connection, "newPassword");
```

18.2.7.2 Permanently Decrypting the Database Encrypted with Your Own Password

Use the procedure to decrypt the database encrypted with your own password.

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.

18.2.7.3 Encrypting the Database with a Password Generated by MAF

Use the procedure to encrypt the database using the MAF-generated password.

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 How to Connect to the Database).

4. Encrypt the database as follows:

   ```
   AdfmfJavaUtilities.encryptDatabase(connection, new String(password));
   ```

18.2.7.4 Decrypting the Database Encrypted with a Password Generated by MAF

Use the procedure to decrypt the database and delete the MAF-generated 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

Customizing MAF AMX Application Feature Artifacts

This chapter describes how to customize existing MAF AMX pages, task flows, and page definition files.

This chapter includes the following sections:

- Introduction to Customizing MAF AMX Pages and Artifacts
- Customizing MAF AMX Pages and Artifacts

19.1 Introduction to Customizing MAF AMX Pages and Artifacts

JDeveloper and Oracle Metadata Service provide standard customization mechanisms to customize existing MAF AMX application feature artifacts and metadata files. Customization are applied at application deployment, and can be seen at runtime.

You can use the standard customization mechanism provided by JDeveloper and Oracle Metadata Service (MDS) to customize your existing MAF AMX application feature artifacts and metadata files, including the following:

- MAF AMX files (`.amx`)
- Task flow files, such as `ViewController-task-flow.xml`
- Page definition files (`<page name>.PageDef.xml`)
- Data control XML file—a package file that contains a data control structure file (that is, a package file named for a data control and prefixed with `persdef.`).

The customization changes that you make at design time are applied to your files during deployment and become visible at runtime. MAF AMX supports the static seeded customization, where the final version for a specific customization context is seeded during deployment and work statically at runtime for that customization context. For each customization context you have to deploy a separate MAF application.

Note:

MAF AMX does not support the user customization that both creates and applies customization at runtime.

See Customizing MAF Application Artifacts with MDS .

19.2 Customizing MAF AMX Pages and Artifacts

ID-prefix tokens, added when customization layers are configured, ensure the accurate application of customizations of AMX pages and artifacts.

You customize your MAF AMX pages and artifacts by following steps outlined in Introduction to Applying MDS Customizations to MAF Files.

When configuring customization layers, to help ensure the uniqueness of the identifier so that customizations are applied accurately, you can add an `id-prefix` token. When you add a new element, such as, for example, a `commandButton` to a MAF AMX page during customization, JDeveloper adds the `id-prefix` of the layer and layer value to the autogenerated identifier for the element to create an `id` for the newly added element in the customization metadata file. As shown in the following example, the `site` layer has an `id-prefix` of " `s` " and the `headquarters` layer value has an `id-prefix` of " `hq` ".

```
<cust-layers  xmlns="http://xmlns.oracle.com/mds/dt">
   <cust-layer name="industry" id-prefix="i">
      <cust-layer-value value="financial"
                        display-name="Financial"
                        id-prefix="f"/>
      <cust-layer-value value="healthcare"
                        display-name="Healthcare"
                        id-prefix="h"/>
   </cust-layer>
   <cust-layer name="site" id-prefix="s">
      <cust-layer-value value="headquarters"
                        display-name="HQ"
                        id-prefix="hq"/>
      <cust-layer-value value="remoteoffices"
                        display-name="Remote"
                        id-prefix="rm"/>
   </cust-layer>
</cust-layers>
```

When you select `site/headquarters` as the tip layer and add a MAF AMX Button component to a page, the `commandButton` element will have an `id` of " `shqcb1` " in the metadata customization file.

When the customization process is complete, JDeveloper creates a metadata file for the customizations and a subpackage for storing them. The metadata file contains the customizations for the customized object, which are applied over the base metadata at runtime. JDeveloper gives the new metadata file the same name as the base file for the object, but includes an additional `.xml` extension, as Figure 19-1, Figure 19-2, Figure 19-3, and Figure 19-4 show.

Figure 19-1 Customization File for MAF AMX Page

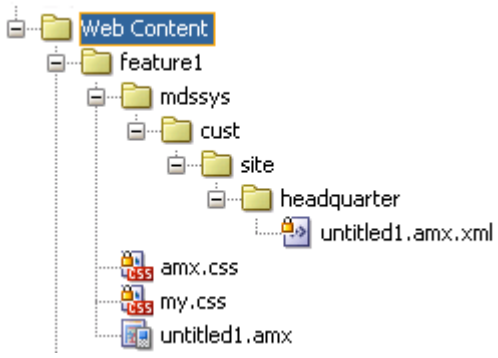


Figure 19-2 Customization File for Task Flow

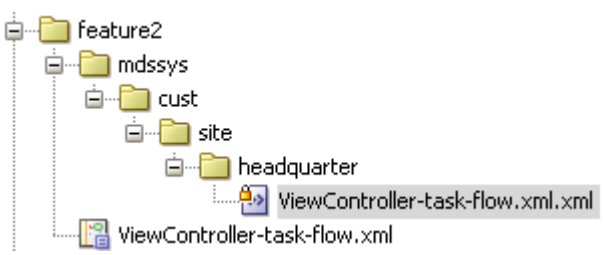
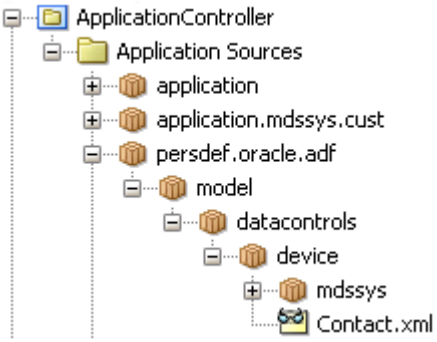


Figure 19-3 Customization File for Page Definition

Figure 19-4 Customization File for Data Control XML File



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:

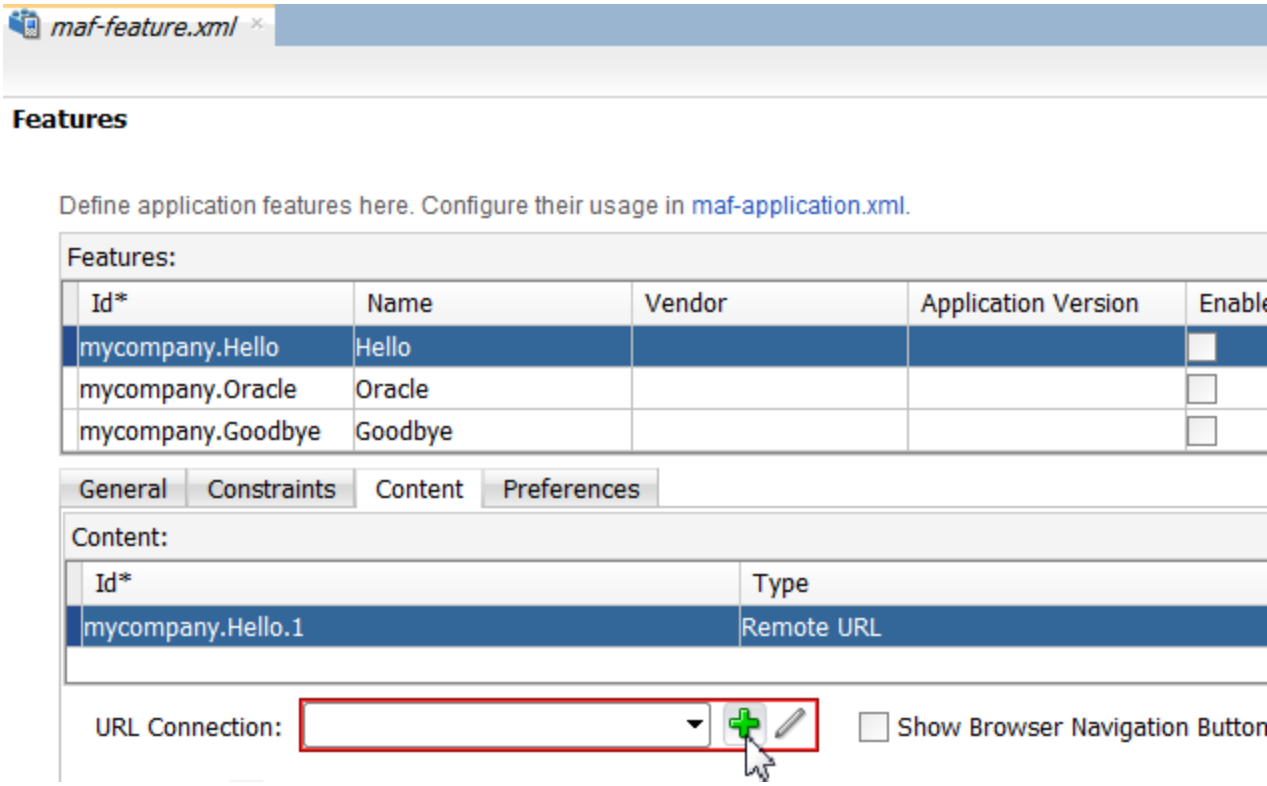
- Introduction to Remote URL Applications
- Enabling Remote Applications Access Container Services
- Whitelisting Remote URLs in Your MAF Application
- Enabling the Browser Navigation Bar on Remote URL Pages

20.1 Introduction to Remote URL Applications

You can access a server-hosted application by creating a URL connection in an application feature with a Remote URL content type in the overview editor of the `maf-feature.xml` file.

Such server-hosted applications as shown below differ from the 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 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.

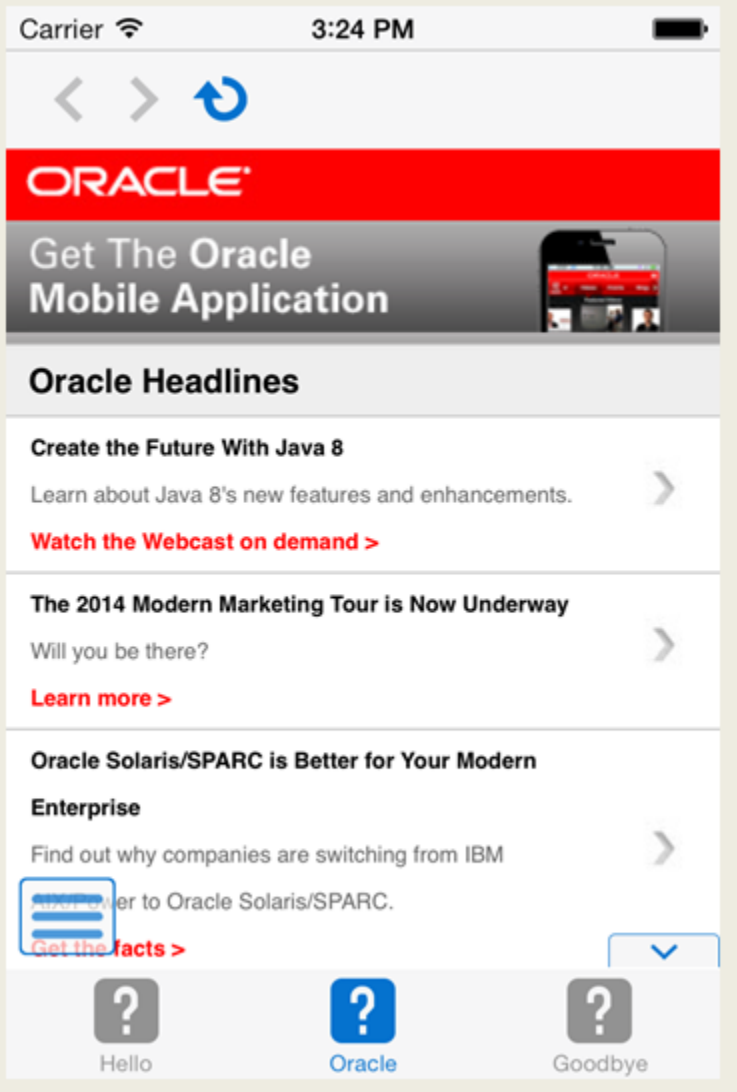
Tip:

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 How to Whitelist Remote URLs on the Android Platform.
- iOS platform, see How to Whitelist Remote URLs on the iOS Platform.
- Universal Windows Platform, see 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 Registering Additional Plugins in Your MAF Application.

For more information about developing a plugin for use in a MAF application, see 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 Introduction to Using Plugins in MAF Applications. Once you complete development of your plugin, register it in your MAF application, as described in 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 Registering Additional Plugins in Your MAF Application.

For more information about developing a plugin for use in a MAF application, see Introduction to Using Plugins in MAF Applications.

20.4 Enabling the Browser Navigation Bar on Remote URL Pages

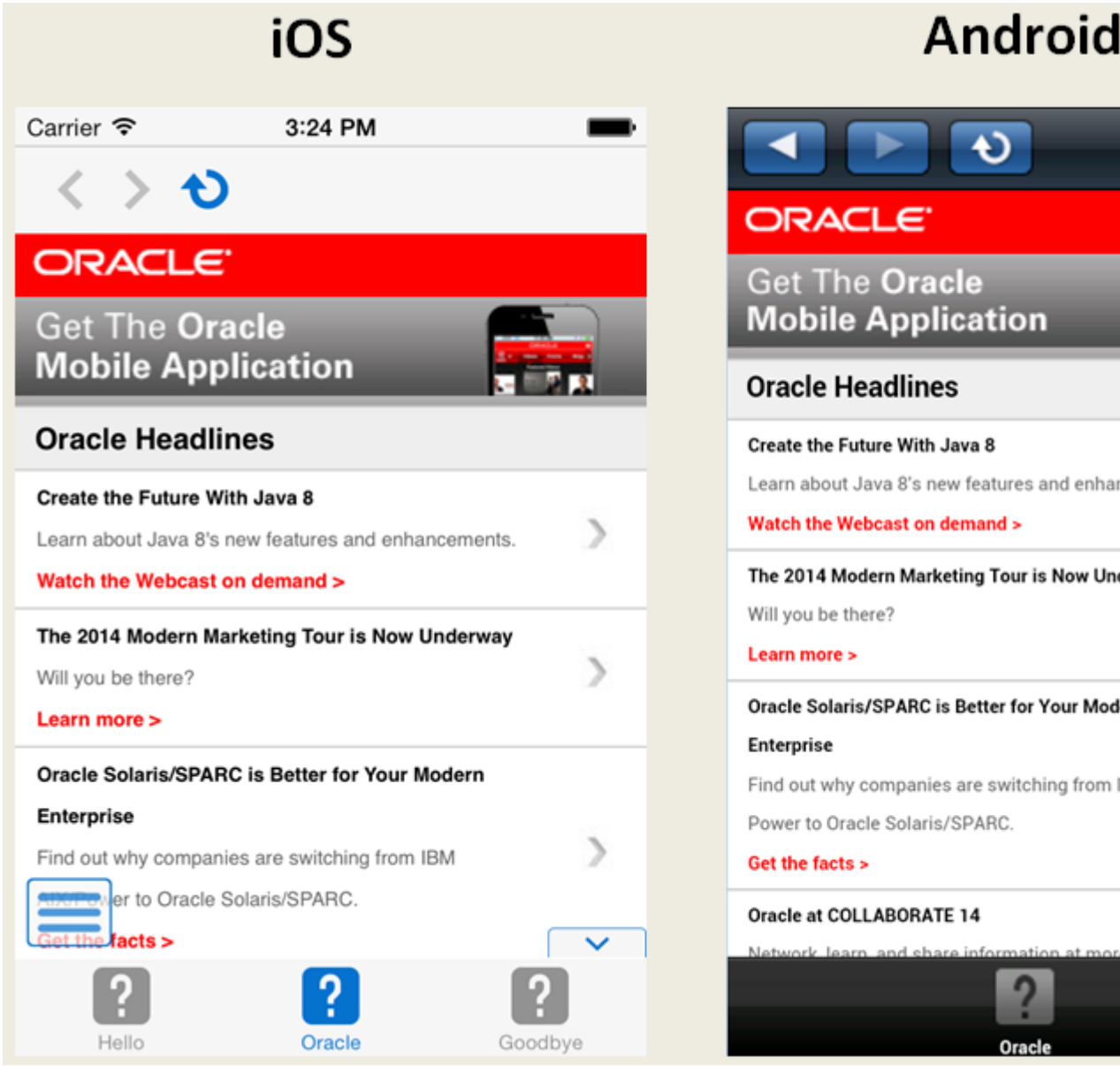
MAF enables you to add a navigation bar with buttons for back, forward, and refresh actions for application features implemented as remotely served web content that open within the MAF web view.

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 overview editor for the `maf-feature.xml` file.

Before you begin:

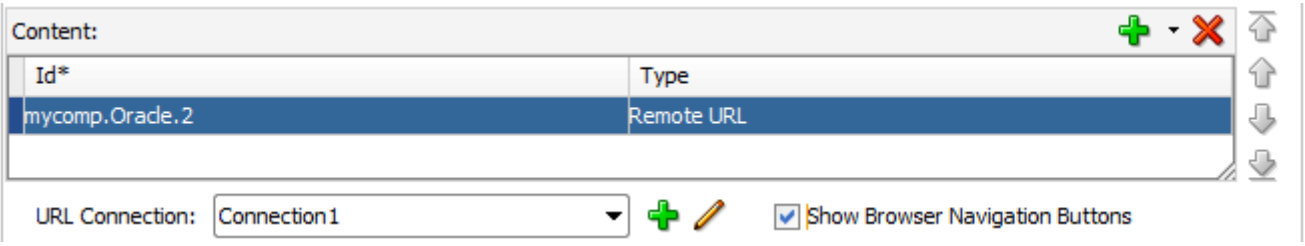
Designate an application feature's content be delivered from a remotely hosted application by first selecting **Remote URL** and then by creating the connection to the host server, as described in Defining the Application Feature Content as Remote URL or Local HTML.

To enable a navigation bar:

1. Select the Remote URL application feature listed in the Features table in the `maf-feature.xml` file.

2. Click **Content**.
3. Select **Show Browser Navigation Buttons**, as shown in Figure 20-4.

Figure 20-4 Selecting Navigation Options



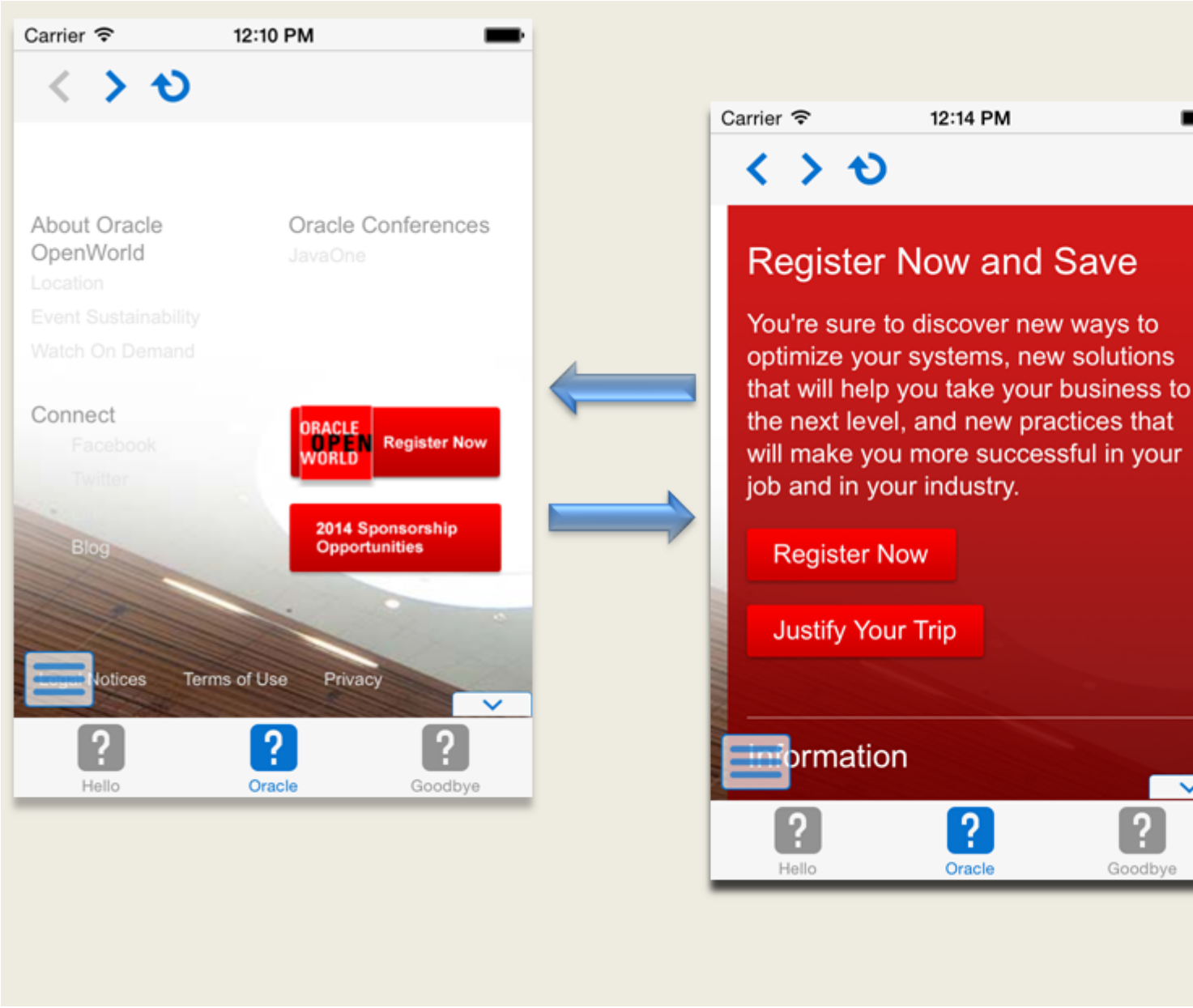
20.4.2 What Happens When You Enable the Browser Navigation Buttons for a Remote URL Application Feature

JDeveloper updates the `adfmf:remoteURL` element with an attribute called `showNavButtons`, which is set to `true`, as shown in the following example.

```
<?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:

- Creating User Preference Pages for a Mobile Application
- Creating User Preference Pages for Application Features
- Using EL Expressions to Retrieve Stored Values for User Preference Pages
- 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.

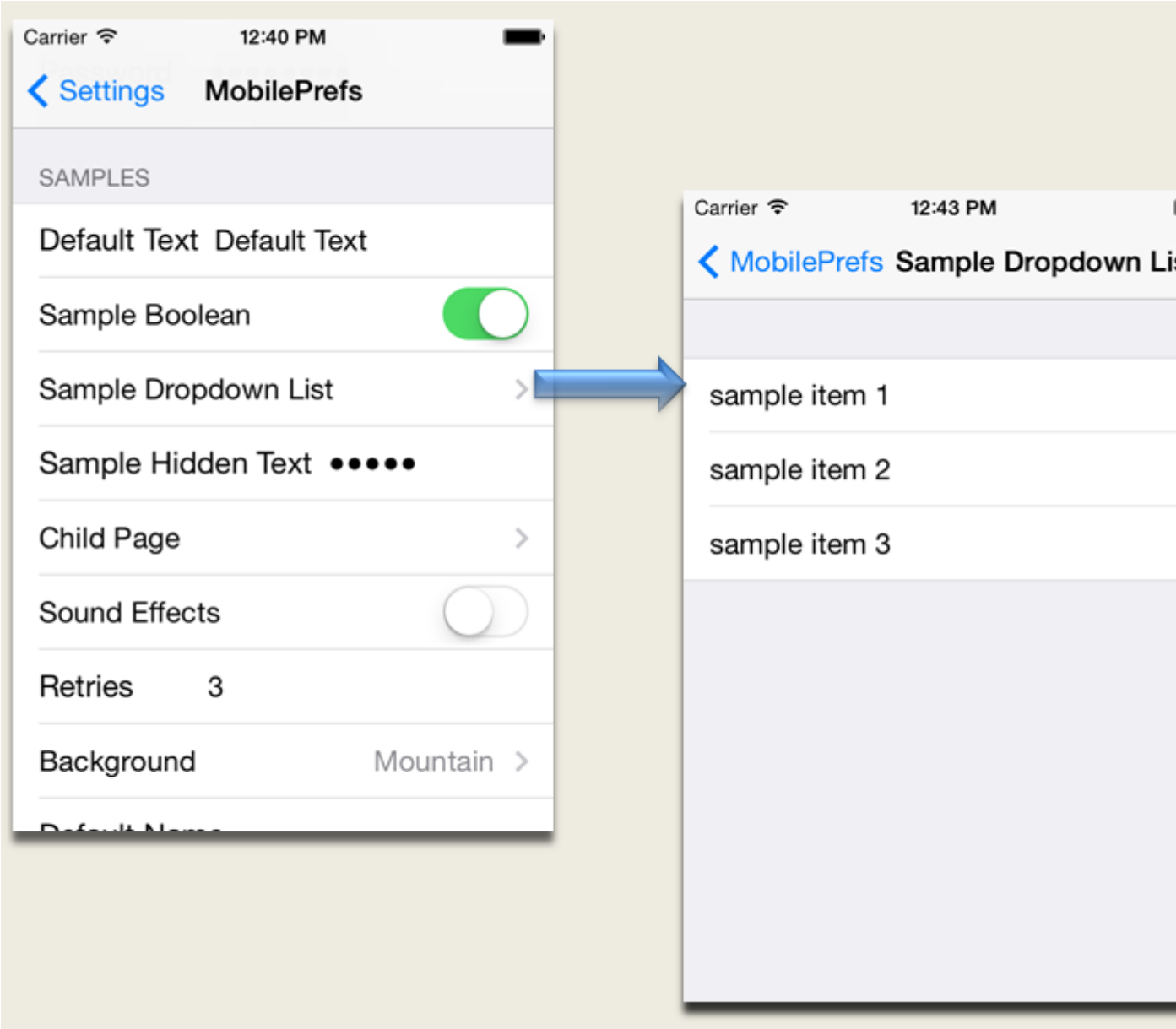
As shown in the following example, the child element of `<adfmf:preferences>` called `<adfmf:preferenceGroup>` and its child elements define the user preferences by creating pages that present options in various forms, such as text strings, drop-down menus, or in this case, as a child page that can present 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 a child user preference 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.

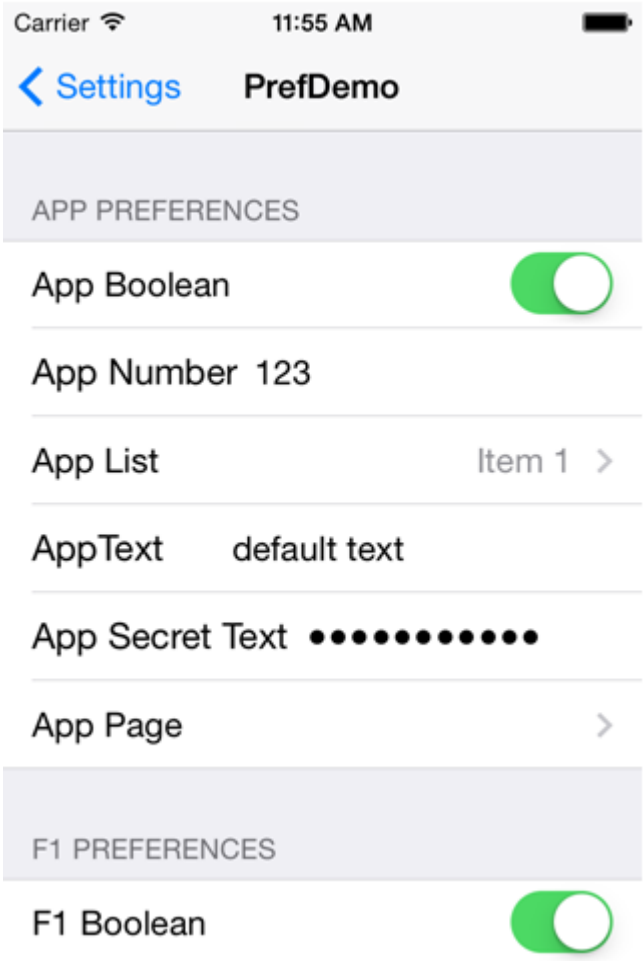
See *Tag Reference for Oracle Mobile Application Framework* for more information on these elements and their attributes.

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 located in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.maf/Samples`

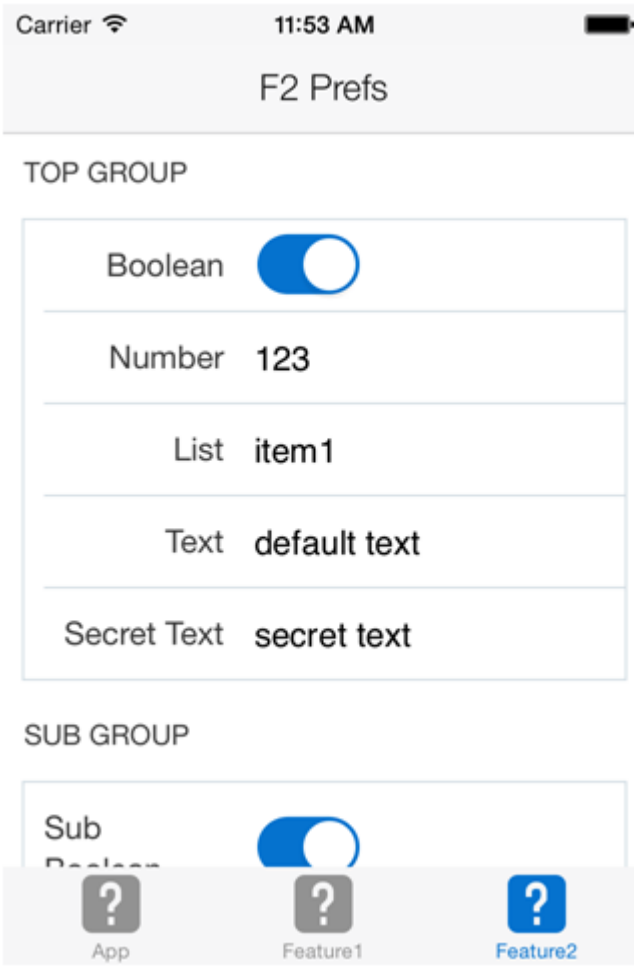
The PrefDemo application is comprised of an application-level settings page as well as three application feature preference pages, which are implemented as MAF AMX. Figure 21-2 shows the PrefDemo application settings page, which you invoke from the general settings page. In this illustration, the preference 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 Using EL Expressions to Retrieve Stored Values for User Preference Pages.

Figure 21-3 An Application Feature Preference Page from the PrefDemo Application

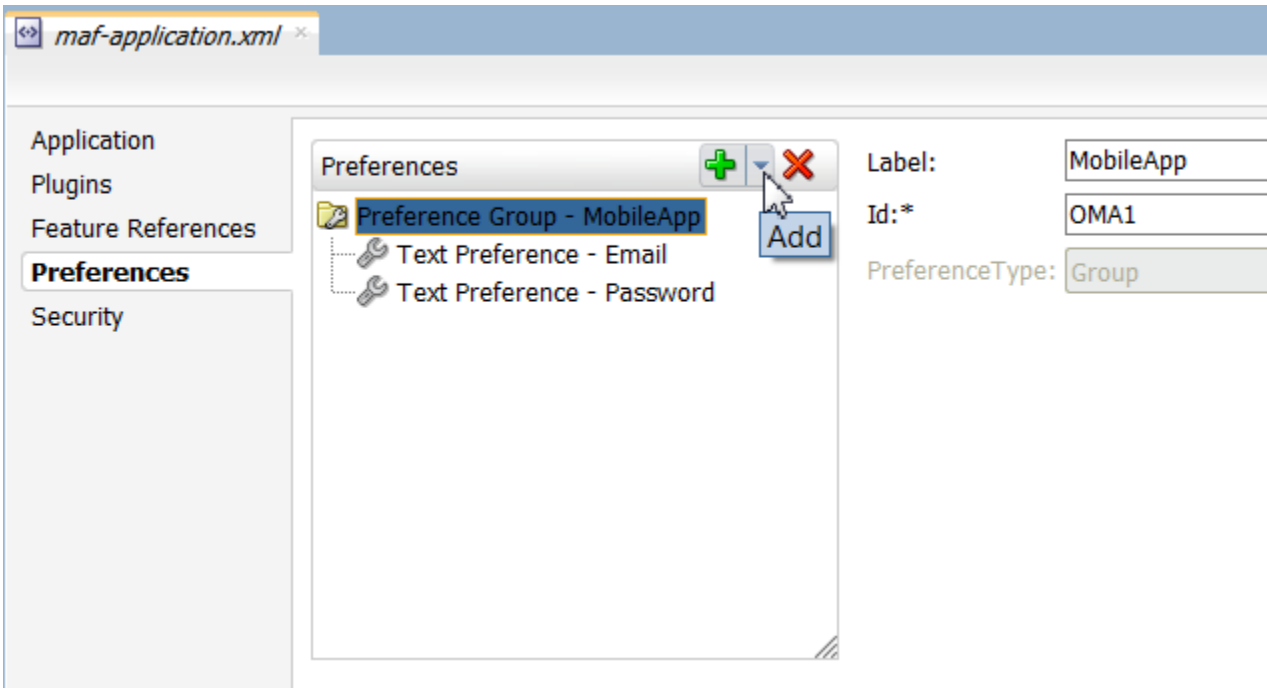


In the PrefDemo application, each MAF AMX preference page is referenced by a single bounded task flow comprised of a view activity and a control flow case that enables the page refresh.

21.1.1 How to Create Mobile Application-Level Preferences Pages

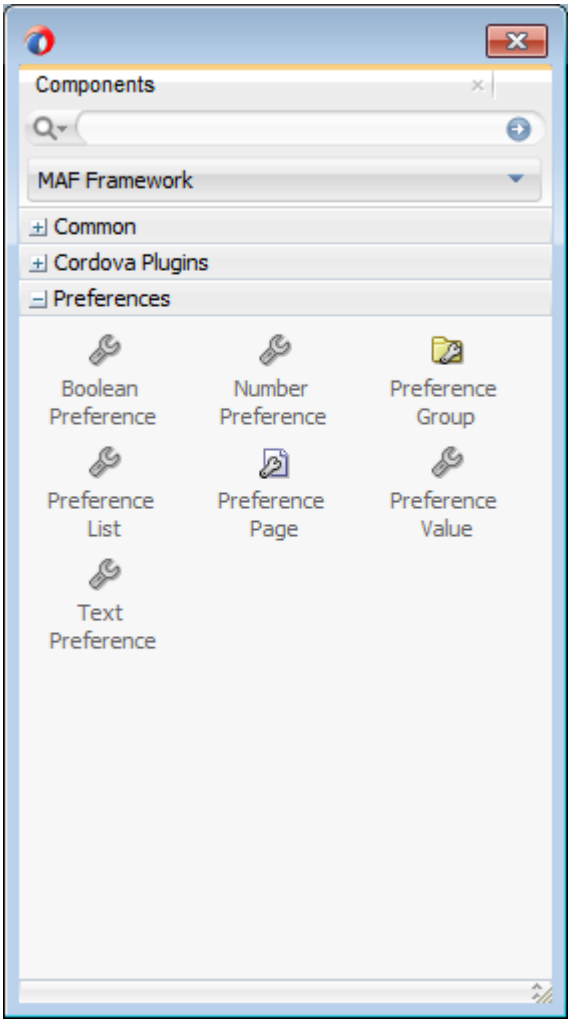
The **Preferences** page of the `maf-application.xml` overview editor, shown in Figure 21-4, enables you to build sets of application-level preference pages by nesting the child preference page elements within `<adfmf:preferenceGroup>`. The page presents the `<adfmf:preferenceGroup>` and its child elements as similarly named options (such as Preference Page, Preference List, Boolean Preference), which you assemble into a hierarchy (or tree), similar to the Structure window in JDeveloper.

Figure 21-4 Adding Mobile Application-Level Preferences Using the Preferences Page



To ensure that the `maf-application.xml` file is well-formed, use the **Preferences** page's **Add** drop-down list, shown in Figure 21-4, to construct the user preferences pages. While you can also drag components from the **Preferences** palette, shown in Figure 21-5, into either the editor, the **Source** window, or the **Structure** window, the page's drop-down list presents only the elements that can have the appropriate parent, child, or sibling relationship to a selected preferences element. For example, Figure 21-4 shows only the components that can be inserted within the **Preference Group** element, *MobileApp*. The editor also enables you to enter the values for the attributes specific to each preference element.

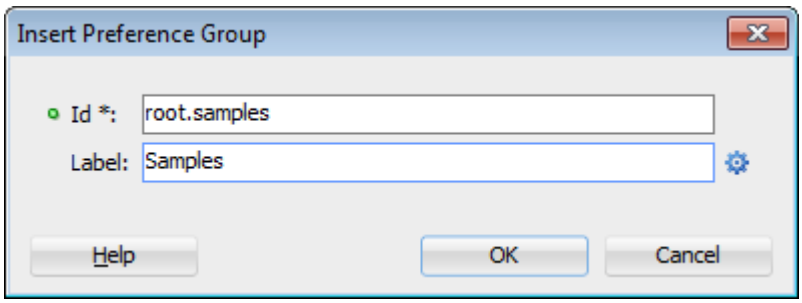
Figure 21-5 Preferences in the Component Palette



To create preferences pages:

1. In the `maf-application.xml` overview editor, click **Preferences**.
2. Click **Add** to create the parent `<adfmf:preferenceGroup>` element.
3. Enter the following information in the **Insert Preference Group** dialog, shown in Figure 21-6.

Figure 21-6 Defining the Parent Preference Group Element



- Enter a unique identifier for the **Preference Group** element.

- Enter the descriptive text that displays in the user interface. For an example of how this text displays in the user interface, see *Sample* in Figure 21-1.

4. Click **Add** to further define the preference pages using the **Insert Before**, **Insert Inside**, **Insert After** options to ensure that the XML document is well formed.

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. You create a **Preference Page** using the **Insert Before**, **Insert Inside**, **Insert After** options.

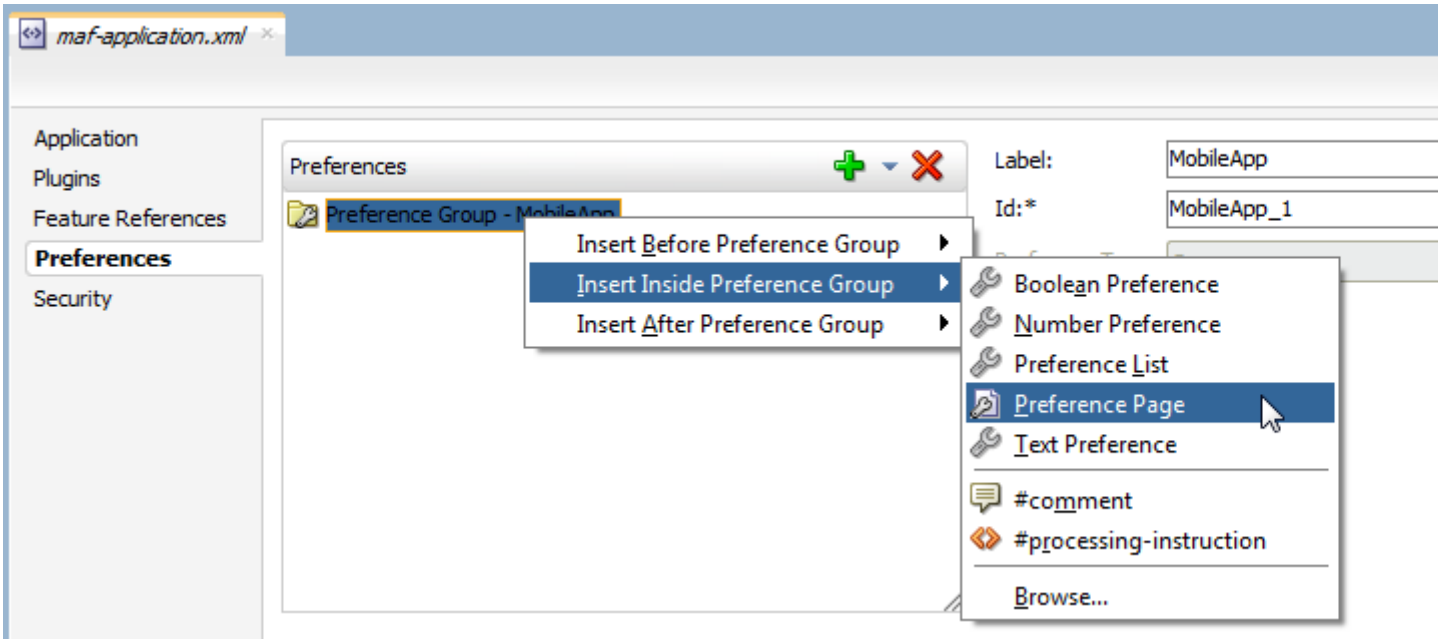
Before you begin:

You must create a **Preference Group** element.

To create a new user preference page:

1. In the **Preferences Page** of the `maf-application.xml` overview editor, select the **Preference Group** element. In this example, the **Preference Group** is called *MobileApp*.
2. Click **Add**, then choose **Insert Inside Preference Group** > **Preference Page**, as shown in Figure 21-7.

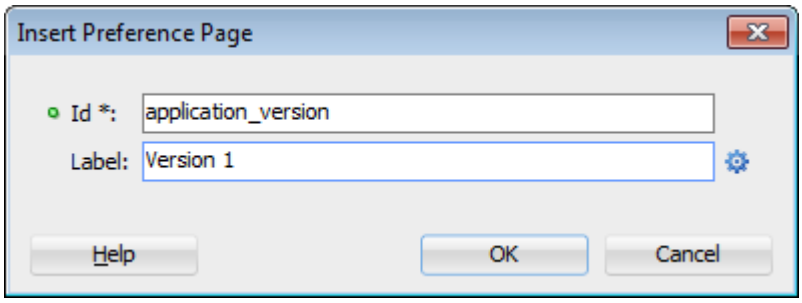
Figure 21-7 Selecting the Preference Page Component



3. Define the following **Preference Page** attributes in the **Insert Preference Page** dialog, shown in Figure 21-8:

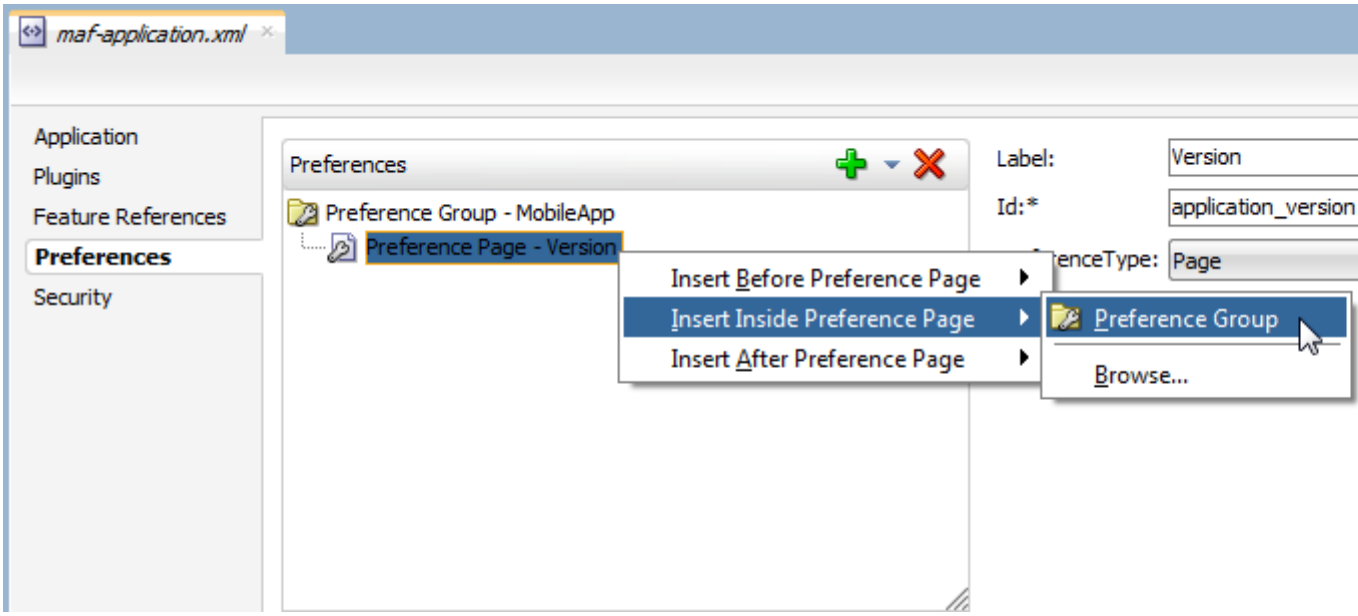
- Enter a unique identifier for the **Preference Page** element.
- Enter the descriptive text that displays in the user interface.

Figure 21-8 The Insert Preference Page Dialog



4. Create the body of the preference page by inserting a child **Preference Group** element by selecting the **Preference Page**, and then first choosing **Insert Inside Preference Page** and then **Preference Group**, as shown in Figure 21-9. 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, as shown in What Happens When You Add a Preference Page.

Figure 21-9 Adding a Preference Group to a Preference Page



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, JDeveloper generates an `<adfmf:preferencePage>` with attributes, as shown in the following example. The `<adfmf:preferencePage>` is nested within a parent `<adfmf:preferenceGroup>` 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.

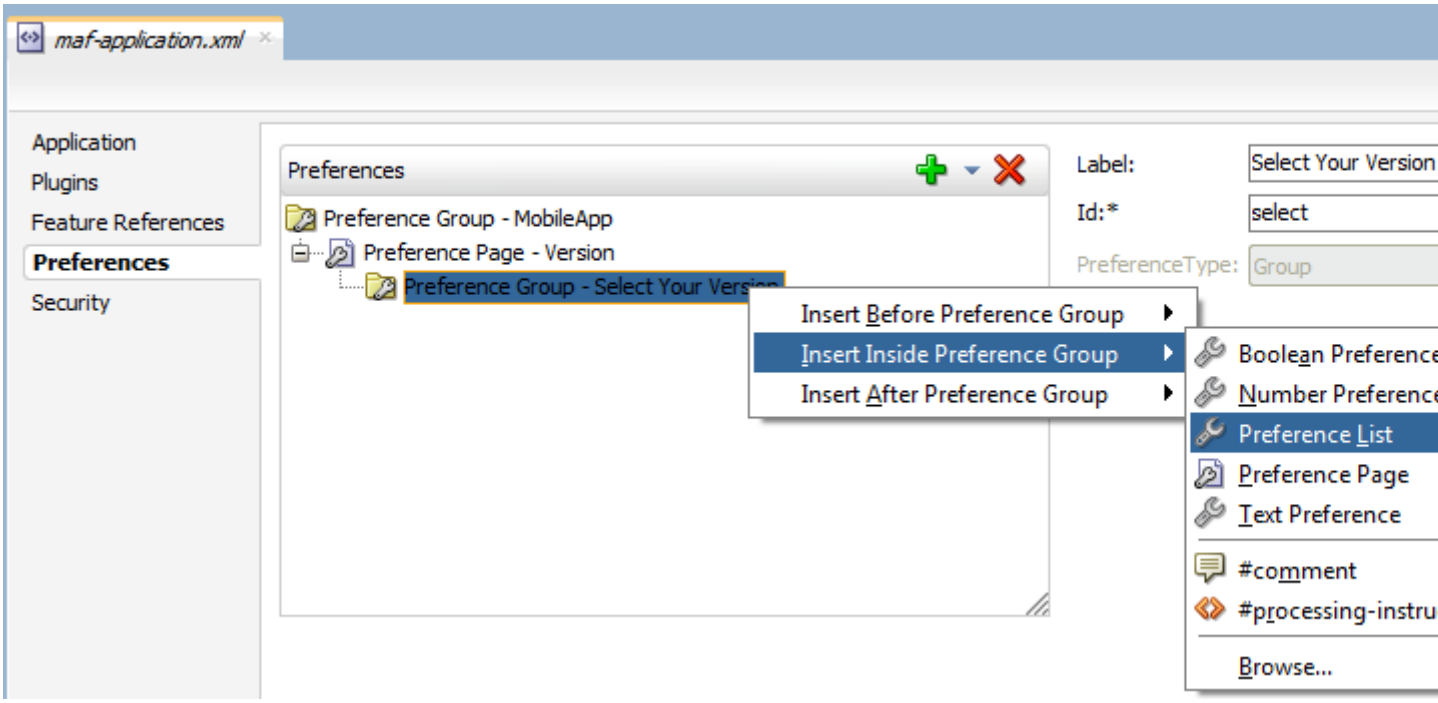
Before you begin:

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**. Figure 21-10 shows adding a **Preference List** as a child of a **Preference Group** component called *Select Your Version*.

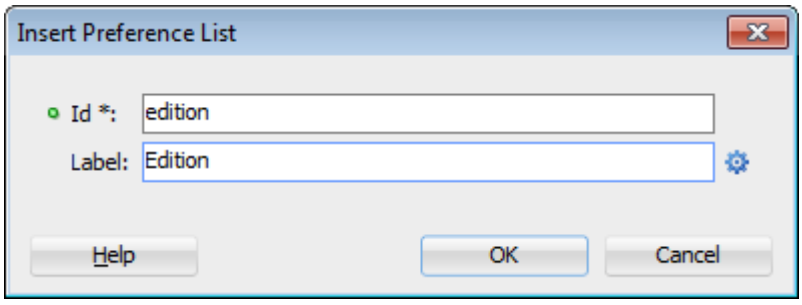
Figure 21-10 Adding a Preference List Component to a Preference Group



2. Define the following attributes using the **Insert Preference List** dialog, shown in Figure 21-11, and then click **OK**.

   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 21-11 The Insert Preference List Dialog



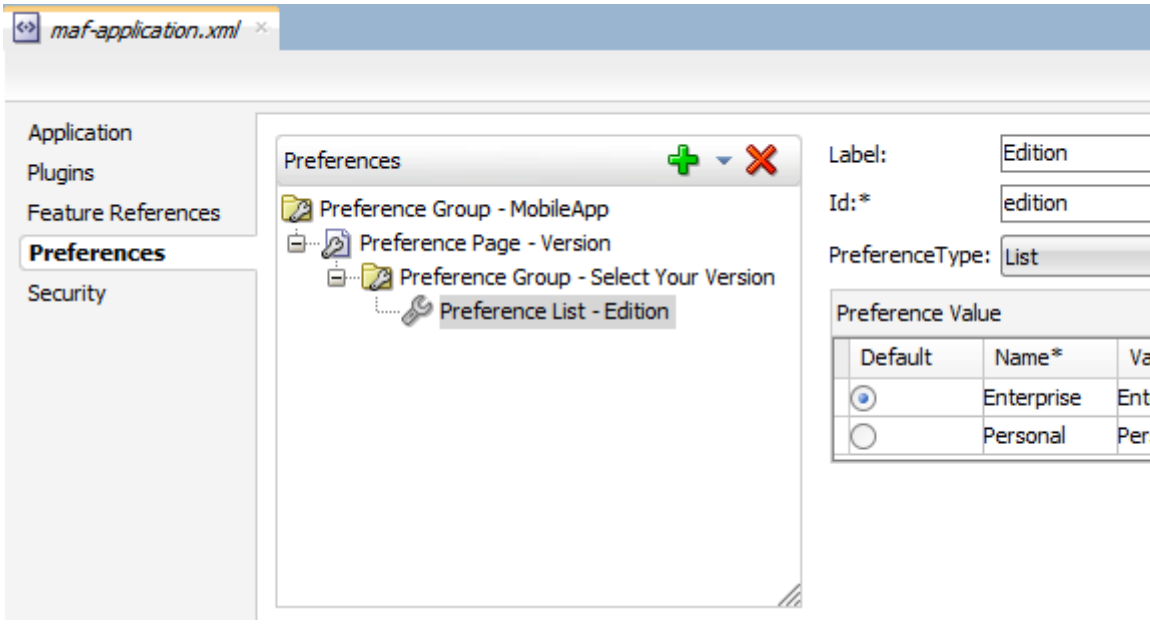
3. Define a list of items by clicking **Add** in the **Preference Value** table, shown in Figure 21-12. You can also remove a preference value by selecting it and then clicking **Delete**. You can change the order in which the preference values display by selecting the preference value and then using the up- and down-arrows.

   You can present the user with a default setting by choosing **Default**. As illustrated in What Happens When You Add a Preference Page, the default status is defined within the `<adfmf:preferenceList>` element as `default="ENTERPIRSE"`.

   Tip:

   In addition to clicking **Add**, you can add **Preference Value** components by dragging them either into the **Structure** window or the **Source** window.

Figure 21-12 Adding Preference Values



21.1.1.4 What Happens When You Create a Preference List

After you add a **Preference List** component to a **Preference Group** and then define a series of **Preference Values**, JDeveloper updates the `<adfmf:preferences>` section with an `<adfmf:preferenceList>` element, as shown in What Happens When You Add a Preference Page.

21.1.1.5 How to Create a Boolean Preference List

See, for example, Creating User Preference Pages for a Mobile Application.

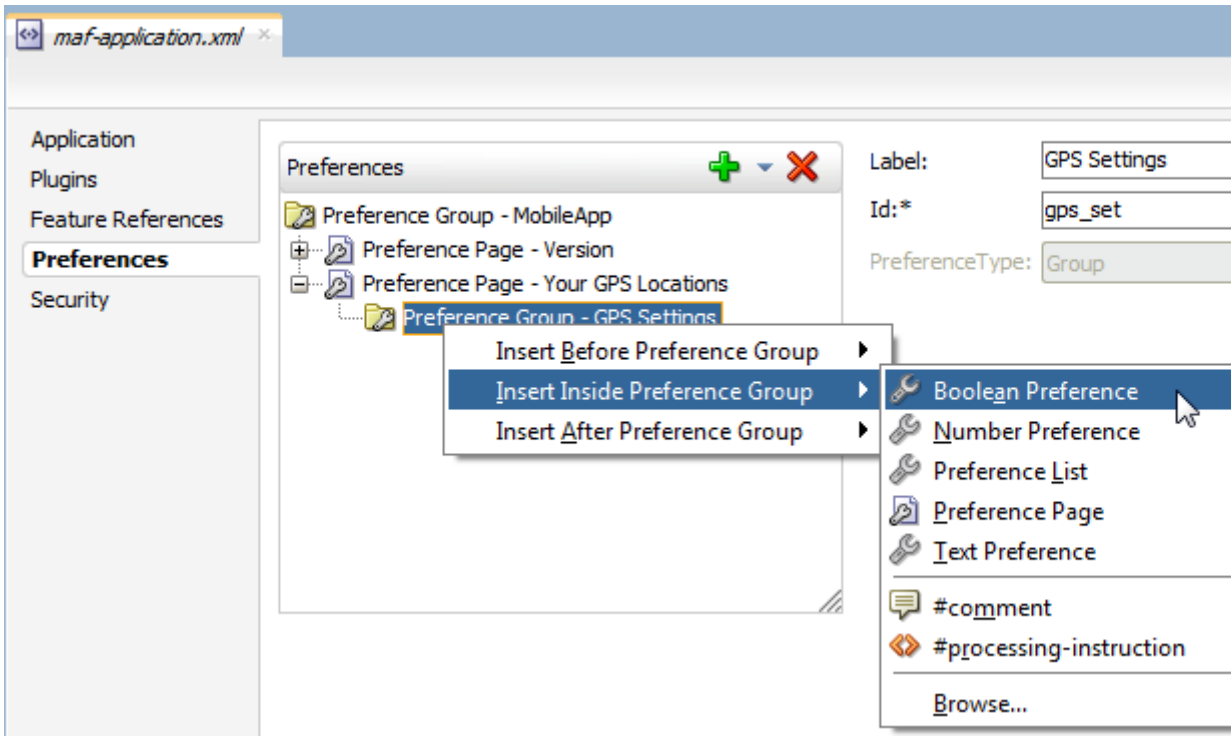
Before you begin:

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 user boolean list:

1. Select a **Preference Group** element, such as GPS Settings in Figure 21-13.

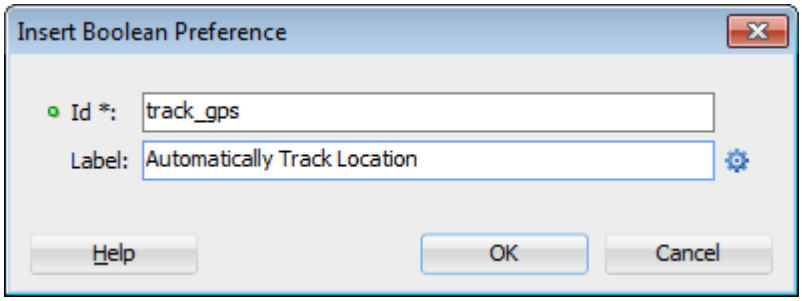
Figure 21-13 Adding a Boolean Preference to a Preference Group



2. Define the following attributes using the **Insert Boolean Preference** dialog, shown in Figure 21-14, and then click **OK**.

   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 21-14 The Insert Boolean Preference Dialog



3. Accept the default value of `false`, or select `true`.

21.1.1.6 What Happens When You Add a Boolean Preference

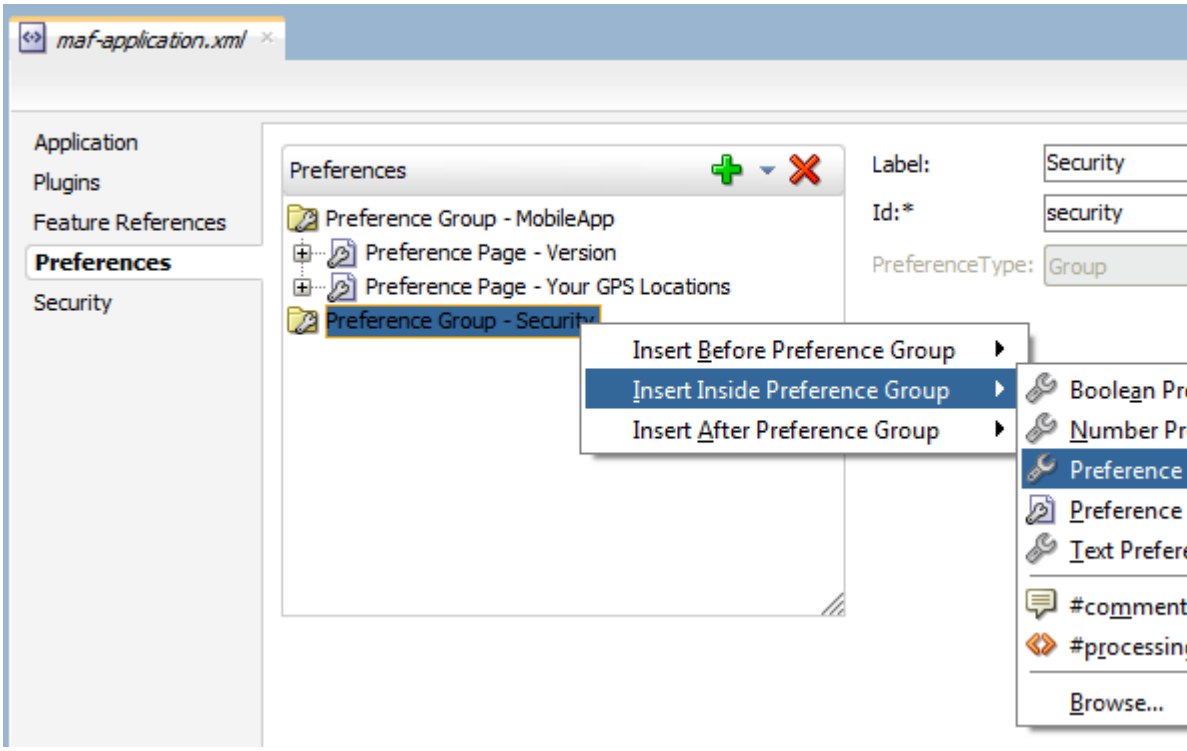
When you add a Boolean Preference and designate its default value, JDeveloper updates the `<adfmf:preferences>` section of the `maf-application.xml` file with a `<adfmf:preferenceBoolean>` element, as illustrated in the following example.

```
<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 21-15, to create a **Text Preference**, a dialog that enables users to store information or view default text. Figure 21-15 shows creating a text preference within a **Preference Group** called *Security*.

Figure 21-15 Inserting a Text Preference



Before you begin:

Create a **Preference Group** element.

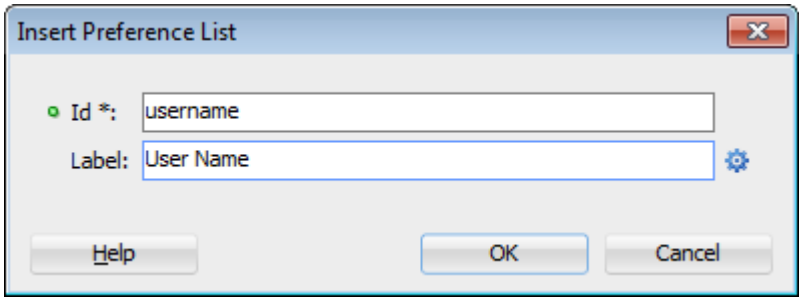
To create a text preference:

1. Select a **Preference Group** element.
2. Select **Insert Inside** and then **Text Preference**.

3. Enter the following information into the **Insert Text Preference** dialog, shown in Figure 21-16, and then click **OK**.

   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

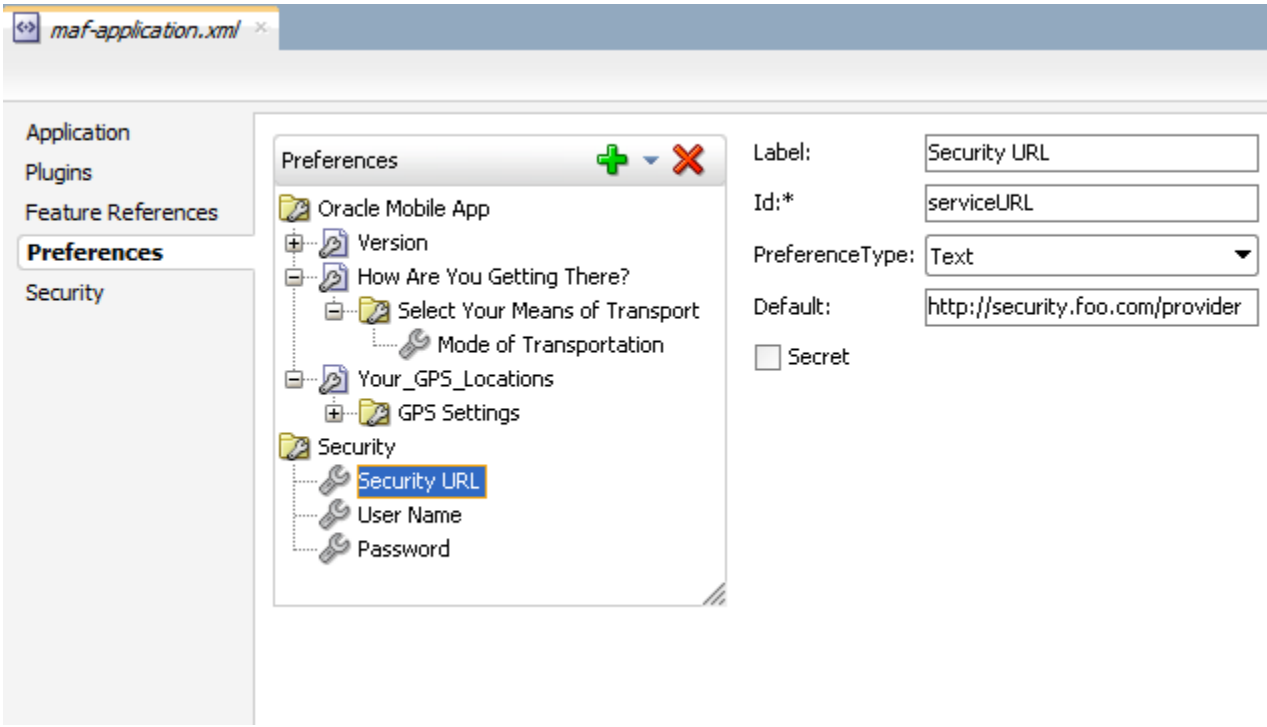
Figure 21-16 The Insert Text Preference Dialog



4. Define the following for the preference text dialog:

   - Enter the default text value.
   - Select **Secret** to hide the text preference.

Figure 21-17 Defining the Text Preference



21.1.1.8 What Happens When You Define a Text Preference

When you add a **Text Preference** and designate its default value, JDeveloper updates the `<adfmf:preferences>` section of the `maf-application.xml` file with a `<adfmf:preferenceText>` element, as illustrated in the following example.

```
<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 21-18.

Figure 21-18 Text Preferences

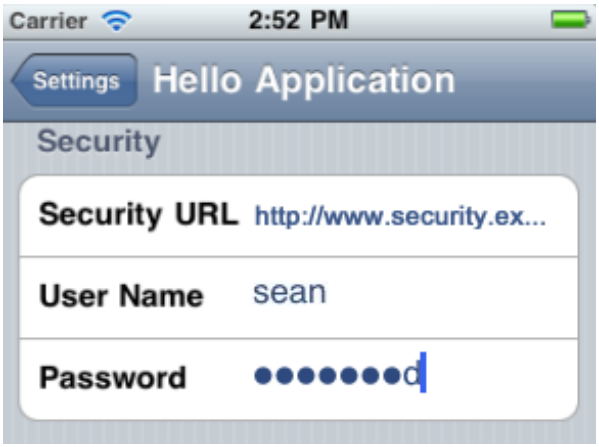


Figure 21-18 illustrates `<adfmf:preferenceText>` elements with a seeded value for the Security URL and an input value for the User Name. Because the MAF preferences are integrated with the iOS Settings application, the `secret="true"` attribute for the Password input text results in the application 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 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 Converting Preferences for Deployment.

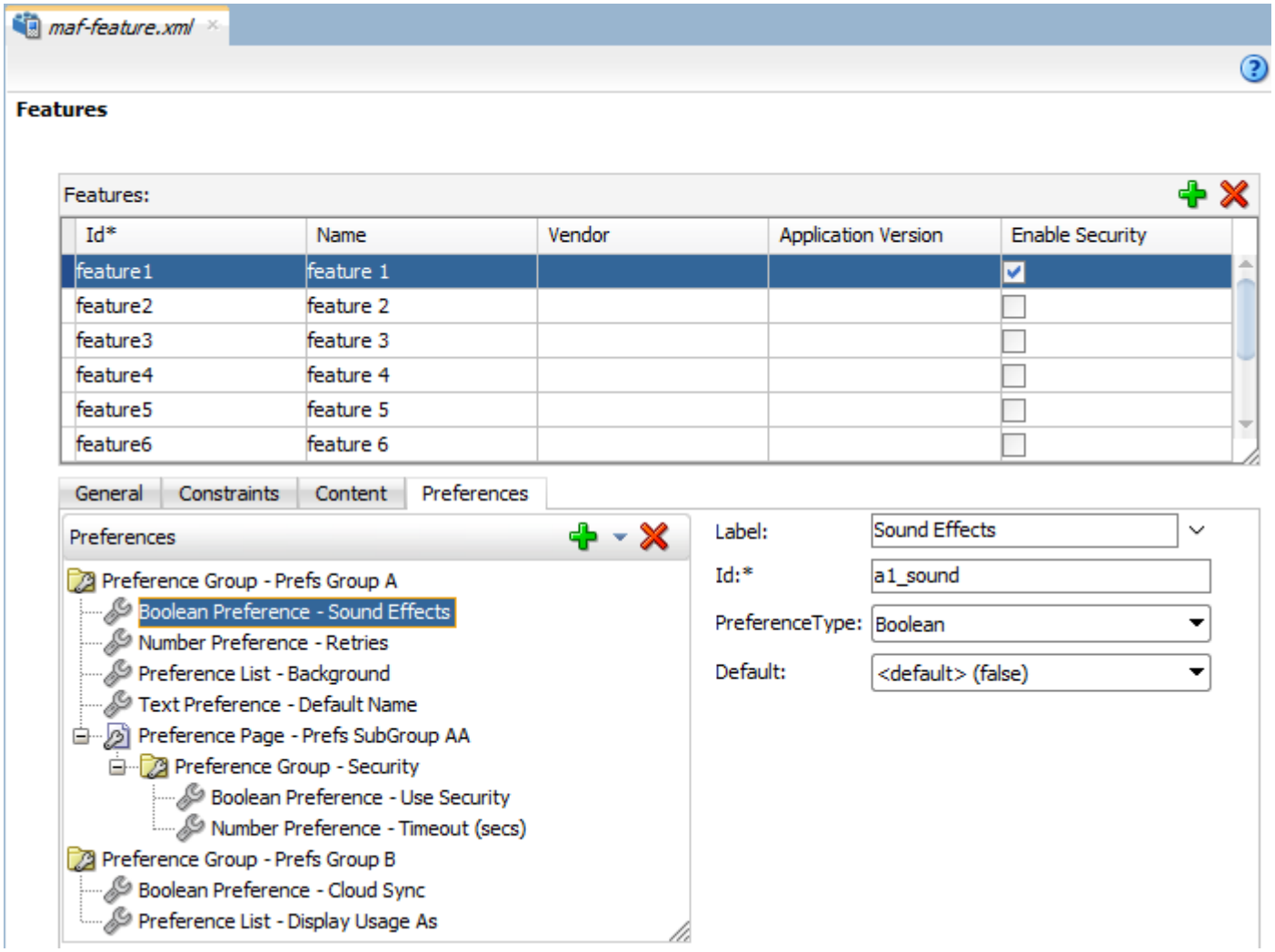
21.2 Creating User Preference Pages for Application Features

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.xml` overview editor, shown in Figure 21-19. You build application feature preferences in

the same manner as the application-level preferences, which are described in Creating User Preference Pages for a Mobile Application. After you define the preferences in the `maf-feature.xml` file, 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 Creating and Using UI Components.

Figure 21-19 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.

The MAF AMX components, such as the Input Text component in the following example.

```
<amx:inputText label="Number" id="it1" inputType="number"
               value="#{preferenceScope.feature.Feature1.f1top.f1Number}"/>
```

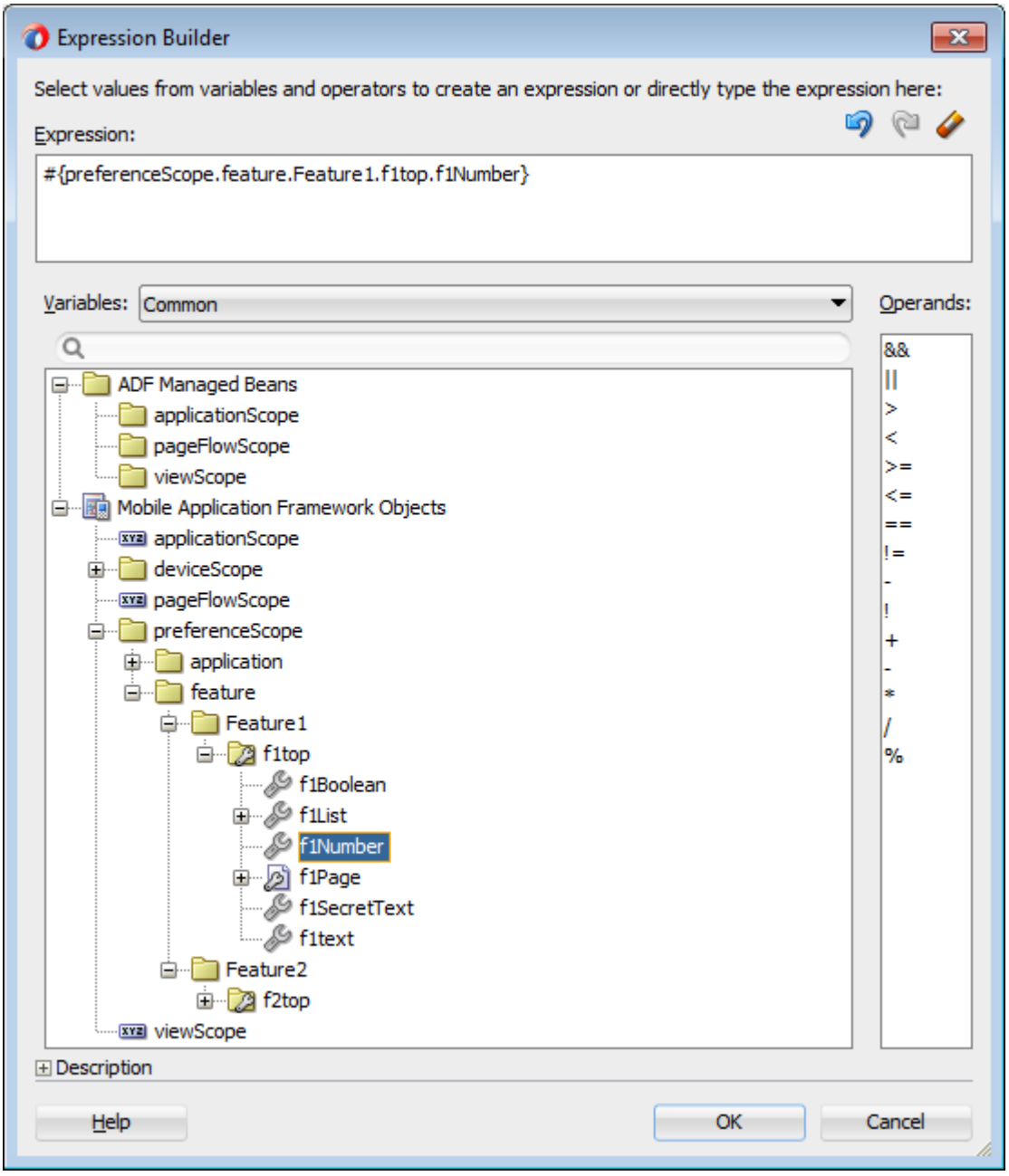
As illustrated in this example, EL expressions use the `preferenceScope` object to enable applications to access an application feature-level preference. These EL expressions are in the following format:

`preferenceScope.feature.`*`feature-id`*`.`*`group-id`*`.`*`property-id`*

Figure 21-20 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 21-20 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 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 following 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.

```
//...
-(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 the platforms where you deploy the MAF application.

Consequently, the MAF preference pages display differently on the supported platforms. Table 21-1 shows the display differences between the Android and iOS platforms. The 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

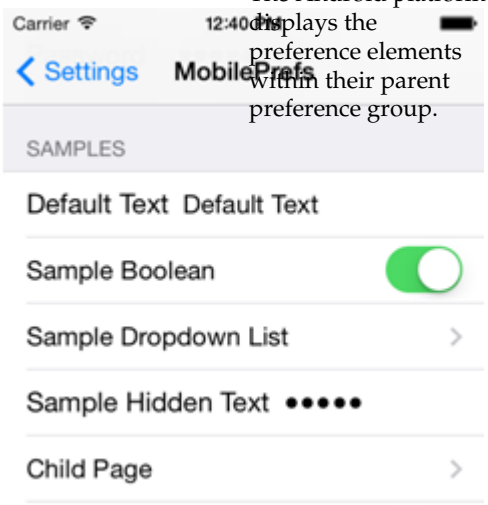
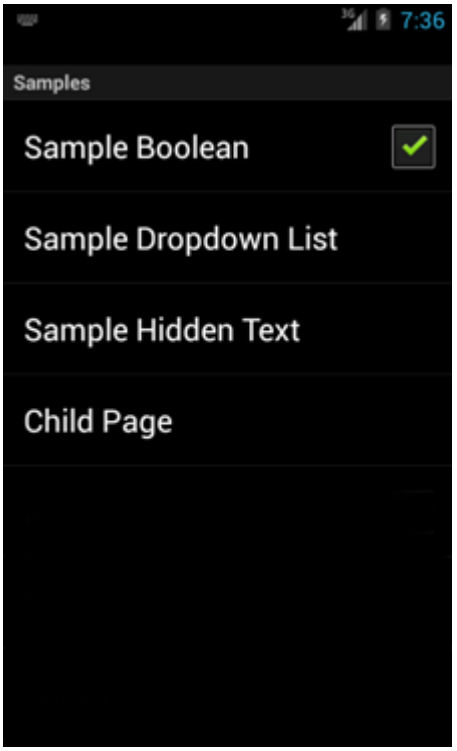
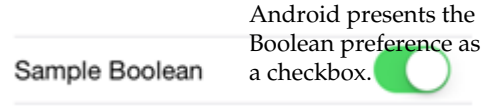
Component	iOS	iOS Display Examples	Android	Android Display Examples
Preference Groups (Category Selection)	The iOS platform displays the preference elements within their parent preference group.		The Android platform displays the preference elements within their parent preference group.	
Boolean Preference List	The Boolean preference is represented as value pair, such as *on* and *off*.		Android presents the Boolean preference as a checkbox.	

Table 21-1 (Cont.) Preference Component Comparison by Platform

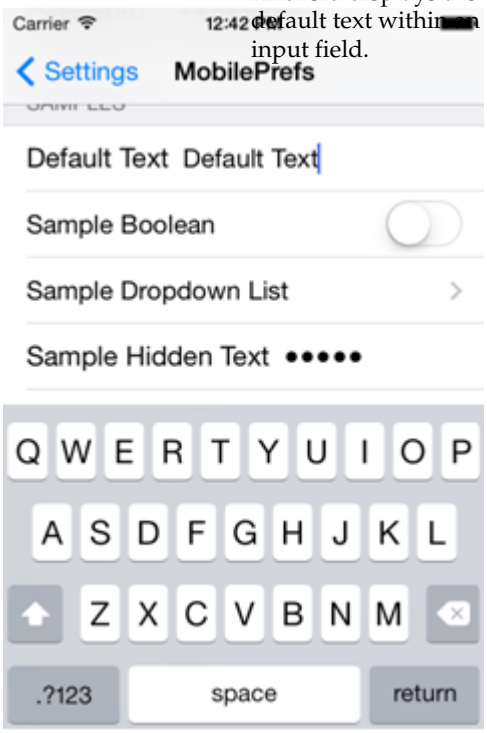
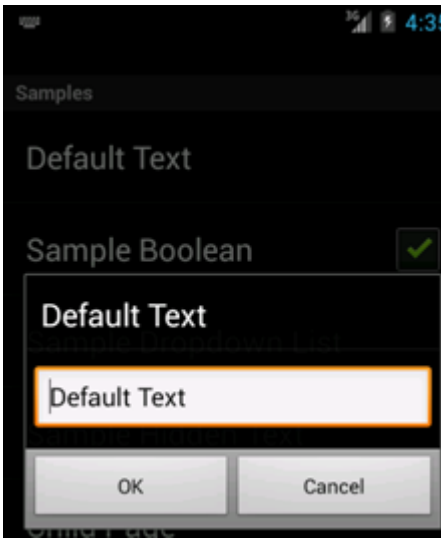
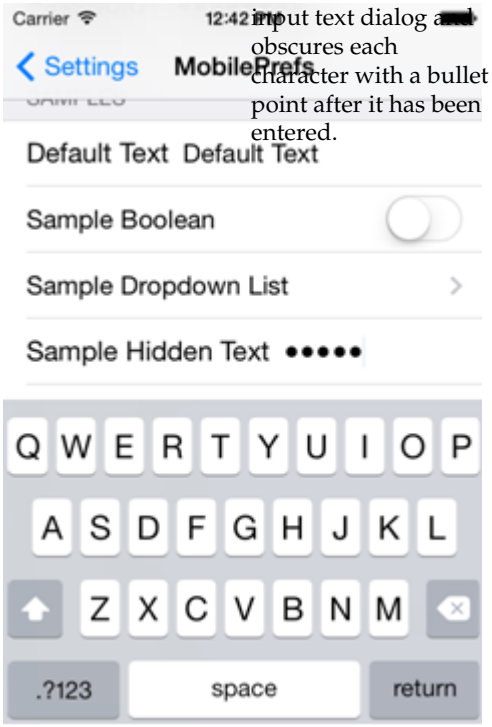
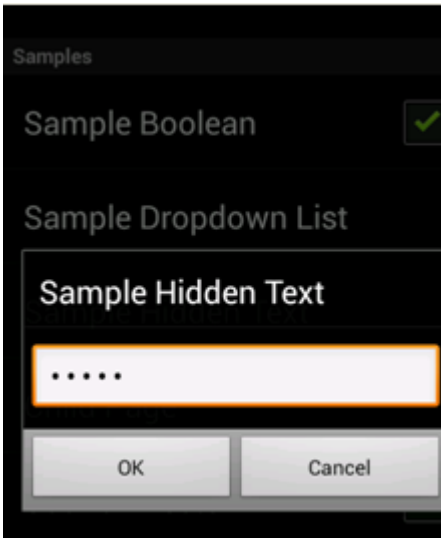
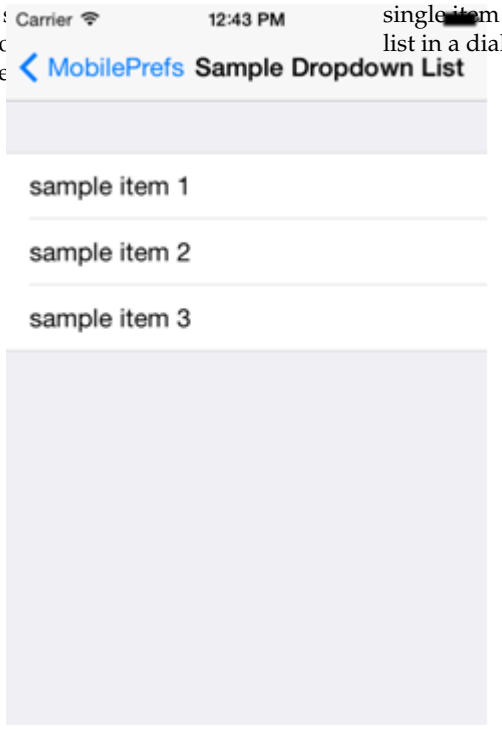
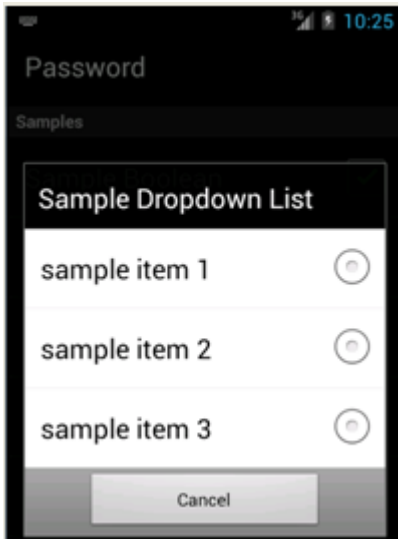
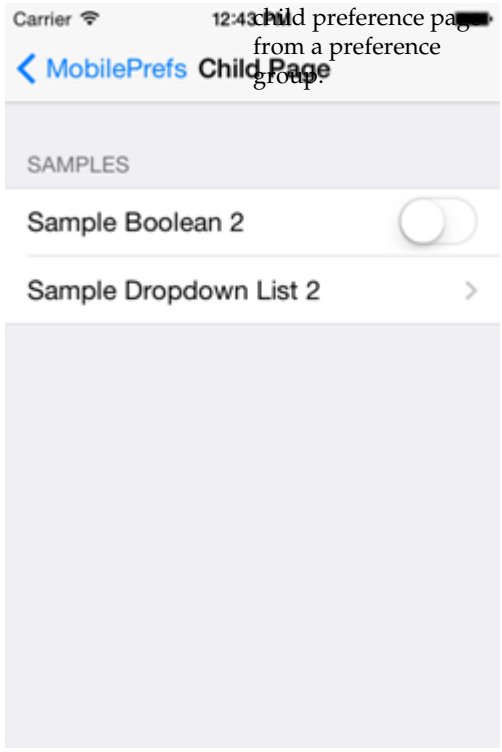
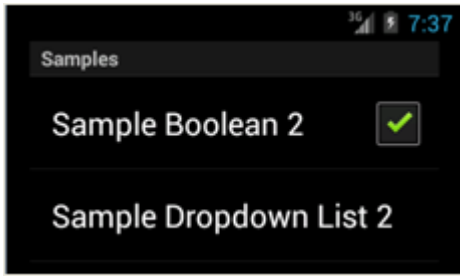
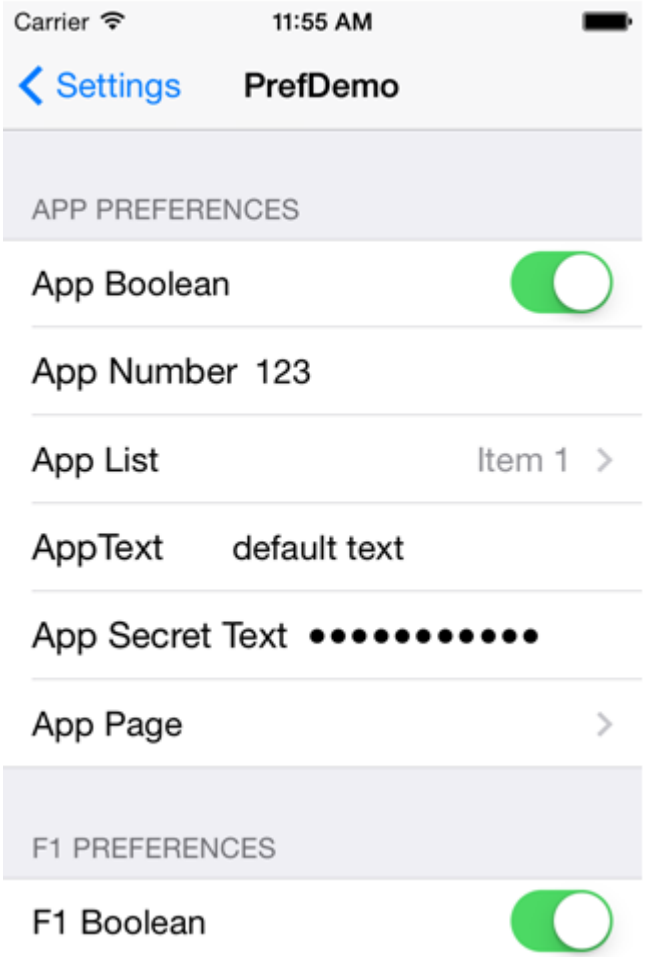
Component	iOS	iOS Display Examples	Android	Android Display Examples
Text Preference	iOS displays the text inline.		Android displays the default text within an input field.	
Text Preference (as secret input text)	On iOS platforms, user enter text inline, with each character obscured by a bullet point after it has been entered. For more information, see 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.	

Table 21-1 (Cont.) Preference Component Comparison by Platform

Component	iOS	iOS Display Examples	Android	Android Display Examples
Single Item Selection List (from a Preference List)	iOS platforms display the s item selectio separate pre 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.	

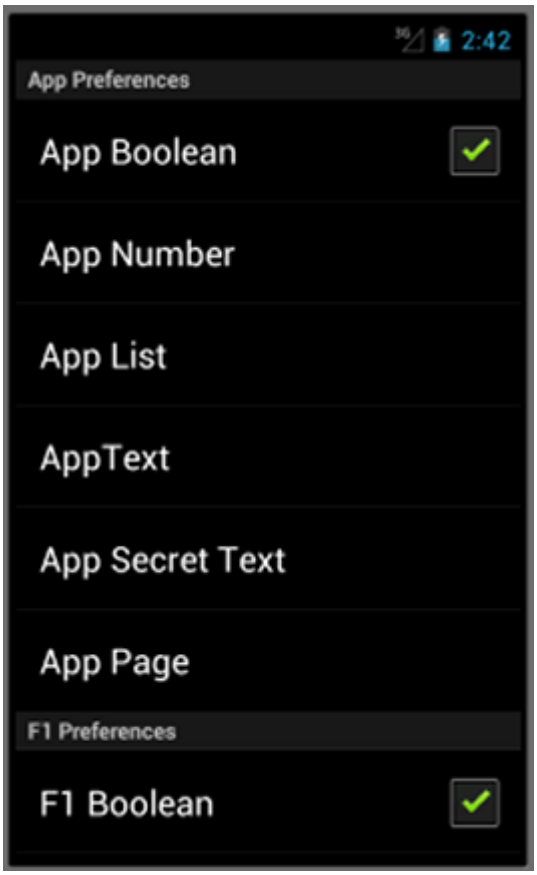
Although iOS and Android platforms have a Settings application, only the iOS platform supports integrating application-level preferences into the Settings application, as shown by the preferences in Figure 21-21.

Figure 21-21 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 21-22 only when the application is running.

Figure 21-22 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:

- Introduction to Constraints
- Defining Constraints for Application Features

22.1 Introduction to Constraints

Constraints are set either at the application feature level or at the content level to indicate when the feature or content must be used. Constraints can restrict access and are evaluated at runtime.

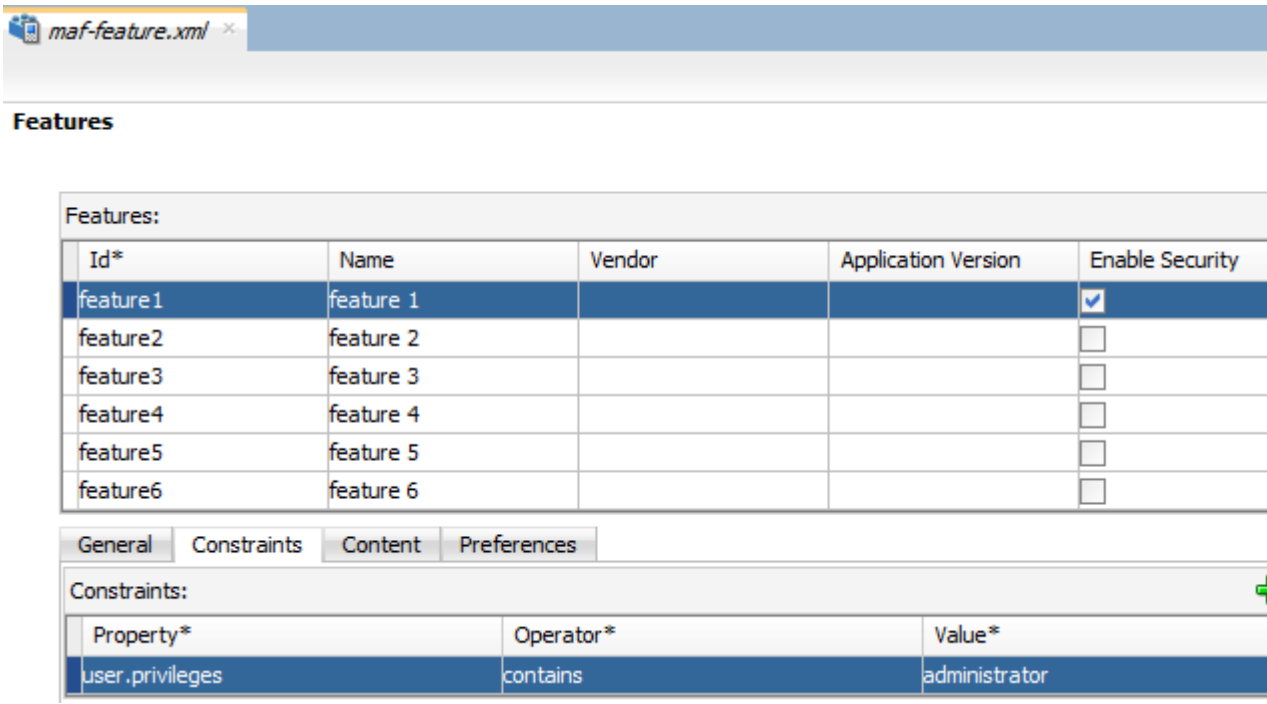
A constraint describes when an application feature or application content should be used. Constraints can restrict access based on users and user roles, the characteristics of the device on which the mobile application is targeted to run, and the hardware available on the device. You can set constraints 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 Contents folder in the MAF Feature Editor holds the Constraints folder. The MAF runtime evaluates constraints and depending on the constraint value that was set, either displays or hides content.

The **Constraints** tab, shown in Figure 22-1, enables you to set the application feature-level constraints. For example, an application feature that uses the device camera displays within the navigation bar or springboard of the mobile application 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 feature, should the MAF runtime evaluate the constraint to `true`. If the runtime evaluates the constraint to `false`, then it prevents an 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

A feature can accommodate different types of content to deliver different versions of the user interface. Set constraints that determine how the UI is displayed.

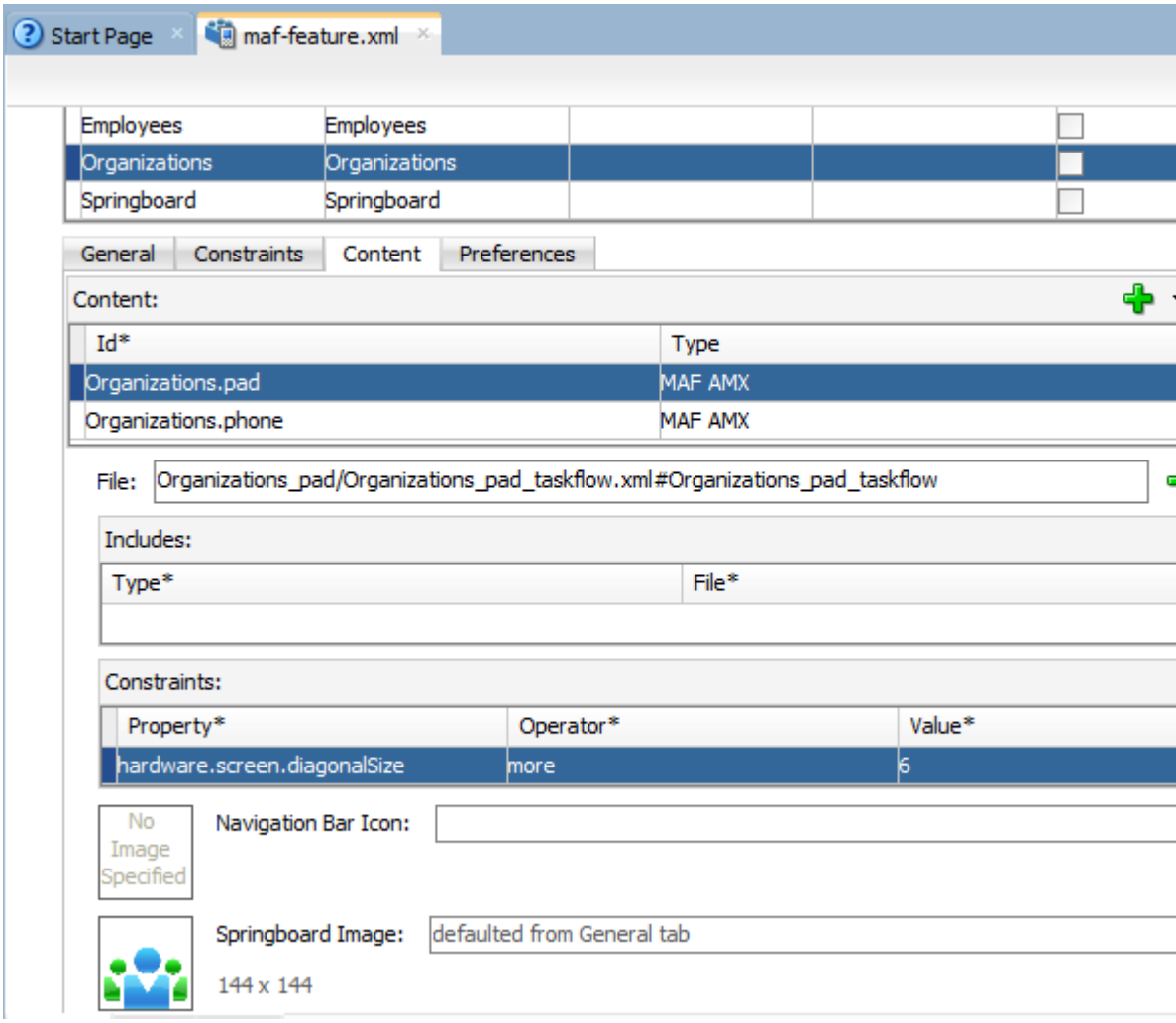
To accommodate such factors as device hardware properties or user privileges, a single application feature can have more than one type of content to deliver different versions of the user interface. By setting constraints on the content of an application feature, you designate the conditions for determining what users can see or how application pages can be used.

Using the **Content** tab, 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 a sample 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 user launches the sample 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 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 task flow that calls iPhone-specific pages, `Employees_phone_taskflow.xml`. In addition, the Content tab 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 General tab.

Note:

Images must adhere to the platform-specific guidelines, as described in Setting Display Properties for an Application Feature.

Figure 22-2 Setting Content-Level Constraints



For more information on the sample applications, see MAF Sample Applications.

22.2 Defining Constraints for Application Features

Set application feature-level constraints to restrict application usage. Use the `property`, `operator`, and `value` attributes of the `<adfmf:constraint>` element to define the restrictions.

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 following example, 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.

```
<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

Use the procedure to define the constraints for a selected application feature using the Constraints folder in the MAF Feature Editor.

You declaratively configure the constraints for a selected application feature using the Constraints tab in the Features page, shown in Figure 22-2.

To define the constraints for an application feature:

1. Click the **Constraints** tab.
2. Click **Add**.
3. Select a property and an appropriate operator and then enter a value. See About the property Attribute.

22.2.2 What Happens When You Define a Constraint

Entering the values in the Constraints tab updates the descriptor file's `<adfmf:constraints>` element with defined `<adfmf:constraint>` elements, similar to the example shown in Defining Constraints for Application Features.

22.2.3 About the property Attribute

A set of property attributes together with operators and appropriate values determine how an application feature is used.

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

The MAF runtime matches a user login against configured user role-based constraints and associated privileges. Manage access to application features using constraints based on `user.roles` and `user.privileges`.

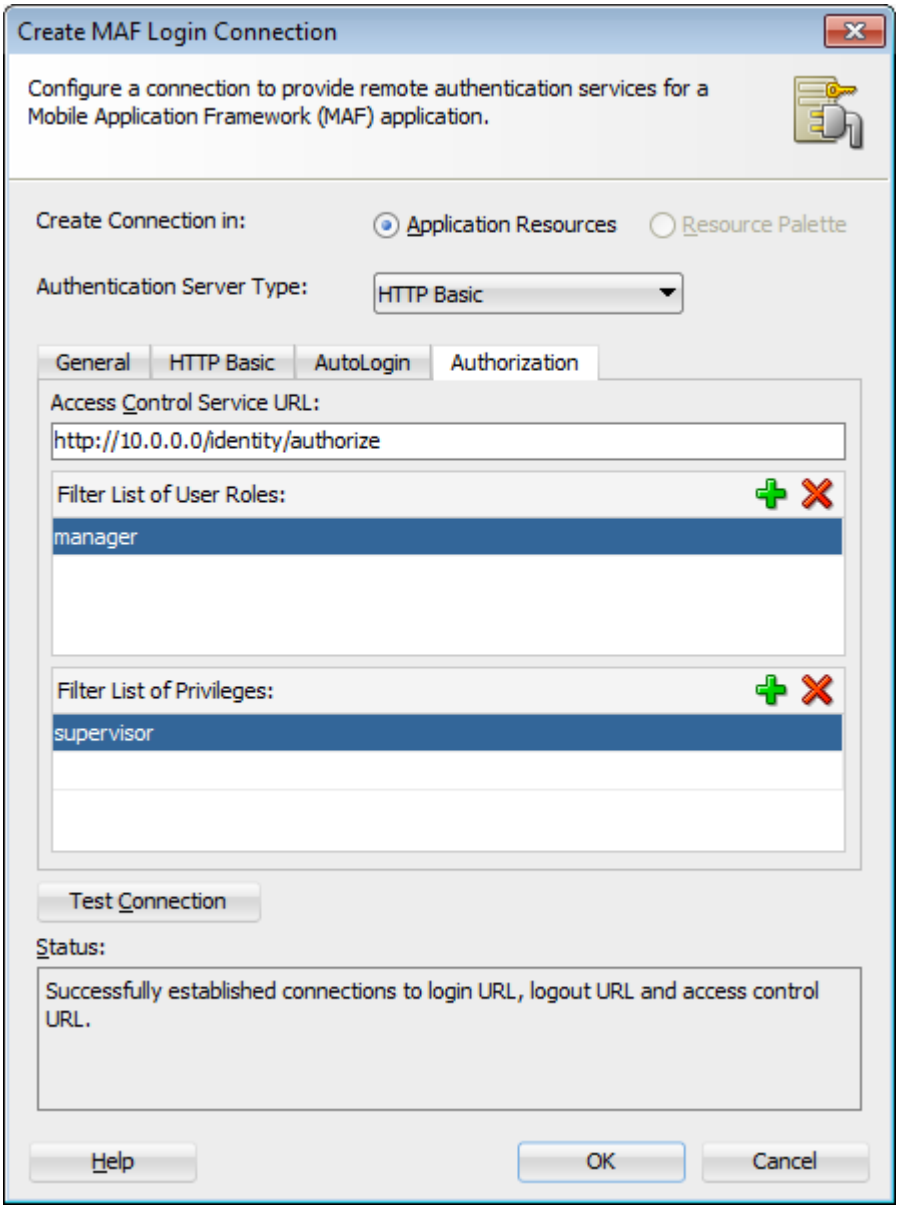
After a user logs into a mobile application, the MAF runtime reconciles the user role-based constraints configured for each application feature against the user roles and privileges retrieved by the Access Control Service (ACS). MAF then presents only the application feature (or application feature content) 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 **Create MAF Login Connection** dialog, shown in Figure 22-3 (and described in 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 What You May Need to Know About the Access Control Service.

Figure 22-3 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 following example shows how to use 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 following example, the application feature is not included if the user's privileges contain the manager privilege.

```
<feature ...>
   ...
   <constraints>
      <constraint property="user.privileges"
                  operator="not"
                  value="manager" />
   </constraints>
   ...
</feature>
```

22.2.5 About Hardware-Related Constraints

Set hardware-related constraints to make features using hardware available on a device. Define constraints such as `hardware.hasCamera`, `hardware.screen.height`, and `hardware.hasCompass` with the attribute value of `true` or `false` so that features become available if the value is `true`.

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 following example 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.

```
<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 following example shows a `true` value, indicating that this application feature is only available if the hardware has an accelerometer.

```
<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 following example, the value is set to `true`, indicating that the application feature is only available if the device includes a camera.

```
<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 following example.

```
<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 following example.

```
<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 following example. 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 following example. 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 geolocation 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 following example. 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 following example. 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 following example. 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 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 following example. 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 width in terms of logical device pixels (such as such as 320 in the following example), 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 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 height in terms of logical pixels, such as 320 or 480, as shown in the following example. 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 screen in its current orientation. Enter a numerical value that reflects the screen width in terms of logical pixels, such as 320 or 480, as shown in the following example. 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 width in terms of logical pixels, such as 320 or 480, as shown in the following example. 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

Constraints defined with EL expressions cause a mobile application to render application features and content dynamically. Definitions with EL expressions provide flexibility in that an EL expression evaluated as `false` can later be evaluated as `true`.

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 following example.

```
<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>
```

This example also shows how you can nest the <adfmf:constraintExpression> element among the static constraints defined within the `<adfmf:constraints>` element of the `maf-feature.xml` file. See *Tag Reference for Oracle Mobile Application Framework*.

22.2.6.1 About Combining Static and EL-Defined Constraints

The MAF runtime must evaluate all the criteria of a static constraint as `true` to enable it to be displayed. The runtime 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

Define a dynamic constraint by creating an `<adfmf:constraintExpression>` by dragging the **Constraint Expression** component into the Source editor or the Structure

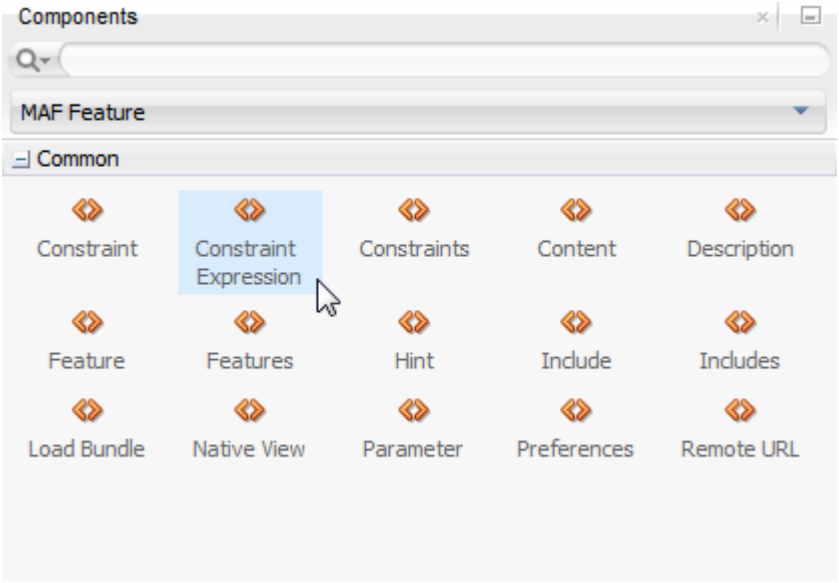
window and by populating the component with an EL expression. Use the procedure to define a **Constraint Expression** component.

Unlike static constraints, you neither create nor 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 a **Constraint Expression** component:

1. Select the Source editor for the `maf-feature.xml` file.
2. Navigate to the `<adfmf-constraints>` element.
3. In the Components window, select the **Common** components, as shown in Figure 22-4.

Figure 22-4 The Constraint Expression Component



4. Select the **Constraint Expression** component and add it to the `<adfmf-constraints>` element using any of the following methods:
   - Double-click the **Constraint Expression** component in the Components window.
   - Drag the **Constraint Expression** component into the `<adfmf-constraints>` element in the Source editor.
   - Drag the **Constraint Expression** component into the **Constraints** node of the Structure window.
5. Enter the EL expression in the Insert constraintExpression dialog, shown in Figure 22-5, or create an EL expression with the **Expression Builder**, which you access by clicking the **Property Menu** icon (the gear) in this dialog.

Figure 22-5 Defining an EL Constraint Using the InsertconstraintExpession Dialog

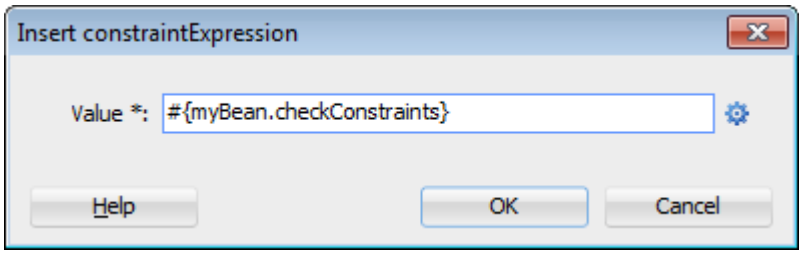
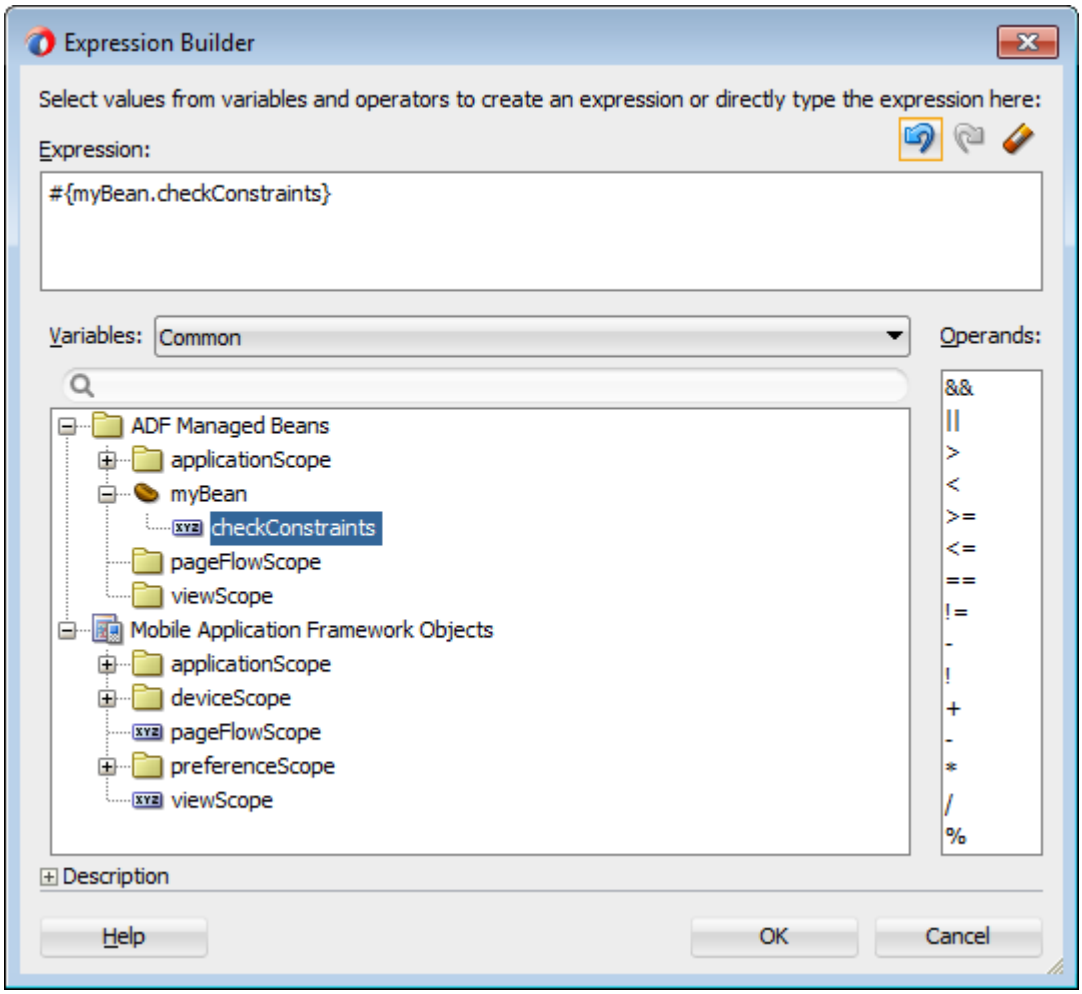


Figure 22-6 shows how to create an EL expression from the ADF Managed Bean category. However, you can create the EL expression of a constraint from any of the categories described in About the Categories in the Expression Builder.

Note:

Only application scope managed beans defined in `adfc-mobile-config.xml` can be used in the EL expression of a constraint.

Figure 22-6 Building the EL Expression of a Constraint



6. Click **OK**.

23

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:

- Introduction to Notifications
- Enabling Push Notifications
- Managing Local Notifications
- Determining Application State When MAF Triggers a Notification Event

23.1 Introduction to Notifications

Notifications are audio, visual, or audio-visual signals such as alerts or banners that users receive without user interaction outside of an application, if the application is running. Push notifications are sent from an external source, such as a server, to a registered application on a mobile device, and local notifications are sent by an application and received by itself.

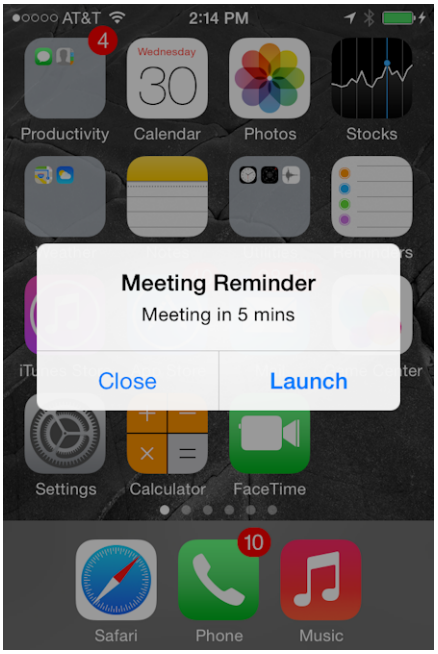
Notifications are signals delivered to the user outside of the regular user interface of a mobile application. These notifications can appear as messages in the form of an alert, or as a banner, depending on the state of the application and user settings. The notifications may be presented visually or 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 users are notified, they can either launch the application or they can select to ignore the notification in which case the application is not activated.

  Figure 23-1 shows a push notification alert on an iOS-powered device.

Figure 23-1 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 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 23-1 describes how the iOS operating system handles push notifications depending on the state of the MAF application.

Table 23-1 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.

Table 23-1 (Cont.) Handling Push Notifications on an iOS-Powered Device

State	Action
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 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 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 user taps on the notification.

  MAF applications on the Universal Windows Platform do not support local or push notifications.

23.2 Enabling Push Notifications

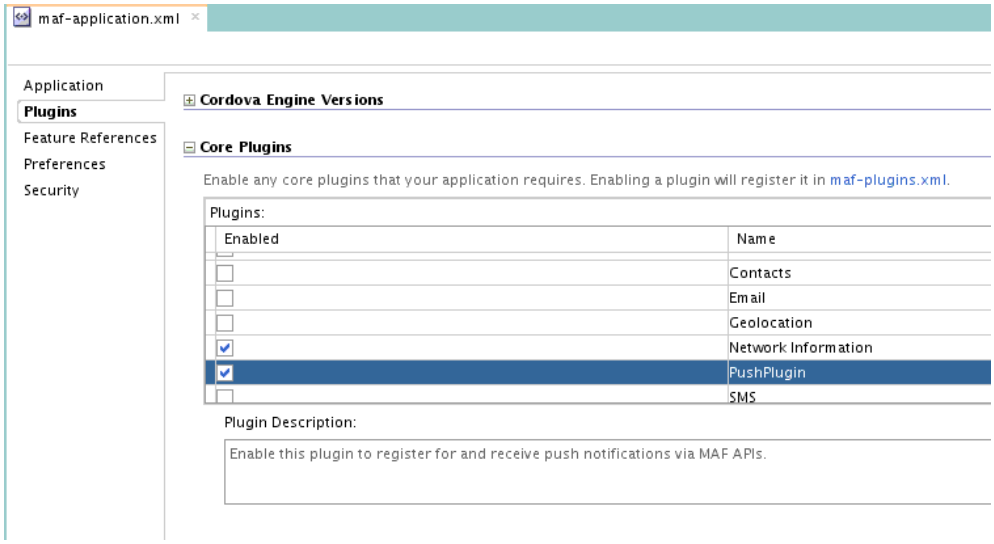
To enable push notifications, enable the PushPlugin, associate it with application features, create a push notification event listener class, and add an object to the event source. Also, register an application lifecycle event listener class, implement the `oracle.adfmf.application.PushNotificationConfig` interface, and add an `EventSource` object to its `start` method.

You can enable push notifications by performing the following tasks:

1. Allow the MAF application to receive push notifications by selecting **PushPlugin** in the **Core Plugins** section of the **Plugins** page of the `maf-application.xml` overview editor, as shown in the following illustration.

Note:

By default, a MAF application does not allow push notifications.



See Using Plugins in MAF Applications.

2. In the application controller project, register an application lifecycle event listener (ALCL) class. See Setting Display Properties for an Application Feature and Using 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 MAF 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());
```

Note: The MAF for iOS deployment profile displays a **Push Notification Environment** dropdown list from where you select `Production` or `Development` to register your deployed application with the Apple Push Notification service (APNs). The default value is `Production`. Set this value appropriately for MAF applications that you deploy to iOS. MAF ignores the value that you select if you do not also select the PushPlugin plugin to enable push notifications. For more information, see Defining the iOS Build Options.

MAF sample applications called PushDemo and PushServer demonstrate how to handle push notifications. These sample applications are located in the `PublicSamples.zip file` within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

23.2.1 How to Register a Device for Push Notifications from MCS

Enable push notifications in the MAF application, and then register the device with MCS to receive push notifications from MCS on the device.

To enable a device to receive push notifications from MCS, the device needs to be registered with MCS. This registration is application specific, so you have to register each MAF application with MCS. Before doing this, enable push notifications in your MAF application, as described in Enabling Push Notifications.

Use the following method in `MCSPersistenceManager` to register the device-application combination with MCS. The following code assumes that your application includes a `mobile-persistence-config.properties` file that is generated when you use the MAF client data model wizards, described in Creating the Client Data Model in a MAF Application, to create the data model of the MAF application.

```
public String registerDevice(String token, String appId, String appVersion)
```

You can call this method from the push notification event listener class that implements the `EventListener` interface, as described in Enabling Push Notifications.

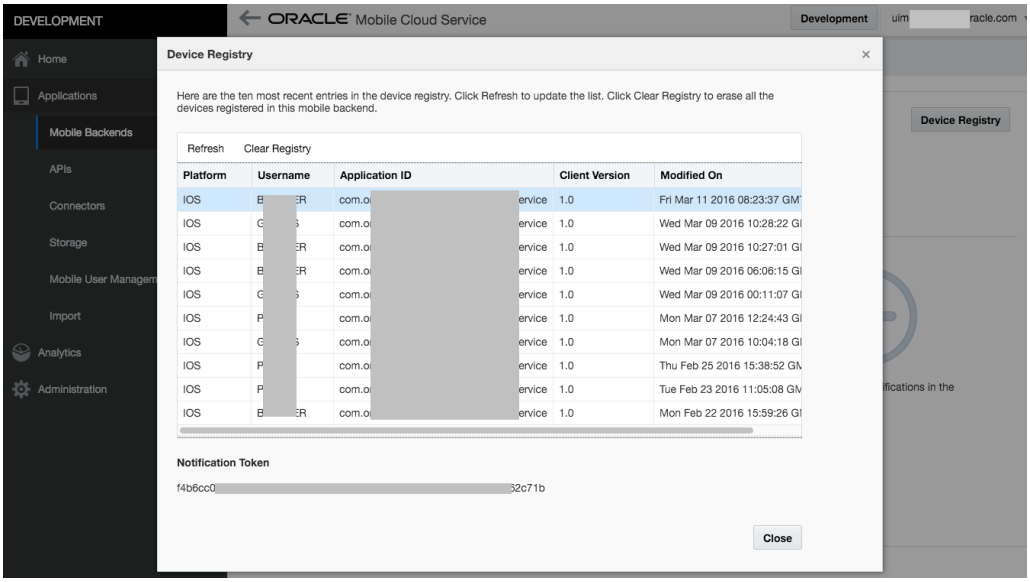
In this class, the `onOpen` method is automatically called by MAF and this method passes in the token that you need to register your device. Here is a code sample:

```
public void onOpen(String token) {
  try {
    String result = new MCSPersistenceManager().registerDevice(token,
"com.company.MyMAFApp", "1.0");
  }
  catch (Exception e) {
```

```
    throw new AdfException(e.getLocalizedMessage(), AdfException.ERROR);
  }
}
```

To check whether the registration was successful, click on the **Device Registry** button in the MCS UI, as shown in Figure 23-2.

Figure 23-2 Device Registry in MCS



As shown in Figure 23-2, the registration includes the username that was used to authenticate against MCS. This means that if you register the device directly in the `onOpen` method as shown above, the registration uses the anonymous user specified through the anonymous access key in the Connection page, described in Connecting to a REST Service to Create the Client Data Model. This is because the MAF login screen only appears once you try to access a secured feature. Secured feature access takes place after the lifecycle listener `start` method fired which in turn triggered the `onOpen` method to fire. This is not a problem as long as you only want to send push notifications to all registered devices. However, if you want to send push notifications to targeted devices based on usernames, you need to postpone the registration until after MAF login. This means that in the `onOpen` method you need to store the token in an application-scoped variable, and then within the first secured feature you make the call to register the device.

Use the following method in the `MCSPersistenceManager` class to deregister a device:

```
public String deregisterDevice(String token, String appId)
```

23.2.2 What You May Need to Know About the Push Notification Payload

The JSON-formatted push notification payload includes the `alert`, `sound`, and `badge` keys.

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 Implementing GCM Server in *Google Cloud Messaging*. This document is available from the Android Developers website at `http://developer.android.com/index.html` or the Android SDK documentation.

23.3 Managing Local Notifications

Manage local notifications with Java APIs, JavaScript APIs, and the methods of the `DeviceFeatures` data control provided by MAF.

You can manage local notifications by using the following:

- Java APIs provided by MAF (see How to Manage Local Notifications Using Java).
- JavaScript APIs provided by MAF (see 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 How to Manage Local Notifications Using the DeviceFeatures Data Control).

A MAF sample application called LocalNotificationDemo demonstrates how to schedule and handle local notifications. This sample application is located in the `PublicSamples.zip file` within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.maf/Samples` directory on your development computer.

23.3.1 How to Manage Local Notifications Using Java

Use the methods of the `oracle.adfmf.framework.api.AdfmfContainerUtilities` class to schedule local notifications.

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. See 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");
}
```

See *Java API Reference for Oracle Mobile Application Framework*.

23.3.2 How to Manage Local Notifications Using JavaScript

Use the methods that are available to manage local notifications using JavaScript APIs.

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
        {
           // notification title (only Android and iOS 8.2 or later)
           "title" : String,
           // notification alert text
           "alert" : String,
           // date-time at which notification should be fired (UTC time zone)
           "date" : Date,
           // either 'minutely', 'hourly', 'daily',
           // 'weekly', 'monthly', or 'yearly'
           "repeat" : String,
           // badge application icon with this number (iOS only)
           "badge" : Number,
           // if set, the default system sound is played
           "sound" : "SYSTEM_DEFAULT",
           // if set to "SYSTEM_DEFAULT", the default vibration is
           // enabled upon an incoming notification (Android only)
           "vibration" : String,
           // custom JSON data to be passed through the notification
           "payload" : Object,
        }


/**
* 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 What You May Need to Know About Local Notification Options and the Application Behavior.

- `cancel`, defined as follows:

```
/**
* Cancel a scheduled local notification
```

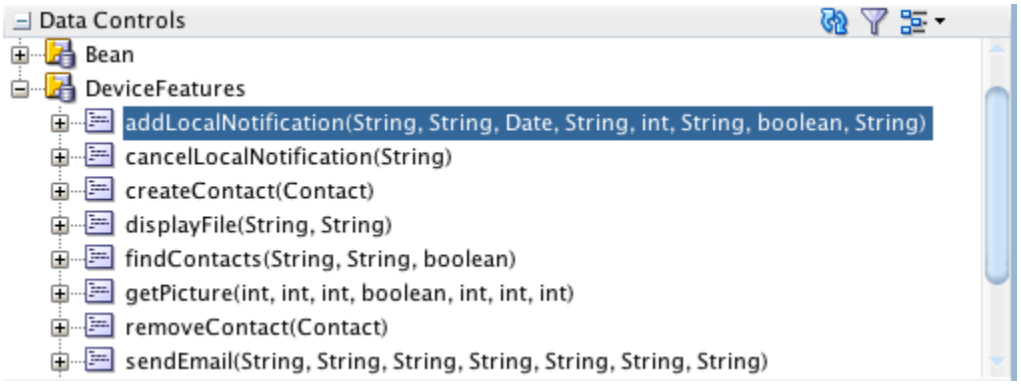
```
*
* @param {string} notificationId - id of the scheduled notification
*                                  that needs to be canceled
* @param {successCallback} scb - success callback
* @param {errorCallback} ecb - error callback
*/
adf.mf.api.localnotification.cancel(notificationId, scb, ecb);

{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 *JSDoc Reference for Oracle Mobile Application Framework*.

23.3.3 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 shown in Figure 23-3.

Figure 23-3 Methods of DeviceFeatures Data Control



For information about the DeviceFeatures data control, see Using the DeviceFeatures Data Control.

23.3.4 What You May Need to Know About Local Notification Options and the Application Behavior

The local notification behavior of a MAF application is determined by notifications settings. Review the tabulated local notification options.

Table 23-2 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 23-2 Local Notification Options

Option	Value	Behavior on iOS	Behavior on Android
`title`	Either: • `null` • not specified	The application name is displayed as the notification title.	The notification title in the notification center appears blank.
`alert`	Either: • `null` • not specified	If the notification has other properties specified, such as `badge` or `sound`, the notification is delivered to the operating system so that it plays a sound or updates the application icon badge, but the notification is not displayed in the notification center. If the application is running in the foreground at the time of the notification delivery, the notification is delivered 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.
`badge`	Either: • `null` • not specified • negative number	The notification does not badge the application icon. Any existing badge is maintained.	NA [1]
`badge`	0	Any existing badge is removed from the application icon.	NA [2]

Table 23-2 (Cont.) Local Notification Options

Option	Value	Behavior on iOS	Behavior on Android
`sound`	Other than `SYSTEM_DEFAULT_SOUND`	An error message is displayed. The notification does not play sound.	An error message is displayed. The notification does not play sound.
`sound`	Not specified	The notification does not play sound.	The notification does not play sound.
`vibration`	Other than `SYSTEM_DEFAULT_VIBRATION`	NA [3]	An error message is displayed. The notification does not trigger vibration of a mobile device.
`vibration`	Not specified	NA [4]	The notification does not trigger vibration of a mobile device.

1 There is no concept of application badging. The setting is ignored.

2 There is no concept of application badging. The setting is ignored.

3 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.

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.

23.4 Determining Application State When MAF Triggers a Notification Event

MAF provides the `EventListener` interface that you implement to listen for notification events. The `onMessage` method from this implementation includes an event parameter from where you can retrieve information about the MAF application state when it receives the event by using the `Event.getApplicationlicationState()` method.

The following example, taken from the `LocalNotificationDemo` MAF sample application, shows how you can get the notification event payload and determine the state of the MAF application when the notification event arrives.

```
NativeLocalNotificationListener implements EventListener {
...

    public void onMessage(Event event) {
    ...
          //MAF application state at the time of this notification
        String appState = stringifyAppState(event.getApplicationState());
        String payload = event.getPayload();
        ...

        HashMap<String, Object> payloadMap =
((NativeLocalNotificationEvent)event).getPayloadObject();
        ...
        JSONObject jsonPayload = (JSONObject)payloadMap.get("payload");
        ...
```

```
        AdfmfJavaUtilities.setELValue("#{applicationScope.notificationAppState}",
appState);
        AdfmfJavaUtilities.setELValue("#{applicationScope.notificationPayload}",
jsonPayload != null ? jsonPayload.toString() : "");
        AdfmfContainerUtilities.gotoFeature("Notification");
    }
    ...
}
```

The possible return values from `getApplicationState()` are:

- `APPLICATION_STATE_UNKNOWN (0)` if your MAF application was resident in memory when the notification arrived, but it could not be determined if the MAF application was running in the foreground or background. This case applies only to the iOS platform and the event payload will contain an additional `foreground` attribute that indicates the correct application state, set to either `1` for foreground or `0` for background. The format of the event payload is similar to:

  ```
  {"alert":"My message","foreground":1}
  ```

- `APPLICATION_STATE_NOT_RUNNING (1)` if your MAF application was not resident in memory and the operating system has launched the MAF application to handle the event.

- `APPLICATION_STATE_BACKGROUND (2)` if your MAF application was resident in memory when the notification arrived and running in the background.

- `APPLICATION_STATE_FOREGROUND (3)` if your MAF application was resident in memory when the notification arrived and running in the foreground.

Use the `NativeLocalNotificationEvent` class, shown in the above example, that extends `oracle.adfmf.framework.event.Event` to manage local events. MAF provides the `NativePushEvent` class that also extends `oracle.adfmf.framework.event.Event` to manage native push events.

Add your listener in the `start()` method of the MAF application lifecycle listener, as demonstrated by the following example from the `LocalNotificationDemo` MAF sample application, to receive events related to local notifications.

```
public class LifeCycleListenerImpl implements LifeCycleListener
{
...
  public void start()
  {
    // Listen for local notifications
    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* and MAF Sample Applications.

24

Displaying Error Messages in MAF Applications

This chapter describes how to use the `AdfException` class to invoke errors and how to localize error messages.

This chapter includes the following sections:

- Introduction to Error Handling in MAF Applications
- Displaying Error Messages and Stopping Background Threads
- Localizing Error Messages

24.1 Introduction to Error Handling in MAF Applications

MAF can manage errors that occur in applications and display messages to end users in response to these errors.

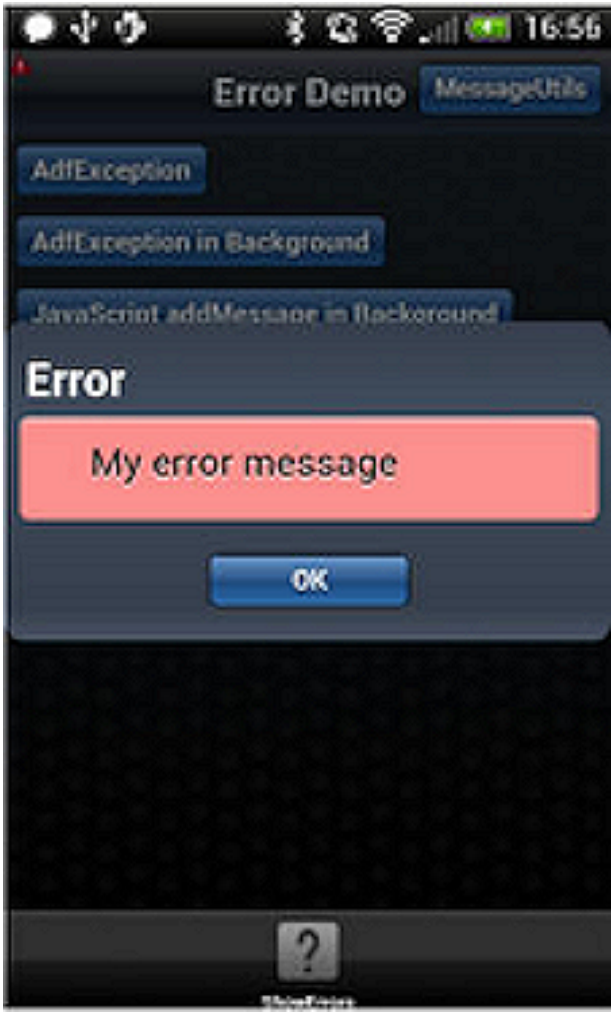
Errors arising from mobile applications might be unexpected, such as a failed connection to a remote server. Other errors, such as a violation of an application business rule, may be expected. Errors or exceptions might occur in the primary request thread or in a secondary thread that runs a background task. If the application supports multiple languages, then it must display the error message in the user's language.

To enable a MAF 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 as expected. A popup message, similar to the one shown in Figure 24-1 displays within the application and shows the severity and provides explanatory text.

```
throw new AdfException("My error message", AdfException.ERROR);
```

Figure 24-1 An Error Message



Note:

Similar error messages display within an application when the exception is thrown within a managed bean or a data control bean.

24.2 Displaying Error Messages and Stopping Background Threads

You can enable an application to stop a thread and display an error message by using MAF APIs.

The `MessageUtils` class, illustrated in the following example, 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 How Applications Display Error Message for Background Thread Exceptions. See also 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.addMessage",
                                 new Object[] {severity,
                                 null,
                                 null});
         if (AdfException.ERROR.equals(severity)) {
            // need to throw an exception 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,
                              message,
                              null,
                              null });
   }

}
```

24.2.1 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 following example 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.getFeatureName(),
                              "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 preceding example, 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 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();
```

24.3 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 following example, 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 following example.

```
ResourceBundle bundle = BundleFactory.getBundle(XLF_BUNDLE_NAME);
String message = Utility.getResourceString(bundle, "MY_ERROR_MESSAGE",null);
throw new AdfException(message,AdfException.ERROR);
```

The following 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 });
```

25

Deploying MAF Applications

This chapter describes how to deploy MAF applications for testing and for publishing.

This chapter includes the following sections:

- Introduction to Deployment of MAF Applications
- Working with Deployment Profiles
- Deploying a MAF Application to the Android Platform
- Deploying an iOS Application
- Deploying a MAF Application to the Universal Windows Platform
- Overview of MAF Quick Deployment of Applications
- Deploying Feature Archive Files (FARs)
- Creating a Mobile Application Archive File
- Creating a New Application from an Application Archive
- Deploying MAF Applications from the Command Line

25.1 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. MAF provides ready-to-use deployment profiles for each platform to facilitate deployment of your MAF application.

Use the ready-to-use deployment profile or create your own 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 profile 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 controller 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 profile (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.

We recommend that you invoke JDeveloper's **Build** > **Clean All** command in a number of scenarios prior to deploying your application so that JDeveloper removes artifacts and settings left over from previous builds in order to begin a fresh build process. These scenarios are:

- Changes to deployment profile settings, such as build mode or application name.
- Adding or removing a Cordova plugin.
- Changes to preferences for the Android platform, such as Gradle proxy settings, Android SDK location, or signing credentials.

25.2 Working with Deployment Profiles

For MAF application deployment, you can either create a ready-to-use deployment profile or use a customized deployment profile. A deployment profile defines how an application is packaged into the archive that will be deployed to the platform-specific device or virtual device.

You deploy a MAF application to the platform that you want your application to run on using a deployment profile. MAF provides a ready-to-use deployment profile for each platform that it supports (Android, iOS, Windows) or you can create a customized deployment profile. A deployment profile 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 profile does the following:

- Specifies the format and contents of the archive.
- Lists the source files, deployment descriptors, and other auxiliary files that will be packaged into the archive file.
- Describes the type and name of the archive file to be created.
- Highlights dependency information, platform-specific instructions, and other information.

In addition to the platform-specific deployment profiles, MAF also provides a deployment profile that 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. See Creating a Mobile Application Archive File.

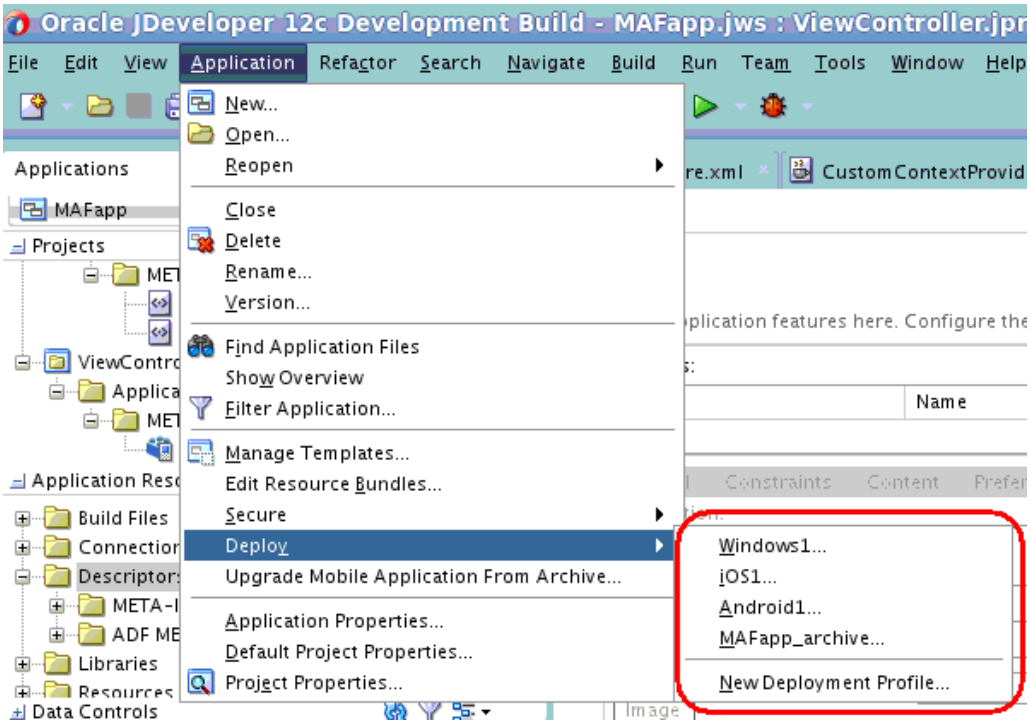
25.2.1 About Automatically Generated Deployment Profiles

When an application is created, MAF generates a set of deployment profiles with default settings and image files. Use the **Deploy** option from the Application menu to use a profile to deploy the application.

MAF generates deployment profiles that are seeded with default settings and image files when you create an application. Provided that you have configured the

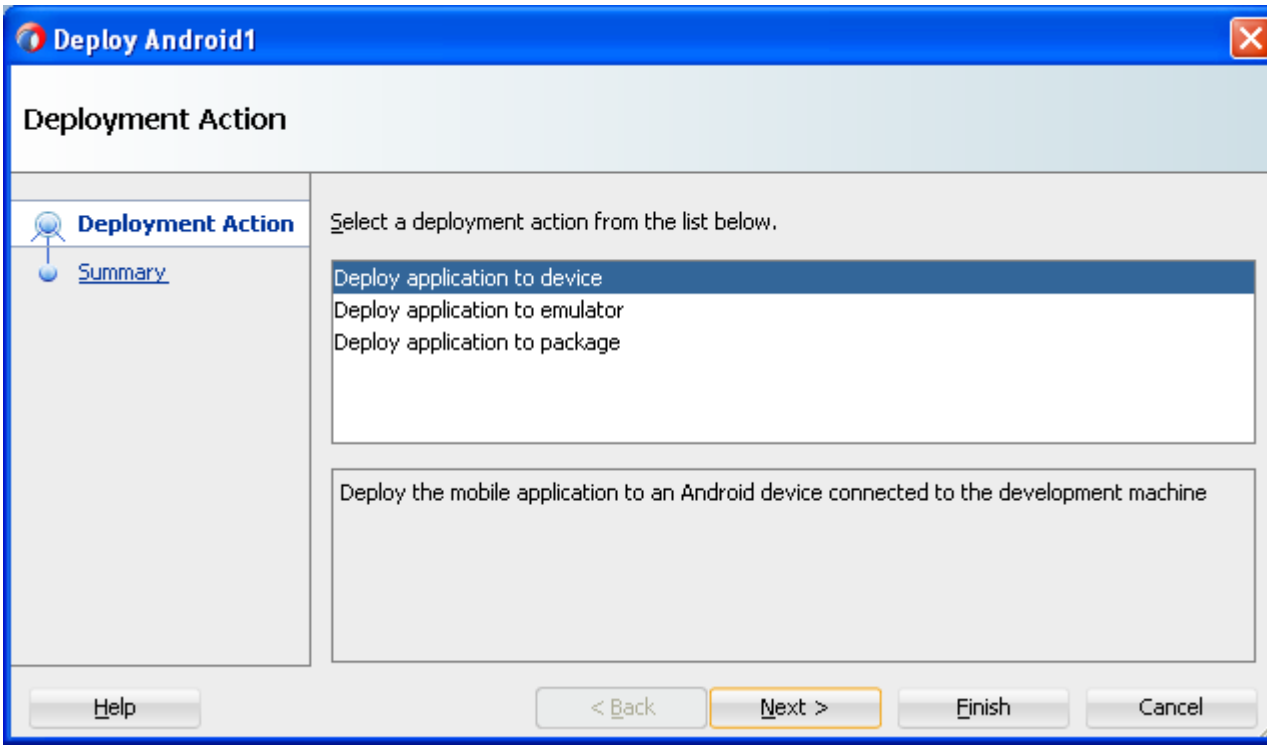
environment correctly, you can use these profiles to deploy a MAF application immediately after creating it by choosing **Application** and then **Deploy**, as shown in Figure 25-1.

Figure 25-1 Default Deployment Profiles



Using the **Deployment Action** page, shown in Figure 25-2, you then select the appropriate deployment target.

Figure 25-2 Selecting a Deployment Target

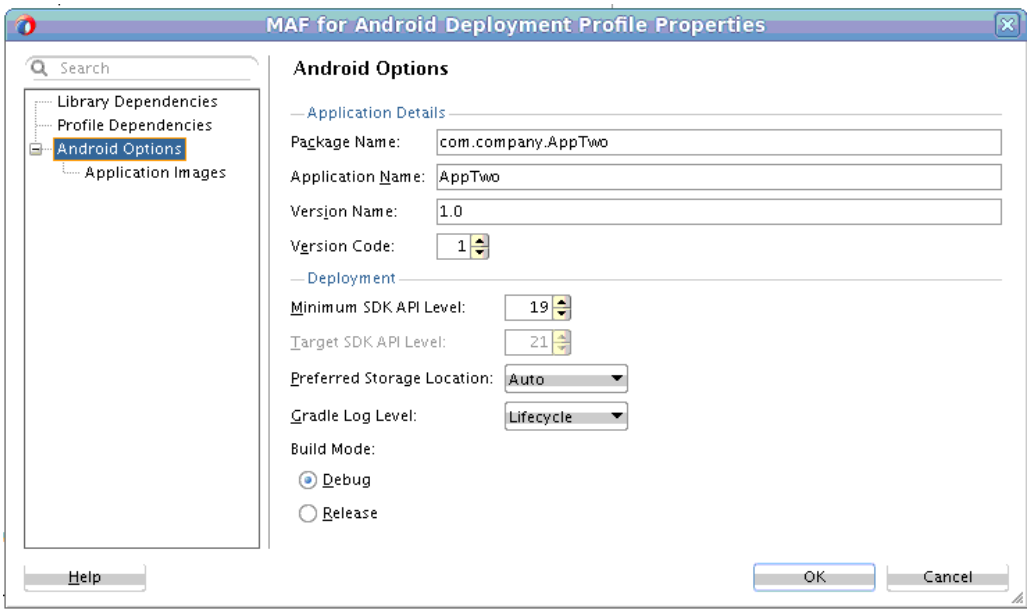


Each deployment profile has distinct environment set up and configuration requirements. See:

- Deploying an Android Application
- Deploying an iOS Application
- Deploying a MAF Application to the Universal Windows Platform

You can accept the default values used for these profiles, or edit them by selecting the profile from the **Deployment** page of the **Application Properties** dialog and then clicking **Edit**. Figure 25-3 illustrates the **Android Options** page for a default Android application profile.

Figure 25-3 Editing a Default Deployment Profile

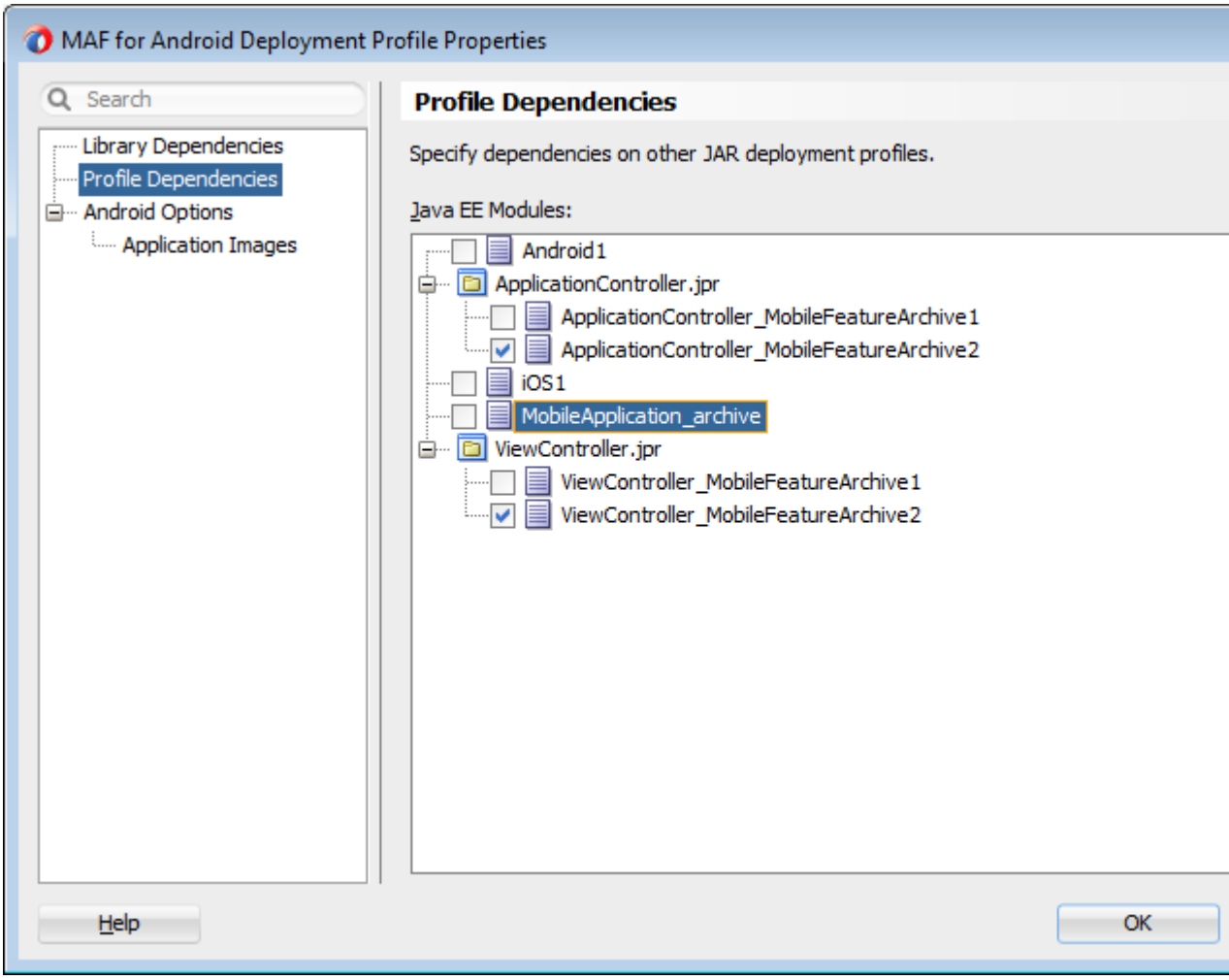


MAF packages the application and view controller projects as separate Feature Archive (FAR) files. These JAR files of MAF files are used as resources for other applications and are described in Deploying Feature Archive Files (FARs). Because MAF creates these FAR files as dependencies to the MAF application profile, you can include or exclude them using the **Profile Dependencies** page of the **Application Properties** dialog, shown in Figure 25-4.

Note:

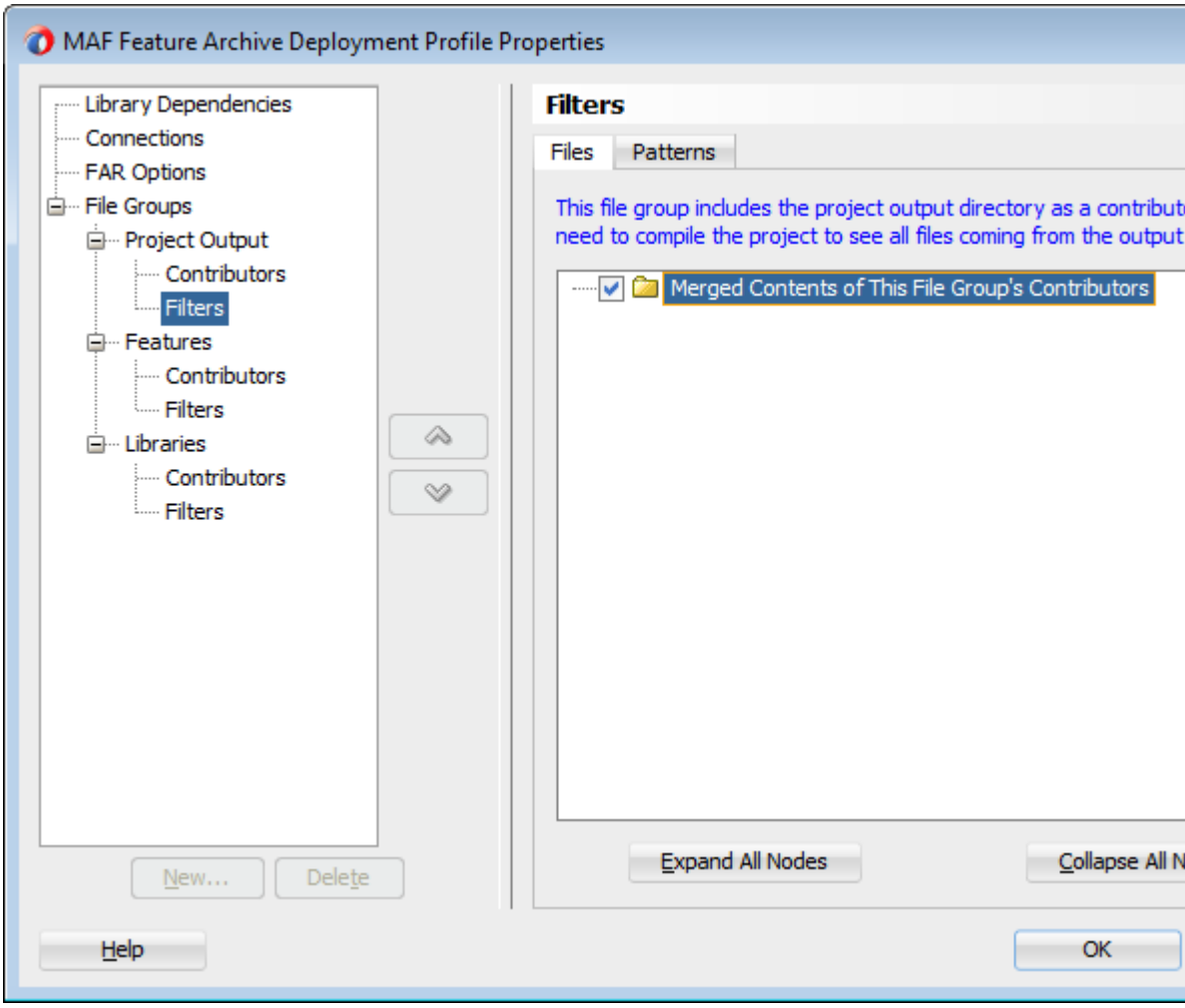
The application controller project must contain a single FAR profile dependency; otherwise, the deployment will fail.

Figure 25-4 Editing FAR Contents from MAF Projects



Using the File Groups-related pages of the **Project Properties** dialog, you can customize the contents of the view controller FAR file, as shown in Figure 25-5. For more information on the **Project Properties** dialog, see the Oracle JDeveloper online help and also the Configuring Deployment Profiles in *Developing Applications with Oracle JDeveloper*.

Figure 25-5 Editing the FAR of the View Controller Project



For more information on editing deployment profiles using the **Application Properties** dialog pages, see Viewing and Changing Deployment Profile Properties in *Developing Applications with Oracle JDeveloper* and the Oracle JDeveloper online help for the **Application Properties** and **Project Properties** dialogs.

25.2.2 How to Create a Deployment Profile

After designating the SDKs for the target platforms, follow the steps in the task to use Create Deployment Profile to create deployment profiles. Although you can create multiple deployment profiles, an application needs only one application-level deployment profile for deployment.

MAF creates a set of deployment profiles when you create a mobile application. You can deploy an application using these profiles, edit them, or construct new ones using the MAF-specific deployment profile pages. The **Create Deployment Profile** wizard, shown in Figure 25-6, enables you to create a default deployment profile from these pages. You can create as many deployment profiles as needed. For more information on these standard deployment profile pages, click **Help** to see the JDeveloper online help.

Note:

MAF application deployment only requires the creation of an application-level deployment profile; you do not have to create a view controller-level deployment profile.

Before you begin:

To enable JDeveloper to deploy mobile applications, you must designate the SDKs for the target platforms as described in Configuring the Development Environment for Platforms and Form Factors in *Installing Oracle Mobile Application Framework*.

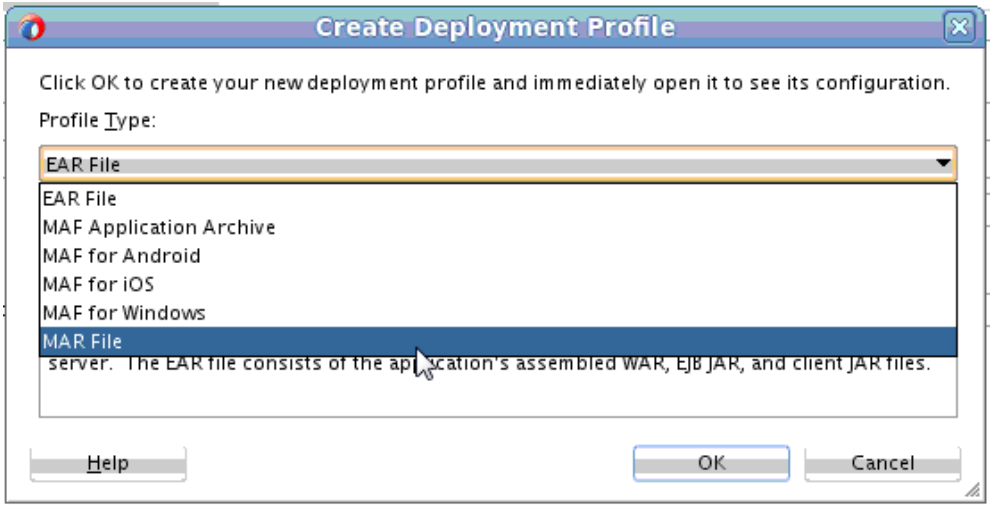
Tip:

For iOS deployments, run iTunes and the iOS Simulator at least once before you configure their directory locations.

To create a deployment profile:

1. Select **Application** and then **Deploy**.
2. Select **New Deployment Profile**.
3. Select the profile for the platform that you want to target from the **Profile Type** drop-down list, as shown in Figure 25-6.
4. Accept the default name for the profile or enter a new one. Click **OK**.
5. If needed, use the **Options** and **Application Images** pages as required for the applications and then click **OK**.

Figure 25-6 The Create Deployment Profile Wizard



25.2.3 What Happens When You Create a Deployment Profile

After you complete the Create Deployment Profile wizard, JDeveloper creates a deployment profile and opens the Deployment Profile Properties editor. When an application is deployed, JDeveloper creates a deployment directory and the related subdirectory in addition to FAR files for the default projects.

After you complete the wizard, JDeveloper creates a deployment profile and opens the **Deployment Profile Properties** editor.

Table 25-1 lists the MAF-specific pages in the **Deployment Profile Properties** editor, shown in Table 25-1.

Table 25-1 MAF-Specific Deployment Profile Pages

Page	Function
iOS Options	Enables you to modify the settings for an application to be deployed on an iOS-powered device or iOS simulator.
Android Options	Enables you to modify the settings for an application deployed to an Android-powered device or Android emulator.
Windows Options	Enables you to modify the settings for an application that you deploy to the Universal Windows Platform.
Application Images	Enables you to assign custom icons to an application by adding the appropriate graphics file.
Device Orientations	Enables you to restrict the display of an application to certain device orientations. This page is used only for iOS deployment profiles.

Note:

Deployment depends on the needs of your application. You can deploy an application using the default values seeded in the pages listed in Table 25-1.

When you deploy an application, JDeveloper creates a deployment directory and related subdirectory. It also creates Feature Archive files (FARs) for the view controller projects (which must have different names) and application controller project. In addition to these two FARs, JDeveloper creates copies of any FARs that were imported into the project. Changes to the compilation profiles require the removal of the deployment directory. You can remove this directory, as well as the deployment directory within the view controller project that contains the FAR, by selecting **Build** and then **Clean All**.

25.3 Deploying a MAF Application to the Android Platform

MAF applications can be deployed to an Android device, an emulator, or to a deployment package that you or end users then use to install the application on an Android device or emulator.

MAF provides two build modes for an application that you deploy to the Android platform. Use debug mode to test and debug your application as you go through the development cycle. Use release mode to deploy an application that is release ready and can be published to end users through an application marketplace, such as Google Play.

MAF provides a ready-to-use deployment profile (`Android1`) that deploys your MAF application. You can use this ready-to-use deployment profile or create one or more other deployment profiles to deploy your MAF application to the Android platform. Using a deployment profile, you can deploy your MAF application in debug mode or release mode.

Creating a new deployment profile or editing an existing profile, you can:

- Specify additional library and profile dependencies for your application
- Choose between debug or release build modes
- Specify options such as the application name and version
- Specify the images that the application uses to render its logo in the splash screen and the icon that appears on the user's device.
- Specify the preferred storage location for the MAF application when installed on the Android device.
- Choose the logging level for the Gradle build tool that MAF uses to build the MAF application. For information about these log levels, see Gradle's documentation.
- Specify the minimum SDK API level that determines the minimum version of Android your MAF application can run on.

For more information about how to edit or create a deployment profile, see Working with Deployment Profiles. Click **Help** in the deployment profile dialogs to view information about the respective tasks that you can perform in each dialog.

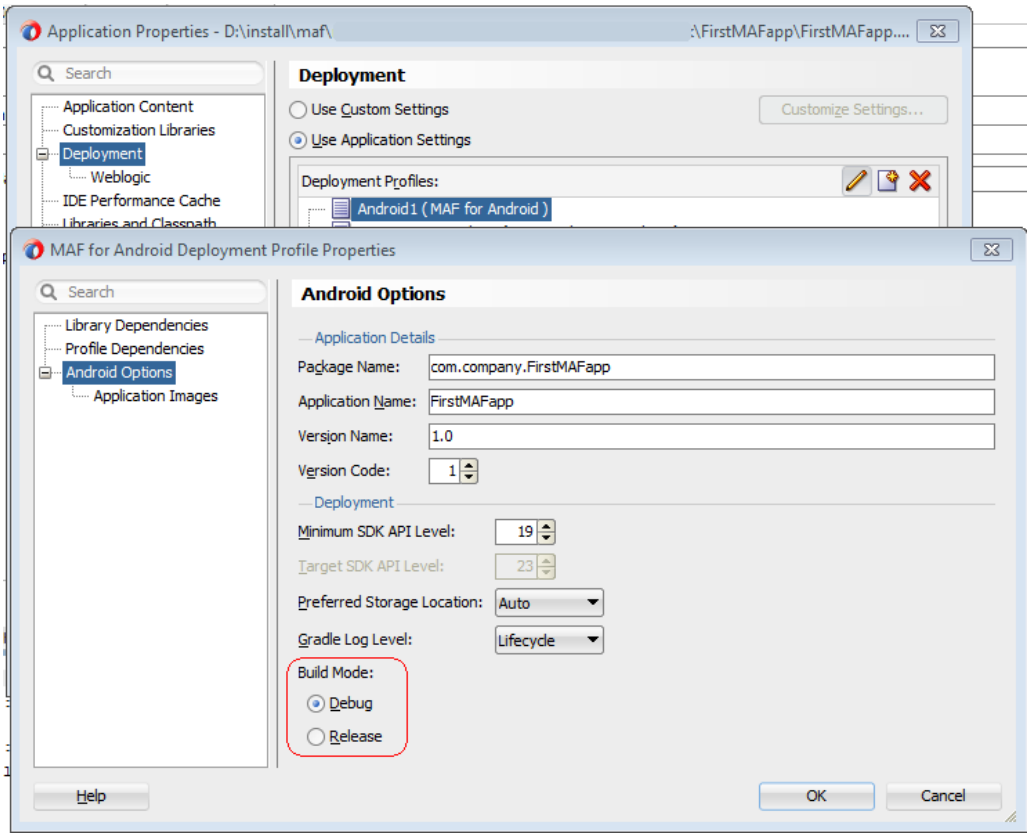
Before you deploy a MAF application using a deployment profile, you download the Android SDK and reference it from the Android Platform dialog in MAF Preferences, as described in Configuring the Development Environment for Target Platforms and How to Install the Android SDK in *Installing Oracle Mobile Application Framework*.

MAF uses Gradle to build and deploy MAF applications to the Android platform. MAF downloads and installs the Gradle build tool the first time that you deploy a MAF application. If your development machine is located behind a corporate firewall, configure the Gradle proxy settings so that MAF can successfully download, install, and configure Gradle. See How to Configure Gradle Proxy Settings.

After you finish development of your MAF application, you can publish it to an application marketplace, such as Google Play. Before you publish the application, make sure that the build mode in the deployment profile you use to deploy the application is in release mode, as shown by Figure 25-7. You access the Android Options dialog from JDeveloper's **Application** > **Application Properties** > **Deployment** menu. To deploy in release mode, you also need to sign the application, as described in How to Sign a MAF Application that You Deploy to the Android Platform.

For information about publishing an application to Google Play, see http://developer.android.com/tools/publishing/publishing_overview.html.

Figure 25-7 Release Mode in Deployment Profile Properties



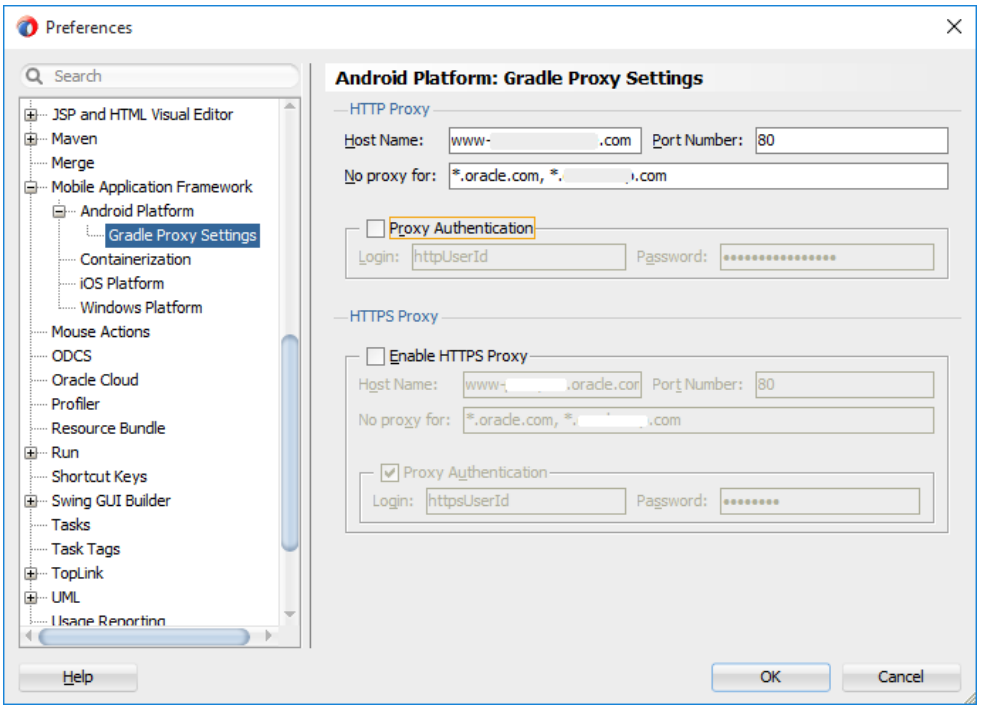
25.3.1 How to Configure Gradle Proxy Settings

If your computer is in a corporate network, configure proxy settings in the Gradle Proxy Settings dialog so that MAF can download and install the Gradle tools required to build MAF applications that you deploy to the Android platform.

To configure Gradle proxy settings:

1. In JDeveloper, click **Tools**, **Preferences**, **Mobile Application Framework**, **Android Platform**, and **Gradle Proxy Settings** to open the dialog shown in Figure 25-8.

Figure 25-8 Gradle Proxy Settings



2. As necessary, complete the fields in the HTTP Proxy and HTTPS Proxy sections of the dialog:

   - **Host Name:** Enter the URI of the proxy server.
   - **Port Number**: Enter the port number that the proxy server uses.
   - **No proxy for**: Enter a `|` delimited list of URIs to ignore. For example, `*.nonproxyrepos.com|*.oracle.com|localhost`.
   - **Proxy Authentication:** Select if the proxy server requires authentication credentials and enter these credentials in the **Login** and **Password** fields.
   - **Enable HTTPS Proxy**: Select if you need to specify details (host name, port number, and so on) for a HTTPS Proxy server. This makes the subsequent input fields writable.

3. Click **OK**.

25.3.2 How to Sign a MAF Application that You Deploy to the Android Platform

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 MAF application is a two-step process: within the Android Platform page of the MAF preferences, you first define debug and release properties for a key that is used to sign Android applications. You only need to configure the debug and release signing properties once. After you define these options, you configure the deployment profile to designate if the application should be deployed in the debug or release mode.

If no keystore file exists, you can create one using the keytool utility, as illustrated in the following example.

```
keytool -genkeypair
        -v
        -keystore c:\jdeveloper\mywork\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 (`http://download.oracle.com/javase/index.html`) for information on how to use the keytool utility.

To configure the key options for the debug mode:

1. Click **Tools**, then **Preferences**, and then **Mobile Application Framework**.
2. Select **Android Platform** and, in the Signing Credentials section, enter a password used by the deployment to create a keystore file and key needed for a debug deployment in the **Key** and **Keystore Password** field. This password, which generates a keystore and keyfile for deployment to an Android-powered device or emulator, can be any value, but must be at least six characters long. The default password is *Android*.

To configure the key options for the release mode:

1. Click **Tools**, **Preferences**, and **Mobile Application Framework**.
2. Select **Android Platform**, then the Release tab, and then define the following:
   - **Keystore Location**—Enter, or browse to and retrieve, the directory of the keystore containing the private key used for signing the application for distribution.
   - **Keystore Password**—Enter the password for the keystore. This password allows access to the physical file.
   - **Key Alias**—Enter an alias for the key. This is the value set for the keytool's `-alias` argument. Only the first eight characters of the alias are used.
   - **Key Password**—Enter the password for the key. This password allows access to the key (identified by the alias) within the keystore.

     Tip:

     Enter the password and key password requested by the keytool utility before it generates the keystore.
3. Click **OK**.

25.3.3 What You May Need to Know About Credential Storage

MAF stores passwords for the key and keystore in the file-based credential store, `cwallet.sso`. This file, which manages credential storage and retrieval, is located within the `o.maf` folder in the user's JDeveloper system folder.

For example, in a Windows environment, the `cwallet.sso` file is located at `C:\Users\jsmith\AppData\Roaming\JDeveloper\system12.1.3\o.maf`.

See About Oracle Wallet in *Administering Oracle Fusion Middleware*.

Note:

MAF stores the key and keystore credentials in a file called `product-preferences.xml`. MAF migrates these credentials to the `cwallet.sso` file if you preserve the preference settings by clicking **Yes** in the Confirm Import Preferences dialog during the installation process of the current version of JDeveloper and MAF. However, the `cwallet.sso` file is not migrated to other installations of the current version of Oracle JDeveloper with MAF. If you reinstall (or create a separate installation), you must either copy the `cwallet.sso` file to the `o.maf` folder or reconfigure the release mode credentials in the Platforms preferences page.

25.3.4 How to Deploy a MAF Application to the Android Platform

Deploy the application to the Android platform using a MAF for Android deployment profile.

To deploy a MAF application to the Android platform:

1. Select **Applications**, then **Deploy**, then select an Android deployment profile.
2. Select the appropriate deployment option from the list in the dialog that appears:
   - **Deploy application to device**

     Before you select this option that deploys directly to an Android-powered device, connect the device to the development computer that hosts JDeveloper, set the device to developer mode, and turn on USB debugging. For more information about these tasks, see How to Set Up an Android-Powered Device in *Installing Oracle Mobile Application Framework*.
   - **Deploy application to emulator**

     Before you select this option that deploys directly to an Android emulator, set up and start the Android emulator. For more information about this task, see How to Set Up an Android Emulator in *Installing Oracle Mobile Application Framework*.
   - Deploy application to package

     Deploys the MAF application to an .APK file that you can then distribute for installation on an Android device or emulator.
3. Review the Summary page and click **Finish**.

25.3.5 What Happens When You Deploy a MAF Application to the Android Platform

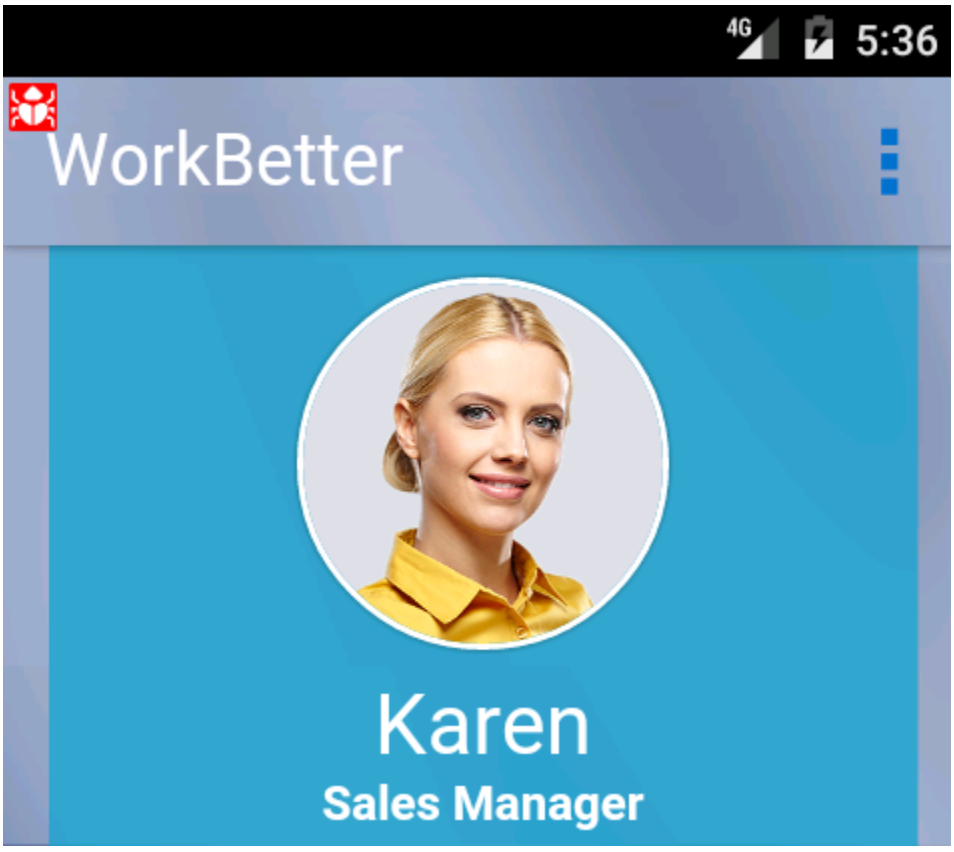
MAF generates an `.APK` file that it uses to install the MAF application on a device or emulator if either of these deployment options were chosen. If you chose the deploy application to package option, navigate to the directory where MAF generated the `.APK` file to retrieve it.

MAF includes the following in the .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 files

For deployment to device or emulator options, MAF installs and launches the MAF application. If you deployed the MAF application in debug mode, a red icon appears on each screen of the MAF application, as shown in the upper-left portion of Figure 25-9. MAF applications that you deploy in release mode do not render this icon.

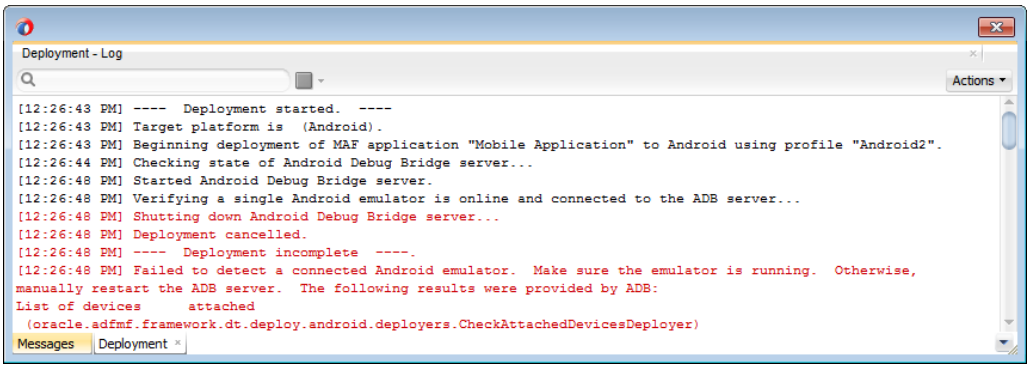
Figure 25-9 MAF Application Deployed in Debug Mode



MAF uses the Android Debug Bridge to connect to the device or emulator where you want to deploy the MAF application. If the deployment does not detect a device, it restarts the Android Debug Bridge server five times until it detects a device (if

deploying to a device) or emulator (if deploying to an Android emulator). If it detects neither, then it ends the deployment process, as shown in Figure 25-10.

Figure 25-10 Deployment Terminated



If you are using the Android Debug Bridge command line tool prior to deployment, then you must enter the same command again after the deployment has completed. For example, if you entered `adb logcat` to view logging information for an emulator or device prior to deployment, you would have to enter `adb logcat` again after the application has been deployed to resume the retrieval of the logging output. For more information about the Android Debug Bridge command line tool, which is located within (and executed from) the `platform-tools` directory of the Android SDK installation, refer to the Android Developers website (http://developer.android.com/tools/help/adb.html).

After you select a deployment action, JDeveloper creates a shortcut on the Deploy menu that enables you to easily redeploy the application using that same deployment action.

By default, MAF applications are installed on an Android-powered device's internal storage after they have been deployed from JDeveloper to a device, or downloaded from an application marketplace, such as Google Play. The following options, which are available from the **Preferred Storage Location** dropdown list, enable you to specify a preferred storage location for the MAF application.

- **Internal**—Forces the MAF application to be installed on the device's internal storage.
- **External**—Allows the application to be installed on the device's SD card. However, if the Android system determines that the application cannot be installed on the SD card (for example, no SD card has been mounted, or the SD card exists but has insufficient space), then it installs the application on the device's internal storage instead. The mobile device user can move the application between internal and external storage using the system settings.
- **Auto**—Specifies that the application may be installed on the device's external or internal storage. The mobile device user can move the application between internal and external storage using the system settings.

Selecting the **External** or **Auto** options enables the deployment framework to update the `<manifest>` element in the `AndroidManifest.xml` file with an `android:installLocation` attribute and a value of `"preferExternal"` or `"auto"`. Populating the `AndroidManifest.xml` file with this attribute enables MAF applications to be stored on an external SD card or internal storage. For more information, see the "App Install Location" chapter in Data Storage Guide, available

from the Android Developers website (http://developer.android.com/guide/topics/data/install-location.html) or from the Android SDK documentation.

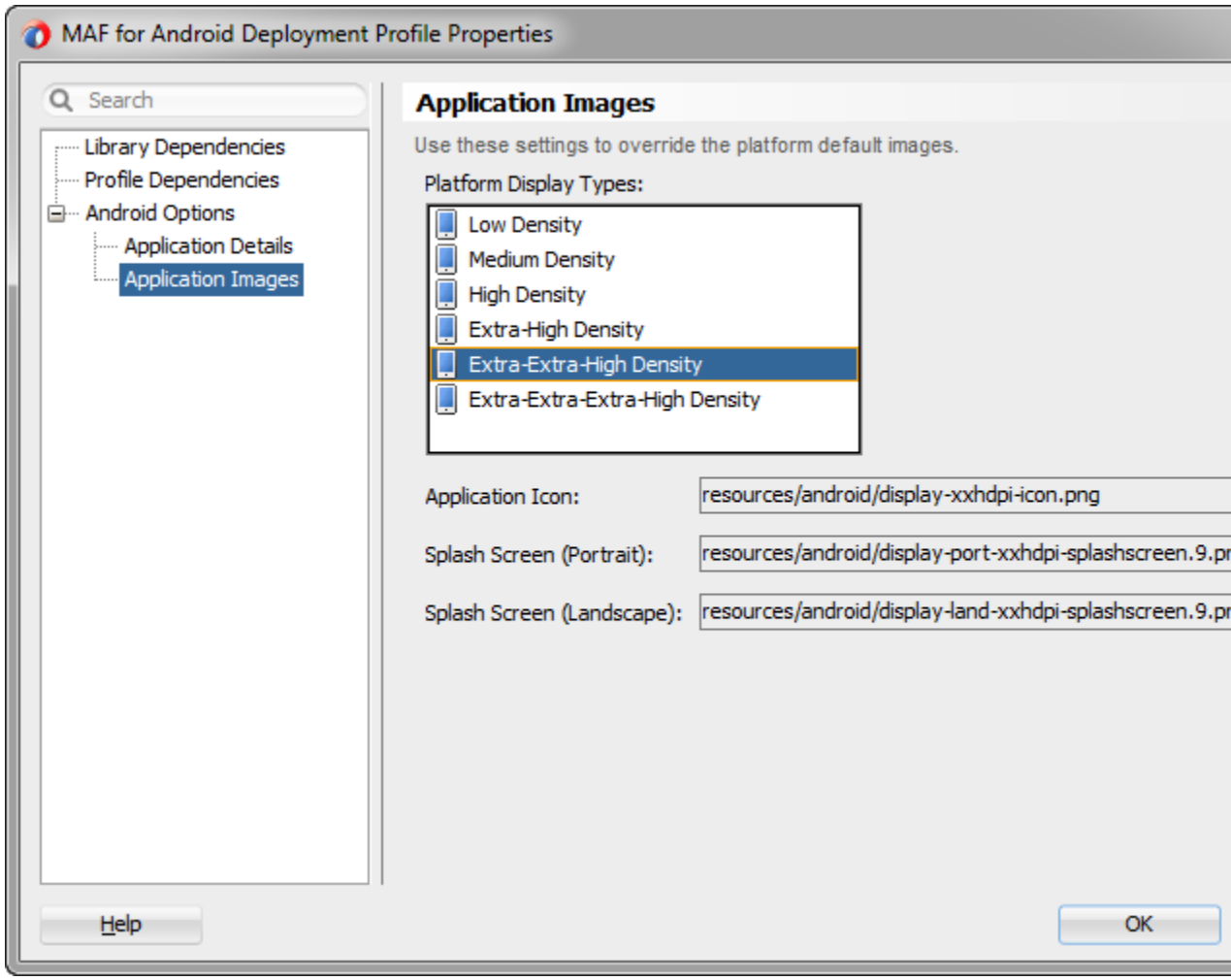
25.3.6 How to Add a Custom Image to an Android Application

Enabling MAF application icons to display properly on Android-powered devices of different sizes and resolutions requires low, medium, high, extra-high, extra-extra-high density, and extra-extra-extra-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 25-11, 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. MAF provides 9-patch images for the default Android splash screens. The 9-patch images indicate which areas of the image may be stretched, and which may not. These images can be stretched to fit any size while maintaining the integrity of designated portions within the image (such as the logo and copyright notice in the default MAF splash screen images).

To create custom images, refer to the *Iconography* document, available from the Android Developers website (`http://developer.android.com/design/style/iconography.html`).

Figure 25-11 Setting Custom Images for an Android Application



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. See the *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 **Application Images**.
2. Use the **Browse** function to select the splash screen and icon image files from the project file. Figure 25-11 shows selecting images for application icons and portrait orientation splash screen images that applications use for displaying on devices with low, medium, high, extra-high, extra-extra-high density, and extra-extra-extra-high density displays.
3. Click **OK**.

25.3.7 What Happens When JDeveloper Deploys Images for Android Applications

During deployment, MAF enables JDeveloper to copy the images from their source location to a temporary deployment folder.

For the default images that ship with the MAF extension (located at *application workspace directory*\Application Resources\Resources\images), JDeveloper copies them from their seeded location to a deployment subdirectory of the view controller project (*application workspace*\ViewController\deploy). As shown in Table 25-2, 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, xhdpi, xxhdpi, and xxxhdpi) and orientation (port, land). Within these directories, JDeveloper renames each icon image file as adfmf_icon.png and each splash screen image as adfmf_loading.9.png or adfmf_loading.png (depending on whether 9-patch images are used).

Table 25-2 Deployment File Locations for Seeded Application Images

Source File (...\resource\Android)	Temporary Deployment File (...ViewController\deploy)
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-xxhdpi-icon.png	drawable-xxhdpi\adfmf_icon.png
display-xxxhdpi-icon.png	drawable-xxxhdpi\adfmf_icon.png
display-port-ldpi-splashscreen.9.png	drawable-port-ldpi\adfmf_loading.9.png
display-port-mdpi-splashscreen.9.png	drawable-port-mdpi\adfmf_loading.9.png
display-port-hdpi-splashscreen.9.png	drawable-port-hdpi\adfmf_loading.9.png
display-port-xhdpi-splashscreen.9.png	drawable-port-xhdpi\adfmf_loading.9.png
display-port-xxhdpi-splashscreen.9.png	drawable-port-xxhdpi\adfmf_loading.9.png
display-land-ldpi-splashscreen.9.png	drawable-land-ldpi\adfmf_loading.9.png
display-land-mdpi-splashscreen.9.png	drawable-land-mdpi\adfmf_loading.9.png
display-land-hdpi-splashscreen.9.png	drawable-land-hdpi\adfmf_loading.9.png
display-land-xhdpi-splashscreen.9.png	drawable-land-xhdpi\adfmf_loading.9.png
display-land-xxhdpi-splashscreen.9.png	drawable-land-xxhdpi\adfmf_loading.9.png

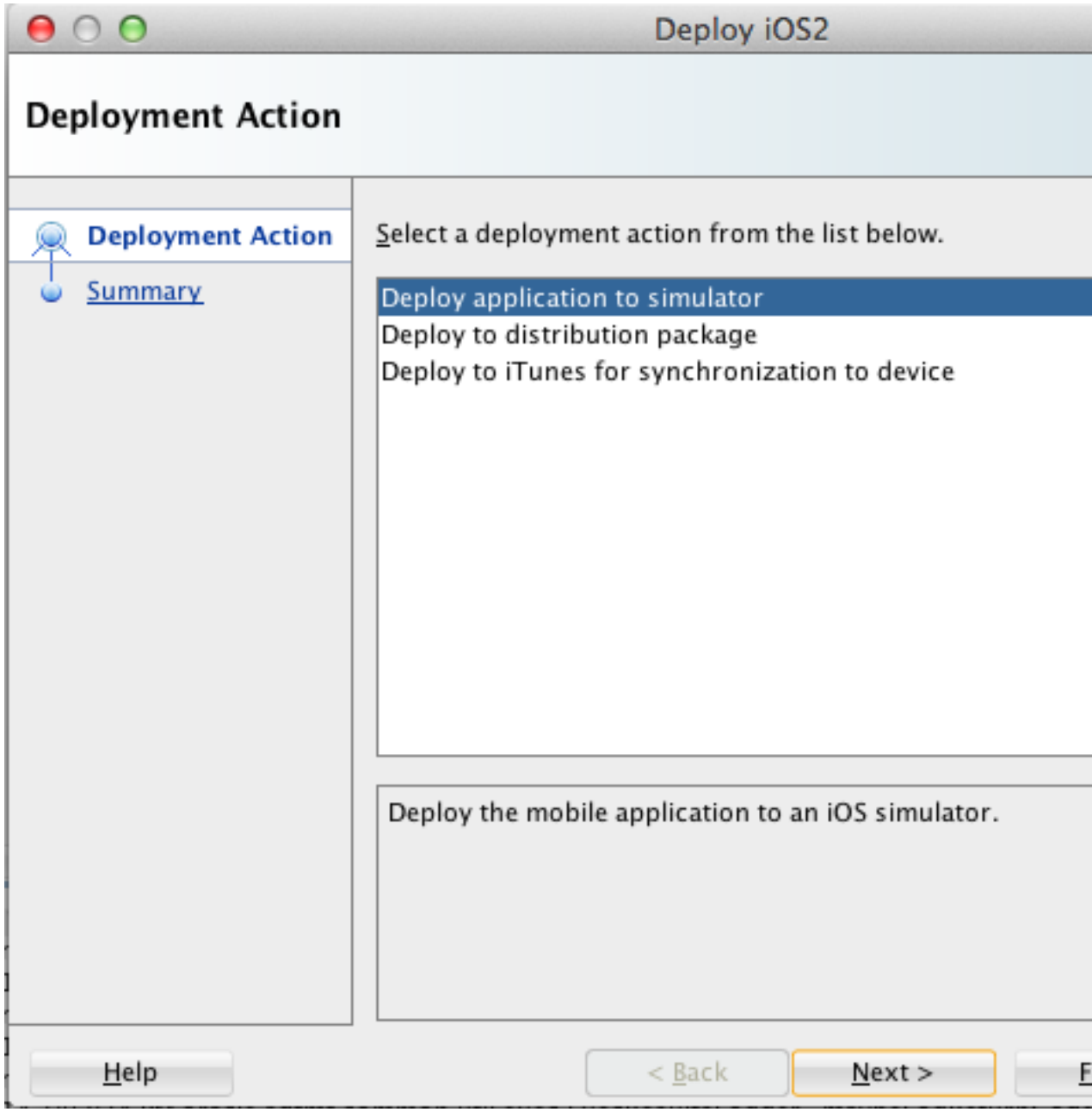
For custom images, JDeveloper copies the set of application icons from their specified location to the corresponding density and orientation subdirectory of the temporary deployment location.

25.4 Deploying an iOS Application

Deploy a MAF application from a Mac computer to an iOS simulator or to an iOS device by means of iTunes. Either use the **Deployment Action** option or manually deploy the application using the OJDeploy command line tool.

The **Deployment Action** dialog, shown in Figure 25-12, 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 How to Deploy an Application to an iOS-Powered Device, and How to Distribute an iOS Application to the App Store), see `https://developer.apple.com/programs/`.

Figure 25-12 The Deployment Action Dialog (for iOS Applications)



Tip:

As an alternative to the **Deployment Action** dialog, you can deploy a mobile application to the iOS platform manually using the OJDeploy command line tool as described in Deploying MAF Applications from the Command Line.

25.4.1 How to Create an iOS Deployment Profile

Use the Deployment Profiles Properties editor to create a deployment profile for the iOS platform. Ensure that prerequisites are met, and follow the steps in the task to create the profile by selecting **iOS Options** for deployment available under **Application Properties** in the Application menu.

For iOS, use the **Deployment Profiles Properties** editor to define the iOS application build configuration as well as the locations for the application icons.

Before you begin:

Download Xcode (which includes the Xcode IDE, performance analysis tools, the iOS simulator, and the Mac OS X and iOS SDKs) to the Apple computer that also runs JDeveloper.

Tip:

Refer to the Certification and Support Matrix on Oracle Technology Network (`http://www.oracle.com/technetwork/developer-tools/maf/documentation`) 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 JDeveloper.

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. See the Configuring the Development Environment for Platforms and Form Factors in *Installing Oracle Mobile Application Framework*.

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 Setting the Device Signing Options.

To create a deployment profile:

1. Select **Application**, **Application Properties**, and then **Deployment**.
2. On the **Deployment** page, double-click an iOS deployment profile.
3. Select **iOS Options**.
4. Accept the default values, or define the following:

   - **Application Bundle Id**—If needed, enter a bundle ID to use for this application that identifies the domain name of the company. The application 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 Registering an Application ID. See also Setting Display Properties for an Application Feature.

     Note:

     The application bundle ID cannot contain spaces.

     Because each application 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 application 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).

   - **Application Archive Name**—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 25-12. It creates an `.app` file when you select the **Deploy application to simulator** option. Otherwise, accept the default name. See How to Deploy an Application to an iOS-Powered Device and 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. See Setting Display Properties for an Application Feature.

   - **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.

   - **Simulator**—Select the hardware and iOS version of the simulator to which you are deploying the application. Available versions are displayed in the drop-down list along with the device ID. For more information, see the *iOS Simulator User Guide*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

   - **Family**—Select the family of iOS products on which the application is intended to run. The default option is for both iPad and iPhone.

25.4.1.1 Defining the iOS Build Options

Build options allow you to deploy an application in the debug or release mode. If a provisioning profile and a signed certificate are available, follow the steps in the task to use the iOS Options page to set build options for the iOS platform.

The iOS build options enable you to deploy an application with debug or release bits and libraries.

Before you begin:

Deployment of an iOS application (that is, an `.ipa` file) to an iOS-powered device requires a provisioning profile, which is a required component for installation, and also a signed certificate that identifies the developer and an application on a device. You must obtain these from the iOS Provisioning portal as described in What You May Need to Know About Deploying an Application to an iOS-Powered Device. In addition, you must enter the location for a provisioning profile and the name of the certificate in the **iOS Platform preference** page, as described in Setting the Device Signing Options.

To set the build options:

1. Select **Application**, then **Application Properties**, and then **Deployment**.
2. On the **Deployment** page, double-click an iOS deployment profile.
3. Select **iOS Options**.
4. Select from the following build options.

    - **Push Notification Environment** – Select `Production` if your deployed application is to use production Apple Push Notification service (APNs) servers or `Development` if it is to use development APNs servers. MAF ignores the value of Push Notification Environment if you do not select the PushPlugin plugin to enable push notifications, as described in Enabling Push Notifications.
    - **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.
    - **Debug**—Select this option for development builds. Designating a debug build results in the inclusion of debugging symbols. See also How to Debug on the iOS Platform and How to Enable Debugging of Java Code and JavaScript.
    - **Release**—Select to compile the build with release bits and libraries.

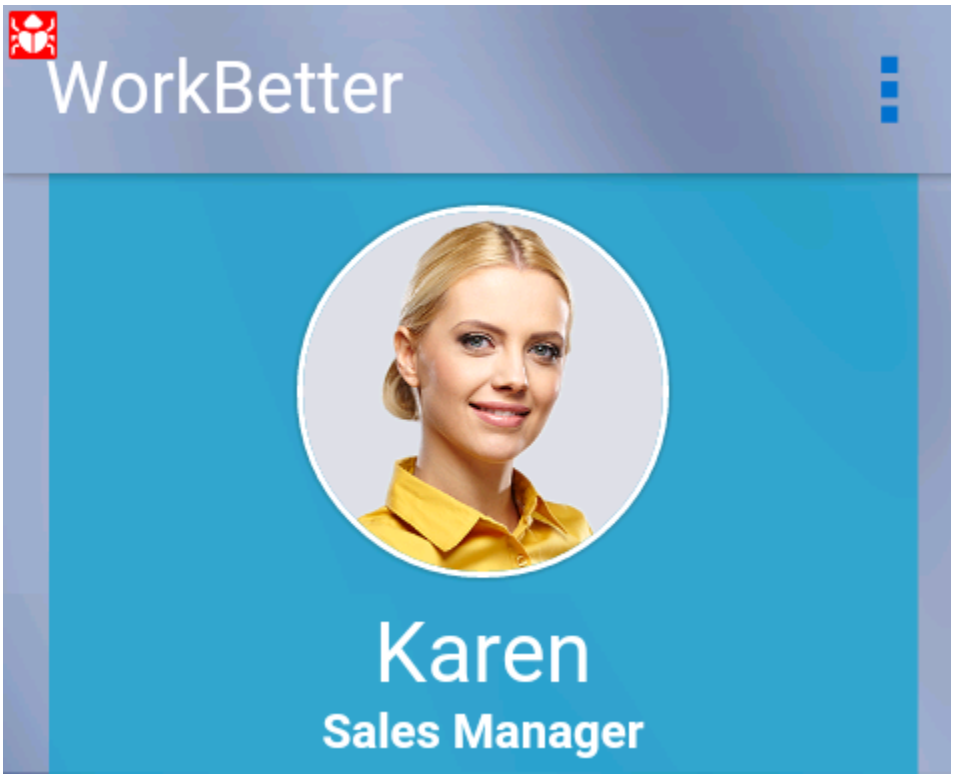
        Tip:

        Use the release mode, not the debug mode, to test application performance.

    - **Additional Build Arguments**—Specify additional arguments that Xcode can use when it builds the MAF application.

When you deploy the MAF application in debug mode, a red icon appears on each screen of the MAF application, as shown in the upper-left portion of Figure 25-13. MAF applications that you deploy in release mode do not render this icon.

Figure 25-13 MAF Application Deployed in Debug Mode



25.4.1.2 Setting the Device Signing Options

Specify the location of the provisioning profile, the identifier of the development team, and the signing identity required for the deployment of an application to a distribution package or to iTunes for synchronization to a device.

The **iOS Platform preferences** page for iOS includes fields for the location of the provisioning profile on the development computer, the identifier of the team, and the name of the signing identity. You must define these parameters if you deploy an application to a distribution package or to iTunes for synchronization to a device. You use a signing identity to code sign your application. When Xcode requests your development certificate, the certificate and its public key is stored in the Member Center, and the signing identity (the certificate with its public and private key) is stored in your keychain. You will not be able to code sign without this private key.

Note:

Neither a signing identity nor a provisioning profile are required if you deploy a mobile application to an iOS simulator.

To set the signing options:

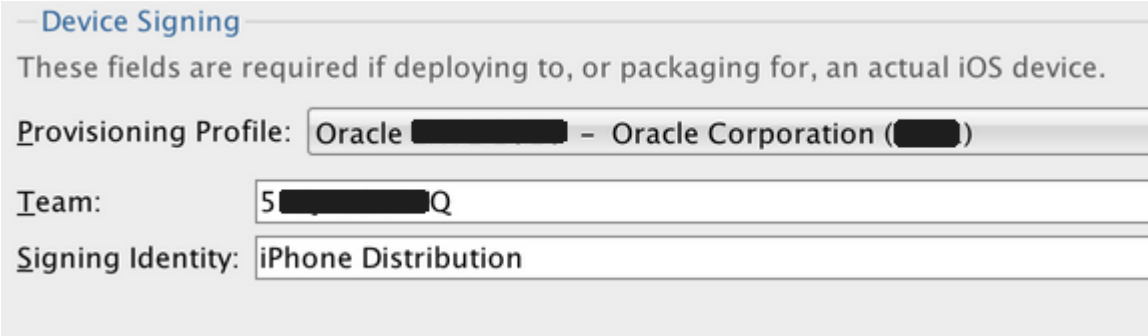
1. Select **JDeveloper**, **Preferences**, **Mobile Application Framework**, and then **iOS Platform** .
2. From the **Provisioning Profile** drop-down list, select the provisioning profile.

3. In the **Team** field, enter the identifier of your team. MAF automatically populates this input field with a value that it extracts from the provisioning profile that you select in Step 2.

4. In the **Signing Identity** field, enter the name of the developer or distribution certificate that identifies the originator of the code (such as a developer or a company). You can view the name of the certificate using the Keychain Access utility (accessed from the Applications folder). Copy the entire name from the Keychain Access utility. The name entered into this field may look similar to the following example.

```
iPhone Developer: John Smith (Oracle123)
```

Figure 25-14 The Device Signing Section of the iOS Platform Preferences 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. See *App Distribution Guide*, which is available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

25.4.1.3 Adding a Custom Image to an iOS Application

Follow the steps in the task to rebrand an application by overriding the default Oracle images used for application icons and artwork with custom images.

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 25-15, enable you to enter the locations of custom images used for different situations, device orientation, and device resolutions. For more information on iOS application icon images, see 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/`).

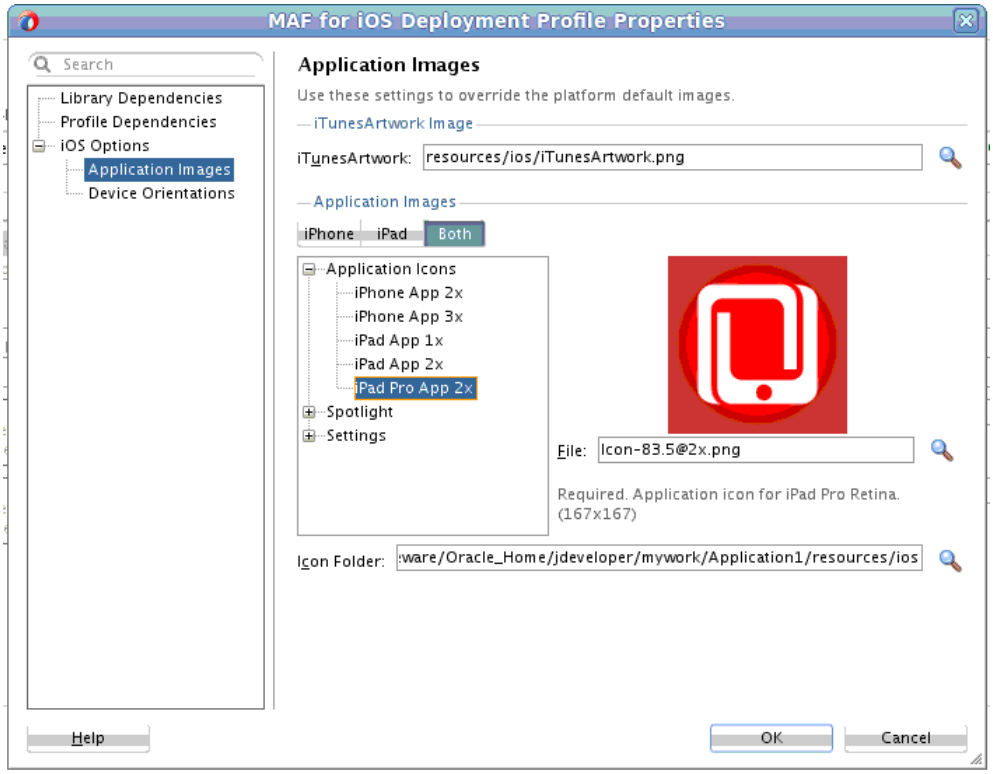
Note:

All images must be in the PNG format.

To add custom images:

1. Select **Application**, **Application Properties**, and then **Deployment**.
2. On the **Deployment** page, double-click an iOS deployment profile.
3. Select **iOS Options**, and then **Application Images** from the tree on the left of the iOS deployment profile properties editor.
4. Select **Browse** to select an icon image to override the default Apple image that iTunes assigns to `.ipa` files. This image is required for all applications and must be 512 x 512 pixels for both iPhone and iPad applications. See What You May Need to Know About iTunes Artwork.
5. Select the device type to display the available image types in the tree. By default, MAF displays all of the image styles and types available to iPad and iPhone devices. However, you can narrow the selection by selecting the device type, as shown in Figure 25-15. In the **Icon Folder** field, MAF displays the location within the `Resources` directory of the application where these image files are stored.

Figure 25-15 Selecting the Application Images



6. Select an image type from the tree.
7. In the **File** field, select **Browse** to select another image. This image file must exist within the current application.

   During deployment, JDeveloper copies the custom image file into the deployment profile and renames it to match the name of the default image.
8. Click **OK**.

25.4.1.4 What You May Need to Know About iTunes Artwork

Applications that are deployed to an iOS device by means of iTunes use the default Oracle image. To differentiate between versions of an application you can select an iTunes artwork image as the icon for a deployed application.

By default, mobile applications deployed to an iOS device through iTunes 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 25-16 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).

Figure 25-16 Custom and Default Application Icons

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*.

Note:

iTunes artwork is only packaged into the application when you select the deployment type called **Deploy to iTunes for synchronization to device**.

25.4.1.5 How to Restrict the Display to a Specific Device Orientation

MAF supports all device display orientations for iPhone and iPad. Follow the steps in the task to restrict the display of an application 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 25-17

Figure 25-17 Select a Device Orientation

To limit the display of an application to a specific device orientation:

1. Select **Device Orientations**, as shown in Figure 25-17.
2. Clear all unneeded orientations from among those listed in Table 25-3. By default, MAF deploys to all of these device orientations. By default, all of these orientations are selected.

Table 25-3 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.

Table 25-3 (Cont.) iPhone Device Orientations

Icon	Description
	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.
	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**.

25.4.1.6 What Happens When You Deselect Device Orientations

When you deselect a device orientation, the source `.plist` file gets updated.

Deselecting a device orientation updates the source `.plist` file.

All iPad orientations must be supported (selected) to enable iPad multitasking. When iPad multitasking is enabled, features such as split view mode can be used on compatible iOS devices.

25.4.2 How to Deploy an iOS Application to an iOS Simulator

Ensure that prerequisites are met, and follow the steps in the task to deploy an application to an iOS simulator using the **Deploy application to simulator** option for **Deploy**.

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:

- Run Xcode after installing it, agree to the licensing agreements, and perform other post-installation tasks, as prompted.

  Note:

  You must run Xcode at least once before you deploy the application to the iOS simulator. Otherwise, the deployment will not succeed.

- On the **iOS Options** page of the deployment profile, select **Debug**, and then click **OK**.
- Before you deploy an application, shut down the iOS simulator if it is running. If you do not shut down the simulator, the deployment will do it for you.
- See *Simulator User Guide*, available in the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`). The iOS simulator is installed with Xcode.

To deploy an application to an iOS simulator:

1. Select **Applications**, then **Deploy**, then select an iOS deployment profile.
2. Select **Deploy application to simulator** and then select **Next**.
3. Review the **Summary** page which displays the following values. Click **Finish**.
   - **Application 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.
   - **Signature**—The developer or company that authored the application. If this value has not been configured in the iOS Platform preferences page, 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 preferences page, then the **Summary** page displays *<Not Specified>*.
   - **Team**—The identifier of the team. MAF automatically populates this input field with a value that it extracts from the provisioning profile.

   Note:

   Deployment to an iOS simulator does not require that the values for Signing Identity, Team, and Provisioning Profile be defined. In this deployment scenario, the **Summary** page displays *<Not Specified>* for these values.

25.4.3 How to Deploy an Application to an iOS-Powered Device

Members of iOS Developer Program or the iOS Developer Enterprise Program can follow the steps in the task to deploy an application to an iOS-powered device by

using the **Deploy to iTunes for Synchronization to device** for **Deploy**. You can also change the default directory for the iTunes Media folder.

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 JDeveloper to an 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 25-18.

  Tip:

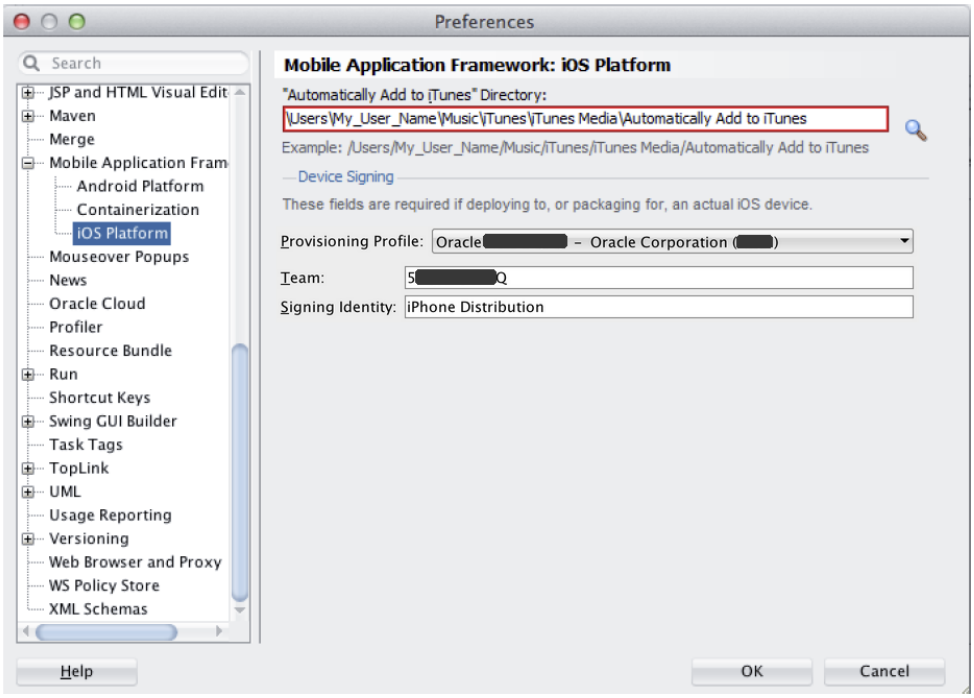
  Although your user home directory (`/Users/<username>/Music/iTunes/iTunes Media/Automatically Add to iTunes.localized`) is the default directory for the iTunes Media folder, you can change the location of this folder as follows:

  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 preferences page.

Figure 25-18 Setting the Location for the iTunes Media Folder



- Enter the name and location of the provisioning profile, the team identifier, and the signing identity 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, team identifier, and signing identity to deploy to a device or to an archive. MAF logs applications that fail to deploy under such circumstances. For more information, see What You May Need to Know About Deploying an Application to an iOS-Powered Device.

- On the **iOS Options** page of the deployment profile, select **Debug** as the build mode and then **OK**.
- Refer to *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. Select **Applications**, then **Deploy**, and then select an iOS deployment profile.
2. Select **Deploy to iTunes for Synchronization to device** and then choose **Next**.
3. Review the **Summary** page, which displays the following values. Click **Finish**.
    - **Application 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.
- **Signature**—The developer (or company) who authored the application. If this value has not been configured in the **iOS Platform preferences** page, 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 preferences** page, then the **Summary** page displays <*Not Specified*>.
- **Team**—The identifier of the team. MAF automatically populates this input field with a value that it extracts from the provisioning profile.

Note:

You must specify the Signature, Team, and Provisioning Profile values in the iOS Platform preferences page to enable deployment to iTunes.

---

4. Connect the iOS-powered device to the development computer.
5. Open iTunes and then synchronize the device.

25.4.4 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 25-16 after a successful deployment.

25.4.5 What You May Need to Know About Deploying an Application to an iOS-Powered Device

A provisioning profile must be created using the iOS Provisioning Portal before an application can be deployed to an iOS-powered device.

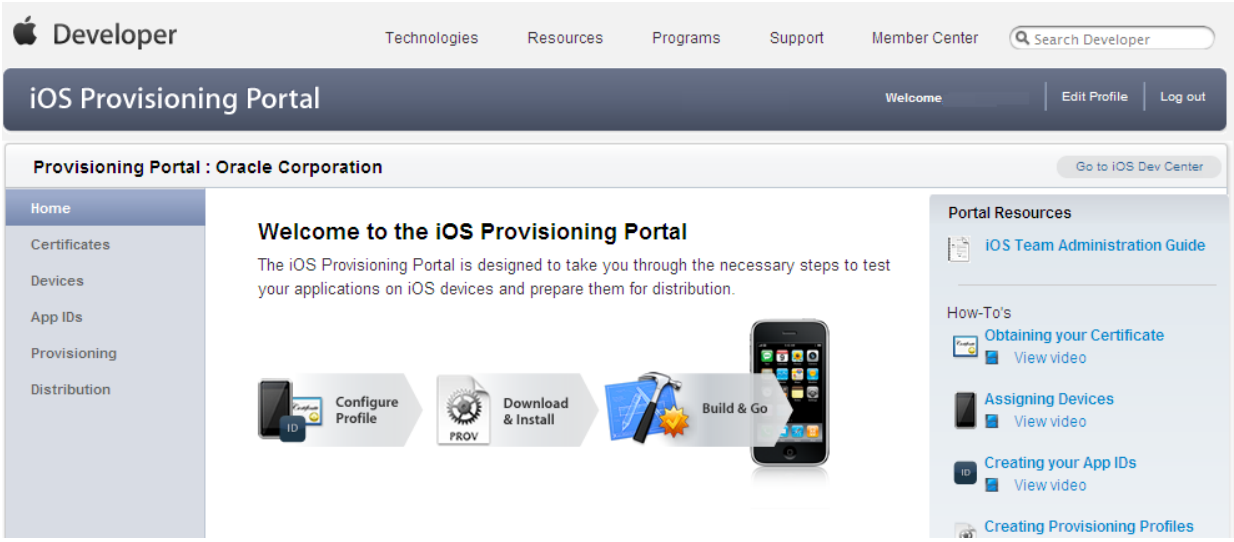
You cannot deploy an iOS application (that is, an `.ipa` file) to an iOS-powered device or publish it to either the App Store or to an internal hosted download site without first creating a provisioning profile using the iOS Provisioning Portal, which is accessible only to members of the iOS Developer Program. You enter the location of the provisioning profile and the name of the certificate in the iOS Platform preferences page as described in Setting the Device Signing Options.

As noted in *App Distribution Guide* (which is available in 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 25-19 The iOS Provisioning Portal



25.4.5.1 Creating iOS Development Certificates

An iOS Development Certificate electronically associates a developer identity with a public key and private key. Download and install a certificate on the development computer to sign applications for deployment.

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.

25.4.5.2 Registering an Apple Device for Testing and Debugging

An Apple device must be included the Current Available Devices list on the Devices page on iOS Provisioning Portal so that it is uniquely identified as a device to which applications can be deployed.

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).

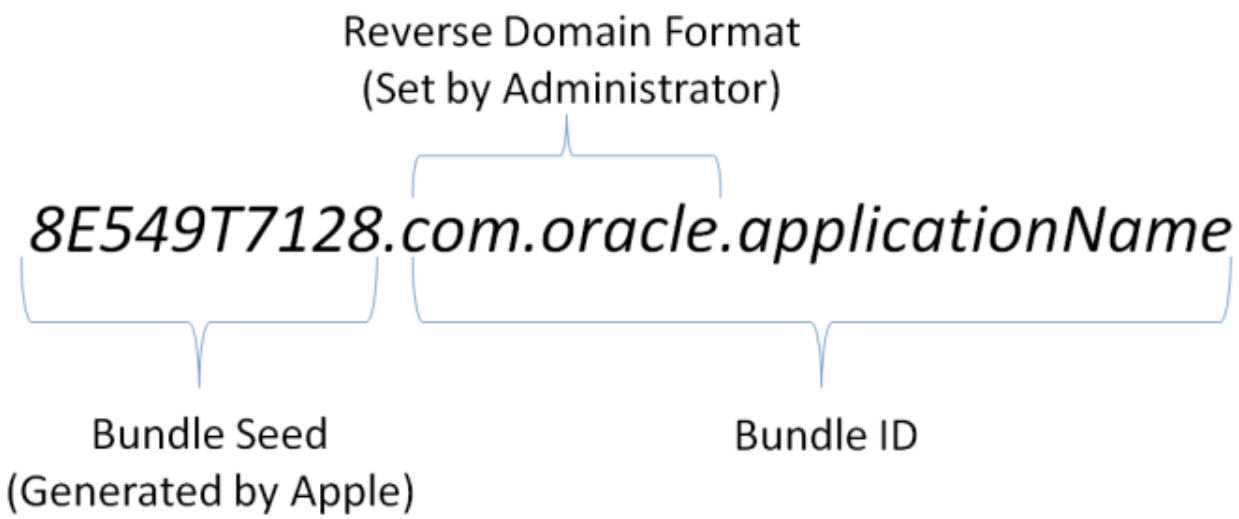
25.4.5.3 Registering an Application ID

Application IDs uniquely identify applications on a device. An application ID consists of a Bundle ID, identical to the application ID on the Preferences page for the iOS Platform, and a 10-character alpha-numeric prefix from Apple.

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 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 25-20 illustrates an application ID that is unique, one that does not share files or the Keychain with any other applications.

Figure 25-20 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. See Provisioning and Development of *Local and Push Notification Programming Guide*, available in 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 iOS Platform preferences page.

25.4.6 How to Distribute an iOS Application to the App Store

After debugging and testing an application on an iOS device, it can be published to the App Store or an internal download site for wider distribution. Ensure that prerequisites are met, and follow the steps in the task to publish 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 25-18, enter the location of the `Automatically Add to iTunes` directory.

  Tip:

  Run iTunes at least once before entering this location. See also How to Deploy an Application to an iOS-Powered Device.

- Test the application on an actual iOS device. See 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, refer to the App Store distribution guidelines at http://developer.apple.com/.

- You may want to review the *iTunes Connect Developer Guide* available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

- In the iOS Options page of the deployment profile, select *App Distribution Guide* **Release** as the build mode and then click **OK**.

To distribute an iOS application to the App Store:

1. Select **Applications**, then **Deploy**, and then select an iOS deployment profile.
2. Select **Deploy to Distribution Package**.
3. Review the **Summary** page, which displays the following values. Click **Finish**.

   - **Application 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.
   - **Signature**—The application's author. If this value has not been configured in the **iOS Platform preference** page of the deployment profile, 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*>.

- **Team**—The identifier of the team. MAF automatically populates this input field with a value that it extracts from the provisioning profile.

Note:

You must specify the Signature, Provisioning Profile, and Team values in the **iOS Platform preferences** 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. See *iTunes Connect Developer Guide* for information (http://developer.apple.com/library/ios/navigation/).
6. See *iTunes Connect Developer Guide* for information on updating the binary (http://developer.apple.com/library/ios/navigation/).

25.5 Deploying a MAF Application to the Universal Windows Platform

MAF applications can either be deployed in the release or debug mode to the local development computer or to an installation package using a PowerShell script. Complete the listed setup and configuration tasks, and deploy the application with the ready-to-use Windows1 deployment profile.

MAF applications can either be deployed to the local machine on which you develop the application, or to an installation package that you use to install the application on a supported UWP device.

MAF provides two build modes for an application that is to be deployed to the UWP. Use the debug mode to test and debug your application as you go through the development cycle. Use the release mode to deploy an application that is release ready.

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. The UWP device that you use must run the Windows 10 operating system. You must enable Developer mode on this machine.

   For more information, see the Installation Requirements for MAF Applications to be Deployed to the Windows Platform section in *Installing Oracle Mobile Application Framework*.

2. Install the Visual Studio software from Microsoft. The Visual Studio package contains the Windows SDK required to deploy applications to the UWP.

   For more information, see the Setting Up Development Tools for the Universal Windows Platform section in *Installing Oracle Mobile Application Framework*.

3. Create a certificate (a personal information exchange file) to digitally sign the application you want to deploy.

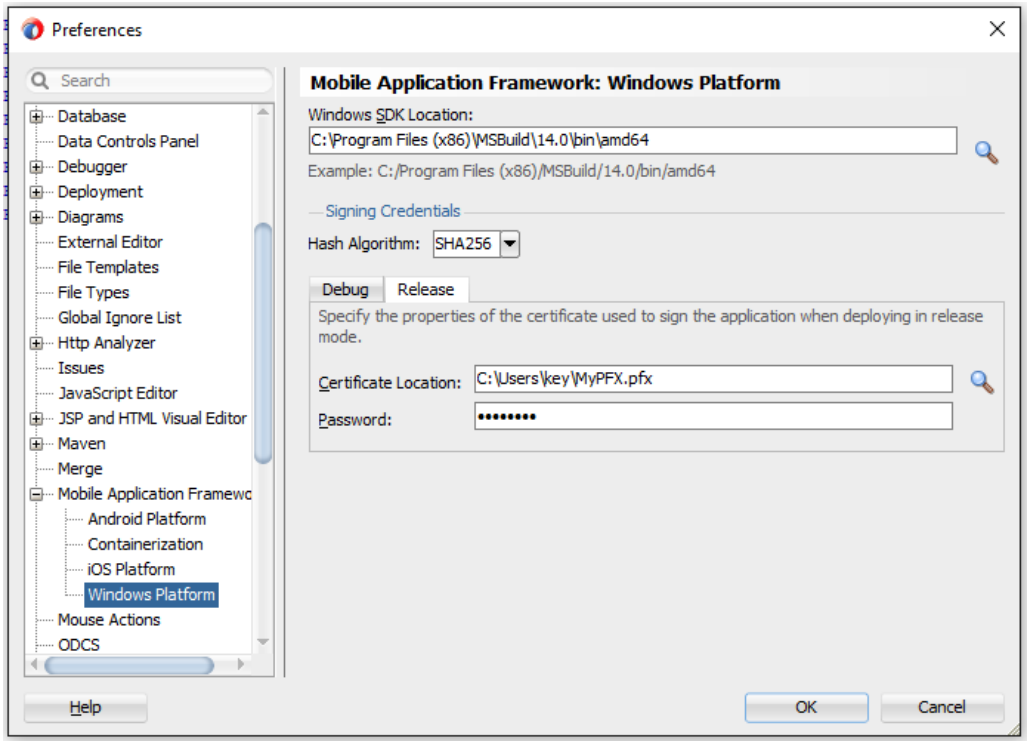
For more information, see the Creating a PFX File for MAF Applications section in *Installing Oracle Mobile Application Framework*.

4. Enter the following values in the Windows Platform dialog shown in Figure 25-21:

   - The location of the Windows SDK that is installed with the Visual Studio software.
   - The location and password (if required) for the certificate that you use to sign the application. Do this in both the Debug and Release dialogs if you intend to deploy your MAF application in both the modes.
   - Ignore the **Hash Algorithm** field. MAF does not use it.

   You access the Windows Platform dialog from the JDeveloper **Tools** menu by clicking **Preferences**, and then clicking **Mobile Application Framework** as shown in Figure 25-21.

Figure 25-21 Windows Platform Preferences to Deploy a MAF Application



MAF provides a ready-to-use deployment profile (`Windows1`) to deploy a MAF application in the debug mode. Provided your development machine and environment meet the setup and configuration requirements, you can deploy the application either to a local machine or to a package.

You can edit this ready-to-use deployment profile or create one or more new deployment profiles. By creating a new deployment profile, or by editing an existing profile, you can:

- Specify additional library and profile dependencies for your application.
- Choose either the debug or release build modes.
- Specify options such as the application bundle ID, the archive name and version.

- Specify the images that the application uses to render its logo on the splash screen and the icon that appears on the user's device.

For more information about how to edit or create a deployment profile, see Working with Deployment Profiles. Click **Help** in the deployment profile dialogs for information about the fields that you can configure in each dialog.

Before you deploy your application, note the following issues that can hinder 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.
- When an application, which uses the Contacts plugin, is deployed to the UWP, ensure that the Contacts plugin is listed first in the list of plugins in the `maf-plugins.xml` file.

For information about how to debug an application that you deploy in the debug mode, see How to Debug Java Code on the Universal Windows Platform.

25.5.1 How to Deploy a MAF Application to the Universal Windows Platform

Deploy the application using a MAF for Windows deployment profile that deploys the application to the UWP.

To deploy a MAF application to the UWP:

1. From JDeveloper's main menu, select **Application**, then **Deploy**, and then **Windows1**.

   Where **Windows1** is a MAF for Windows deployment profile.

2. Select the appropriate deployment option from the list in the dialog that appears.
3. Click **Finish**.

25.5.2 What Happens When You Deploy a MAF Application to the Universal Windows Platform

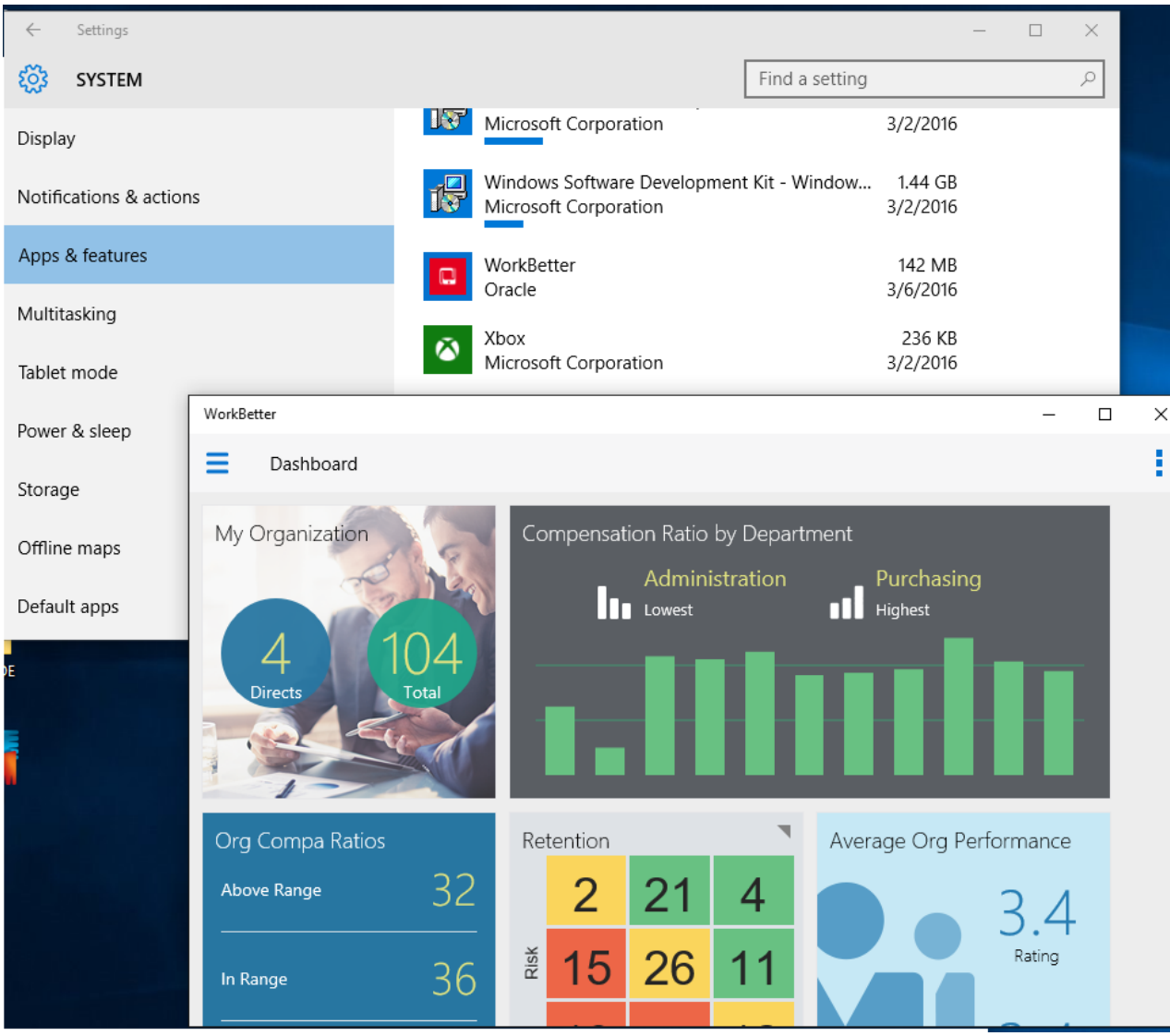
Depending on whether you select the **Deploy application to local machine** option, or the **Deploy application to package** option, MAF deploys the application to a computer or to an installation package.

If you select the **Deploy application to local machine** option, MAF deploys the application to the machine that you are using and installs the application there. In the foreground of Figure 25-22 is an instance of the WorkBetter sample application that

has been deployed to a local machine in release mode. In the background of Figure 25-22, you can view the WorkBetter sample application listed among the installed applications on the Apps & features page of the Windows 10 computer.

A red icon appears in the upper-left of a MAF application screen that you deploy in debug mode to indicate that the application is in debug mode. As with the application deployed in release mode, you can view and uninstall this application in the Apps & features page of the Windows 10 computer.

Figure 25-22 MAF Application Deployed in Release Mode on Windows Local Machine



If you select the **Deploy application to package** option, MAF deploys the application to an installation package in the following directory:

```
C:\path\to\appRoot\deploy\[DeploymentProfileName]\[deployMode]\MafTemplate
\AppPackages
```

For example, the WorkBetter sample application, shown in Figure 25-22, that deployed in release mode using the `Windows1` deployment profile deployed to the following directory:

```
C:\...\PublicSamples\WorkBetter\deploy\Windows1\release\MafTemplate\AppPackages
```

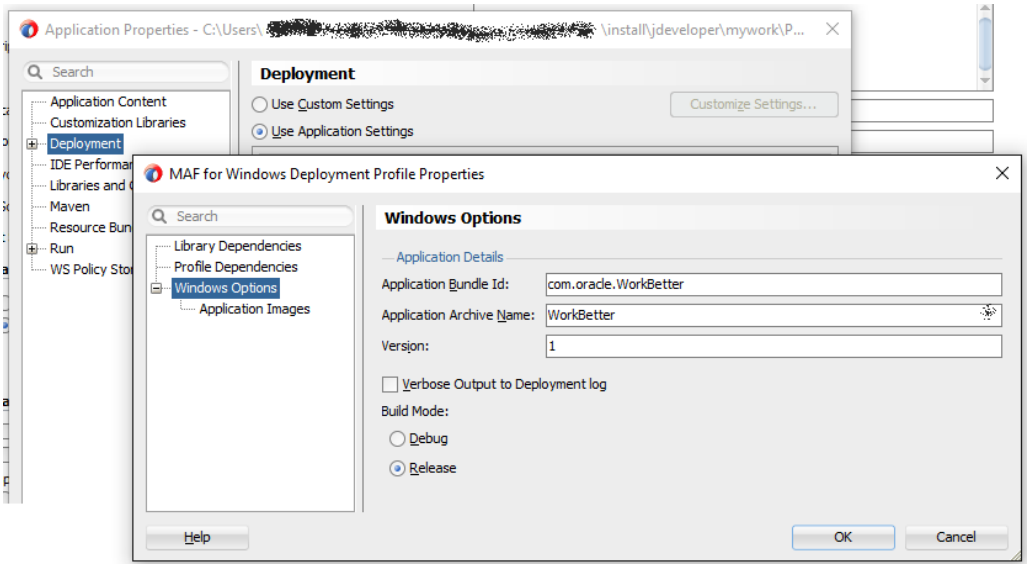
The `AppPackages` directory contains another directory (`MafTemplate_*_Test`). Distribute the contents of this latter directory to the end users with supported UWP devices who want to install your MAF application. The directory includes a PowerShell script (`Add-AppDevPackage.ps1`) that end users execute 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 and Build mode that you specify in the Windows Options dialog, shown in Figure 25-23. You access this dialog from JDeveloper's Deployment page when you edit a deployment profile. To access the Deployment page, click **Application** and then **Application Properties**.

For example, if you specify 2 as the Version number and select the Debug build mode, the directory name is `MafTemplate_2.0.0.0_x64_Debug_Test`. A version number of 3 and the Release build mode produces a `MafTemplate_3.0.0.0_x64_Test` directory.

Figure 25-23 Windows Options Dialog for Deployment Profile



25.6 Overview of MAF Quick Deployment of Applications

A quick deployment skips some deployment steps to save deployment time. Quick deployment saves time by passing only new or changed file content to an emulator or a simulator.

MAF quick deployment ensures developer productivity by providing competitive deployment times without deterioration of performance.

Quick Deployment of Applications

A quick deployment, as opposed to the normal full deployment, saves deployment time by skipping some deployment steps. A quick deployment passes only new or changed file content to an emulator or a simulator.

To use quick deployment on an Android emulator, select the **Deploy application to emulator deployment** action. For example, for a deployment profile named Android2, on the **Application** menu, select **Deploy**, then **Android2**, and then select a deployment action. See Deploy an Android Application to an Android Emulator. To use quick deployment on an iOS simulator, for a deployment profile named iOS2, from the **Application** menu, select **Deploy**, then **iOS2**, and then select a deployment action. See Deploy an iOS Application to an iOS Simulator.

> **Note:** The first deployment is a full deployment. A quick deployment may follow an initial full deployment.

The deployment of an application to an emulator or a simulator starts a Quick Deployment Session Analysis. The results of the analysis decide whether the deployment must be a quick deployment or a full deployment.

The Quick Deployment Session Analysis is initiated to detect application file changes. The MAF tool that analyzes application file changes triggers a quick deployment. If the analysis fails to find deployment artifacts from a previous Quick Deployment Session that used the same deployment profile, a full deployment is triggered.

MAF simplifies the deployment functionality. When you make changes to application files in JDeveloper, those changes are moved to the application that is deployed to an emulator or a simulator, thus removing the need for redeployment.

Key Features of Quick Deployment

The following features define quick deployment.

- Quick deployment is supported for applications that are deployed to an Android emulator or an iOS simulator.
- Virtual box images can be used for quick deployment provided that root access is configured for the images.
- Changes made to an application in JDeveloper pushes the changes to the application deployed to an emulator or a simulator, without a full deployment, for example, changes to AMX pages or task flows. For the complete list of application changes, see Artifacts That Support Quick Deployment.
- Deletion of files triggers a full deployment.
- The quick deployment of an application only updates new or modified files. Quick deployment does no additional deployment processing.

25.6.1 About the Artifacts That Support Quick Deployment

Changes to certain artifacts, AMX pages, binding changes, and others, support MAF quick deployment.

Artifact Changes That Support MAF Quick Deployment

The following artifact changes support quick deployment:

- Changes to AMX pages
- Changes to task flows
- Changes to bindings
- Changes to maf-skins.xml
- Addition of new style sheets and referencing them in maf-skins.xml
- Changes to maf-config.xml
- Changes to adf-config.xml
- Changes to connections.xml
- Changes to wsm-assembly.xml
- Changes to sync-config.xml

25.6.2 About Requirements for Quick Deployment

A quick deployment is only initiated if the listed prerequisites are met.

Prerequisites for the Quick Deployment of a MAF Application

The quick deployment of an application starts when the following requirements are met:

- An installed application is available on the target Android emulator or iOS simulator.
- A unique bundle id for the MAF application.
- An initial, full deployment must have been completed.

25.6.3 What Happens During a Quick Deployment Session

A Quick Deployment Session analyzes and determines whether a deployment must be a quick deployment or a full deployment. The session is initiated when an application is deployed.

The deployment of an application to an emulator or a simulator starts a Quick Deployment Session. The session analyzes and decides whether the deployment must be a quick deployment or a full deployment.

MAF Quick Deployment Session and Analysis

A quick deployment session proceeds as follows:

- Conducts an initial quick deployment session analysis
- Notes the application file changes:
  - Since the last quick deployment session
  - With the same deployment profile
- If the changed files support quick deployment, the following steps for a quick deployment are completed. Otherwise, a full deployment follows.

  - Copies the modified files to the emulator or simulator
  - Restarts the application on the emulator or simulator

25.6.4 How to Start the Full Deployment of an Application

Certain actions can trigger the full deployment of an application. If you want the full deployment of an application, initiate an action from the list.

Actions That Trigger the Full Deployment of an Application

The following actions trigger the full deployment of an application:

- JDeveloper is restarted.
- Files are changed in non-MAF projects.
- Java files in a MAF feature project are created, changed, or deleted.
- Files are changed in a non-MAF project, and a **Clean All** is performed before deployment.
- Files are deleted in the application, outside of MAF feature projects.
- An application is deployed using a different deployment profile, a profile other than the one used in the previous quick deployment.
- Some artifacts at the application level can trigger a full deployment. Changes to the following artifacts trigger a full deployment:
  - maf-application.xml
  - maf-feature.xml
  - maf-plugins.xml
  - maf.properties
  - logging.properties

25.6.5 How to Force the Full Deployment of an Application

Certain actions, when initiated, can force the full deployment of an application. Initiate an action from the list to start a full deployment.

Certain user actions force the full deployment of a MAF application.

Actions That Force the Full Deployment of a MAF Application

The full deployment of an application can be forced in multiple ways. Use any of the following actions to trigger the full deployment of a MAF application:

- Change any file that does not support quick deployment.
- Click **Build**, and then click **Clean All**.
- Uninstall an application on the Android emulator or iOS simulator.
- Restart JDeveloper.

25.6.6 What You May Need to Know About Quick Deployment Limitations

Some limitations are associated with the quick deployment of MAF applications. The lists enumerate the limitations.

Quick Deployment Limitations: Files Deployment and User Actions

The following files cannot be deployed by means of quick deployment because they need additional processing during deployment:

- Files that are created, updated, or deleted in any project other than a MAF ViewController or ApplicationController project
- Application files that are deleted outside MAF feature projects
- Files that are deleted from within any MAF feature project
- New, changed, or deleted Java files
- Changes to the following artifacts at the application level trigger a full deployment:
  - `maf-application.xml`
  - `maf-feature.xml`
  - `maf-plugins.xml`
  - `maf.properties`
  - `logging.properties`

A quick deployment cannot be initiated in the context of the following user actions:

- Changes or removes the emulator application that was installed by the previous quick deployment.
- Deploys an application using a deployment profile other than the one used in the previous quick deployment.

Quick deployment neither detects nor handles the following changes:

- The quick deployment of an application to a different emulator.
- JDeveloper restart. After JDeveloper restarts, the first deployment is a full deployment. No user action is needed to ensure a full deployment after the restart. A quick deployment may follow an initial full deployment.

25.7 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).

As stated in Reusing MAF Application Content , a FAR is a JAR file that contains the application feature artifacts that can be consumed by mobile applications, such as icon images, resource bundles, HTML, JavaScript, or other implementation-specific files. (A FAR may contain Java classes, though these classes must be compiled.) The following example illustrates the contents of a FAR, which includes a single `maf-feature.xml`

file and a `connections.xml` file. For more information on `connections.xml`, see Configuring End Points Used in MAF Applications .

```
/* Contents of a Feature Archive File */
connections.xml (or some form of connection metadata)

  META-INF
     adfm.xml
     maf-feature.xml
     MANIFEST.MF
     task-flow-registry.xml

  oracle
    application1
      mobile
         Class1.class
         DataBindings.cpx
         pageDefs
           view1PageDefs

  model
     adfc-mobile-config.adfc.diagram
     ViewController-task-flow.adfc.diagram

  public_html
     adfc-mobile-config.xml
       index.html
       navbar-icon.html
       springboard-icon.html
     view1.amx
     ViewController-task-flow.xml
```

Working with Feature Archive files involves the following tasks:

1. Creating a Feature Archive file—You create a Feature Archive by deploying a feature application as a library JAR file.
2. Using the Feature Archive file when creating a mobile application—This includes importing FARs and re-mapping the imported connection.
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:

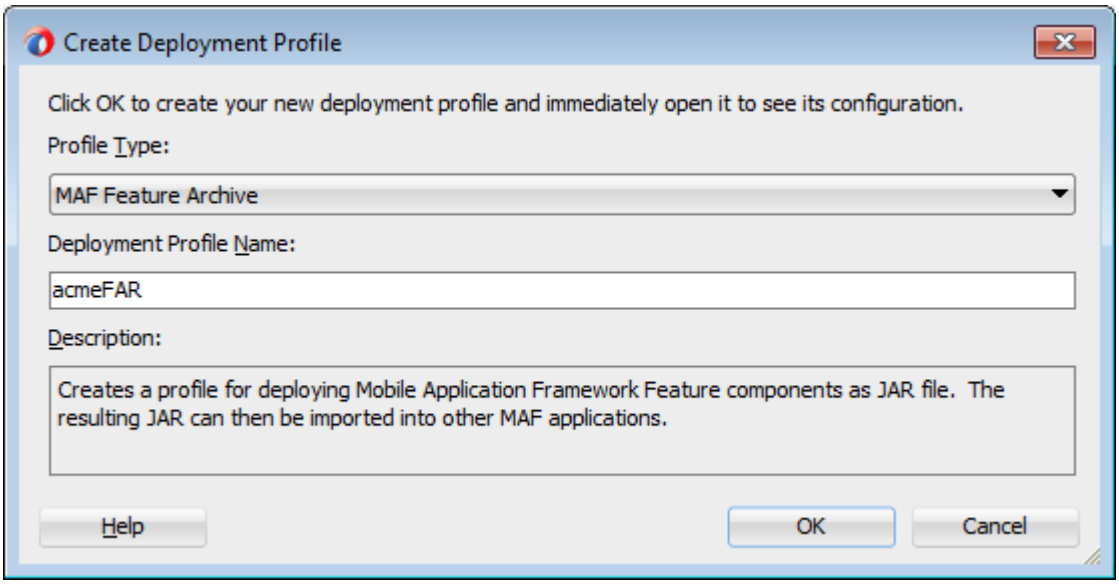
MAF generates FARs during the deployment process. You only need to deploy a view controller project if you use the FAR in another application.

25.7.1 How to Create a Deployment Profile for a Feature Archive

Create connections for the application, and then follow the steps in the task to create a deployment profile for a feature archive.

Use the **Create Deployment Profile** dialog, shown in Figure 25-24.

Figure 25-24 Create MAF Feature Archive Dialog



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.

To create a deployment profile for a Feature Archive:

1. Right-click a view controller project, choose **New**, then **Deploy**, and then **New Deployment Profile**.

   Note:

   You do not need to create a separate, application-level deployment profile.

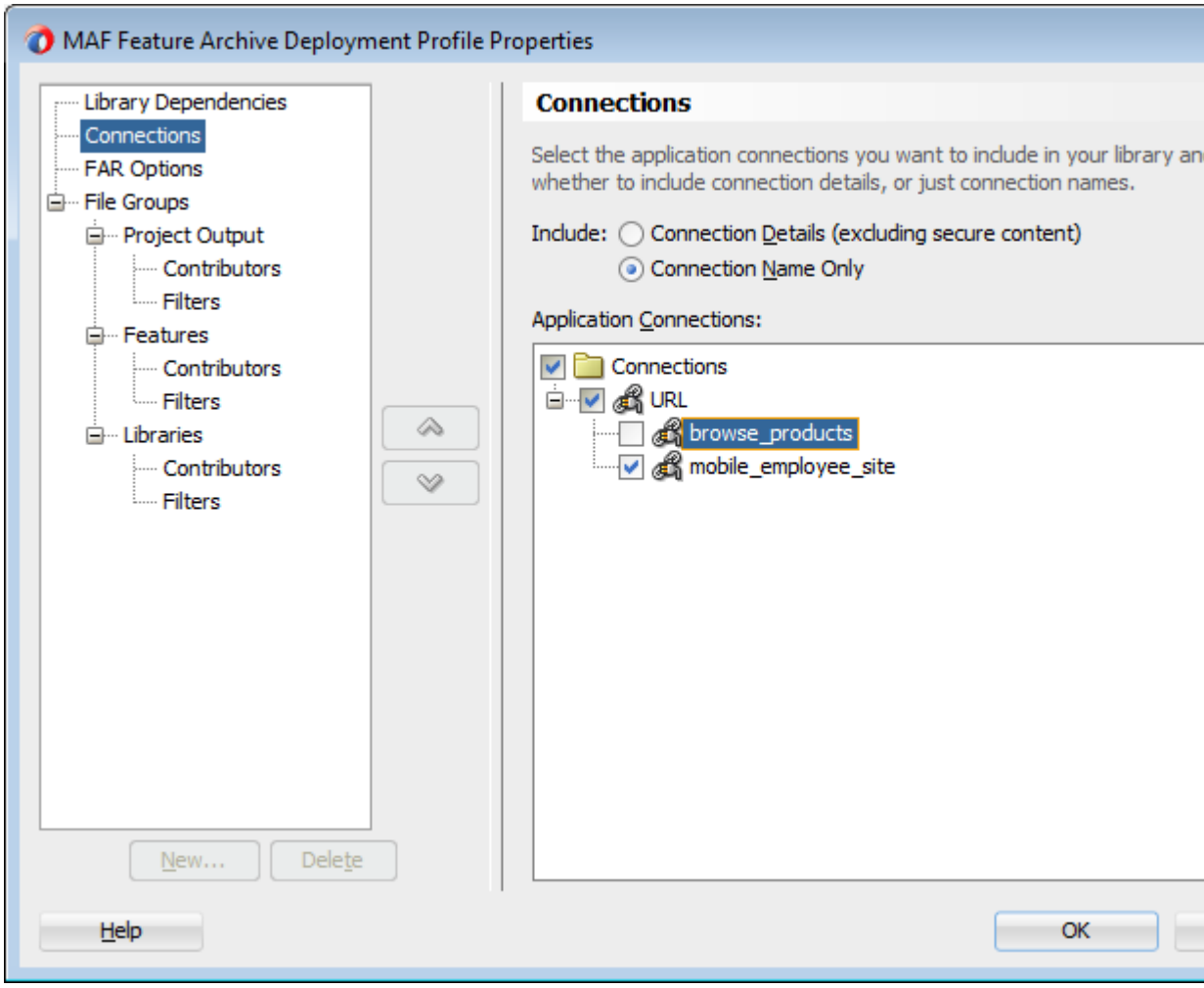
2. Select **MAF Feature Archive** in the **Create Deployment Profile** dialog.
3. Enter a profile name, or accept the default, and then click **OK**.

   Note:

   Name the profile appropriately. Otherwise, you may encounter problems if you upload more than one application feature with the same archive name. See What You May Need to Know About Enabling the Reuse of Feature Archive Resources.

4. Select the connections that you want to include in the Feature Archive JAR file, as shown in Figure 25-25.

Figure 25-25 Selecting a Connection for the FAR



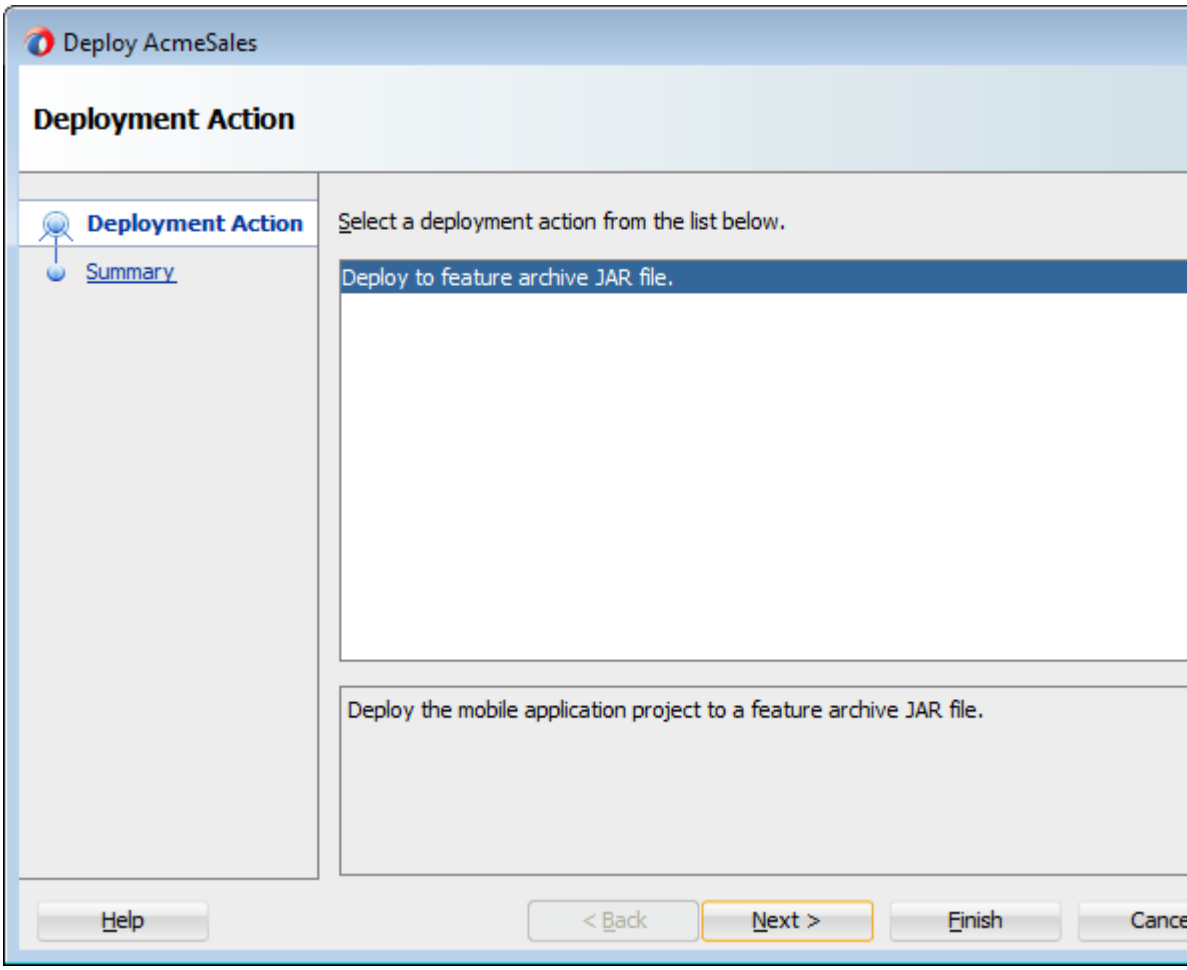
5. Click **Next**, review the options, and then click **Finish**.

25.7.2 How to Deploy the Feature Archive Deployment Profile

MAF provides the **Deploy to feature archive JAR file** option so that you can deploy a FAR as a JAR file. Follow the steps in the task to deploy the Feature Archive deployment profile.

The **Deployment Actions** dialog enables you to deploy the FAR as a JAR file. This dialog, shown in Figure 25-26, includes only one deployment option, **Deploy to feature archive JAR file**.

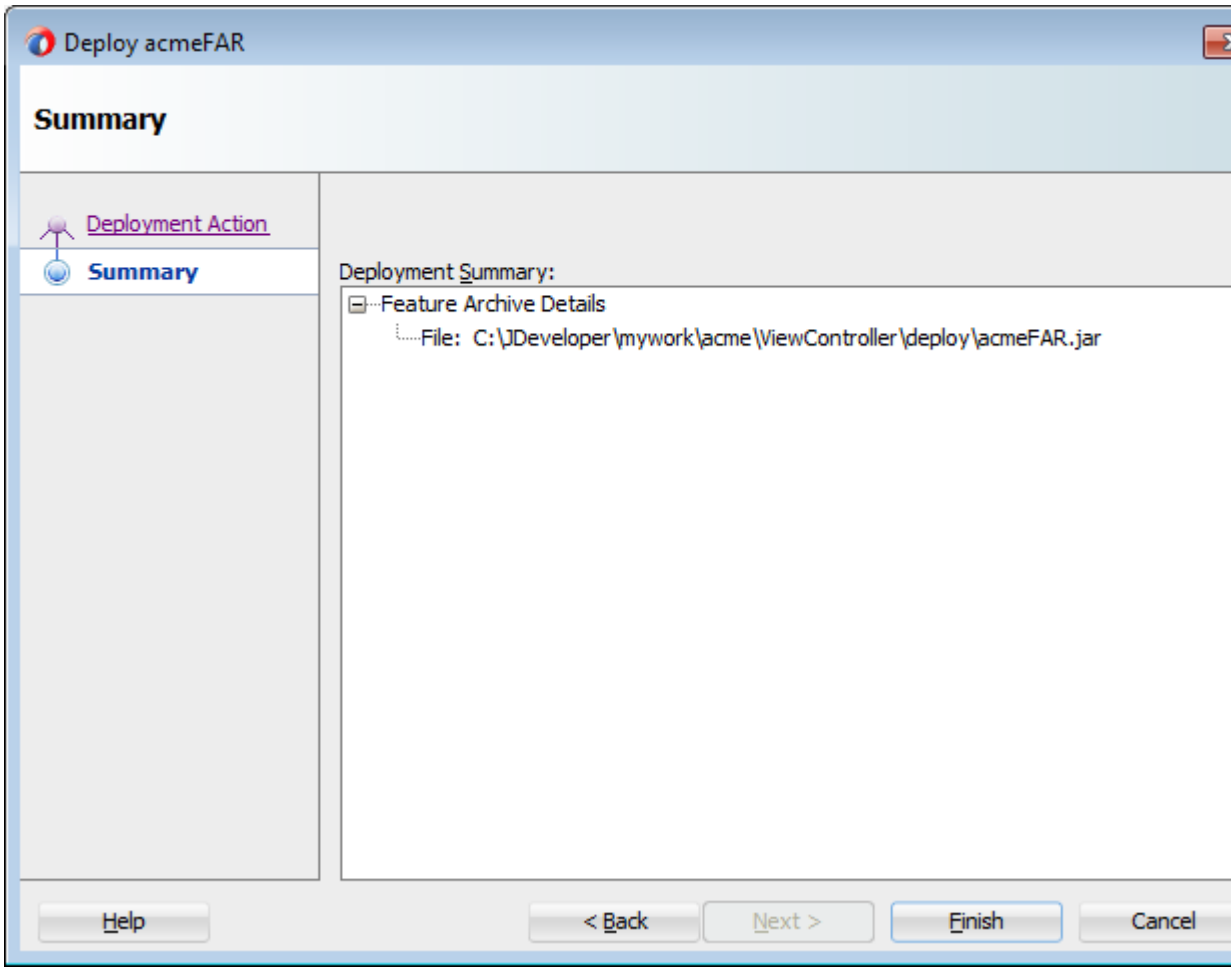
Figure 25-26 Deployment Actions



To deploy the Feature Archive deployment profile:

1. Right-click the view controller project and then select the Feature Archive deployment profile.
2. Click **Finish**. The **Summary** page, shown in Figure 25-27, displays the full path of where the JAR path of the Feature Archive file is deployed.

Figure 25-27 Deployment Summary Page



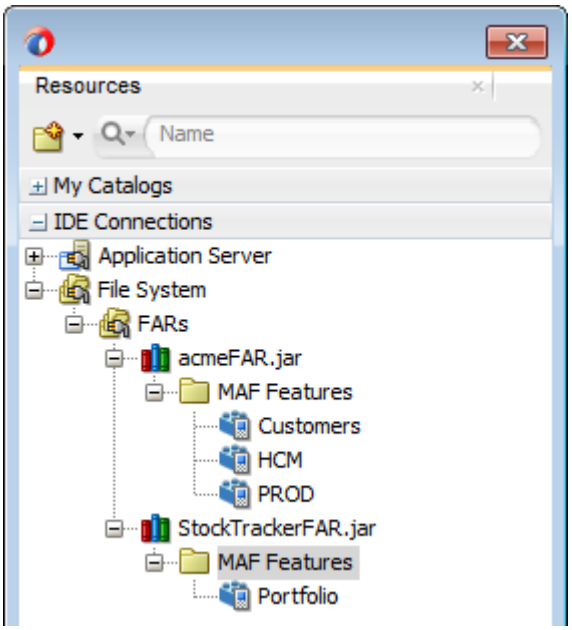
25.7.3 What Happens When You Deploy a Feature Archive File Deployment Profile

MAF packages the content of the application feature into a JAR file (FAR) that it creates in the `AppRoot/ViewController/deploy` directory. This FAR includes AMX pages, task flows, icons, and the other resources that comprise the application feature.

The FAR does not include any shared libraries that the application feature consumed. Review the output in JDeveloper's Deployment – Log window (**Window** > **Log**). If the application feature that you deployed to the FAR includes a shared library, MAF logs a message to notify you that the shared library has been excluded from the FAR. Notify MAF application developers who want to consume the FAR in their applications that they need to obtain the shared library separately if they want to use the shared library resources in their applications when they consume the FAR. For more information about shared libraries, see Reusing MAF Application Content with a MAF Shared Library.

MAF application developers who consume a FAR in their MAF application need to create a file system connection to the directory that contains the FAR. Once they create this connection, the FAR and the application feature(s) it packages appear in the Resources window, as shown by Figure 25-28. For more information about consuming a FAR in a MAF application, see Using FAR Content in a MAF Application.

Figure 25-28 Deployed Feature Archive JARs in the Resources Window



25.8 Creating a Mobile Application Archive File

A new mobile application can be created from an existing mobile application by packaging the original mobile application as a Mobile Application Archive (.maa) file or by creating a new mobile application from the .maa file.

You can create a new mobile application from an existing mobile application by:

- Packaging the original mobile app as a Mobile Application Archive (`.maa`) file
- Create a new mobile application from the .maa file, as described in Creating a New Application from an Application Archive.

An `.maa` file preserves the structure of the mobile application. Table 25-4 describes the contents of this file.

Table 25-4 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.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 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`).
`ExternalLibs`	Contains the application-level libraries (including FARs) that are external to the original mobile application.

Table 25-4 (Cont.) Contents of a Mobile Application Archive File

Directory	Description
`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. • security—Includes the `cacerts` file (the keystore file).

In addition to the artifacts listed in Table 25-4, the `.maa` file includes any folder containing FARs or JAR files that are internal to the original mobile application, as well as its control (`.jws`)file. See also 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.

25.8.1 How to Create a Mobile Application Archive File

Follow the steps in the task to use Mobile Application Archive wizard to create a `.maa` file. You can also create a `.maa` file using OJDeploy.

JDeveloper creates a default MAF Application Archive deployment profile after you create a mobile application. Using the **Mobile Application Archive** wizard, you can create the `.maa` file.

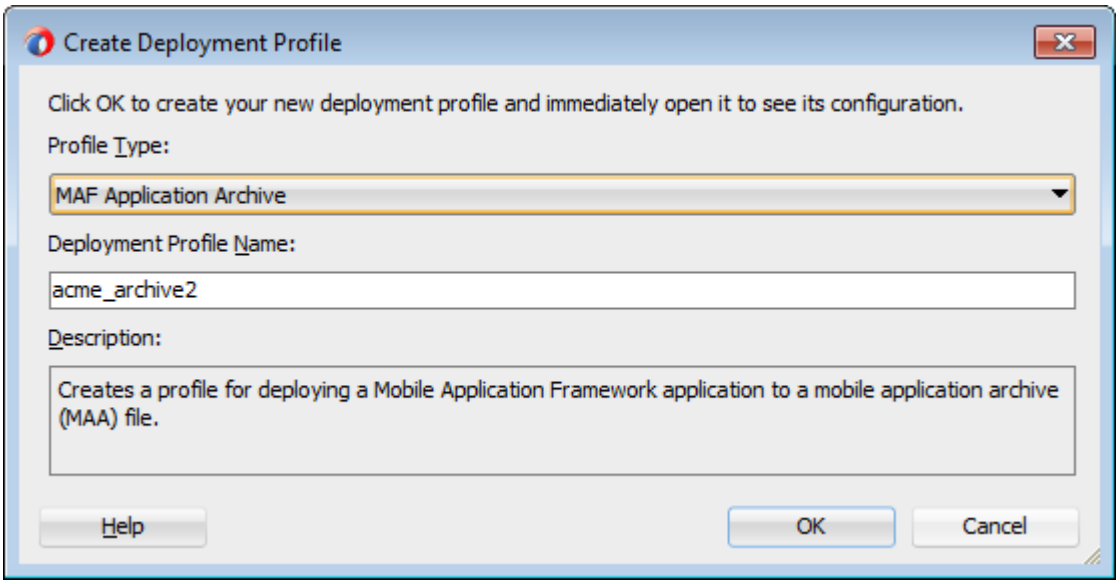
Tip:

You can also create an `.maa` file using OJDeploy, as described in Deploying MAF Applications from the Command Line.

To create a Mobile Application Archive file:

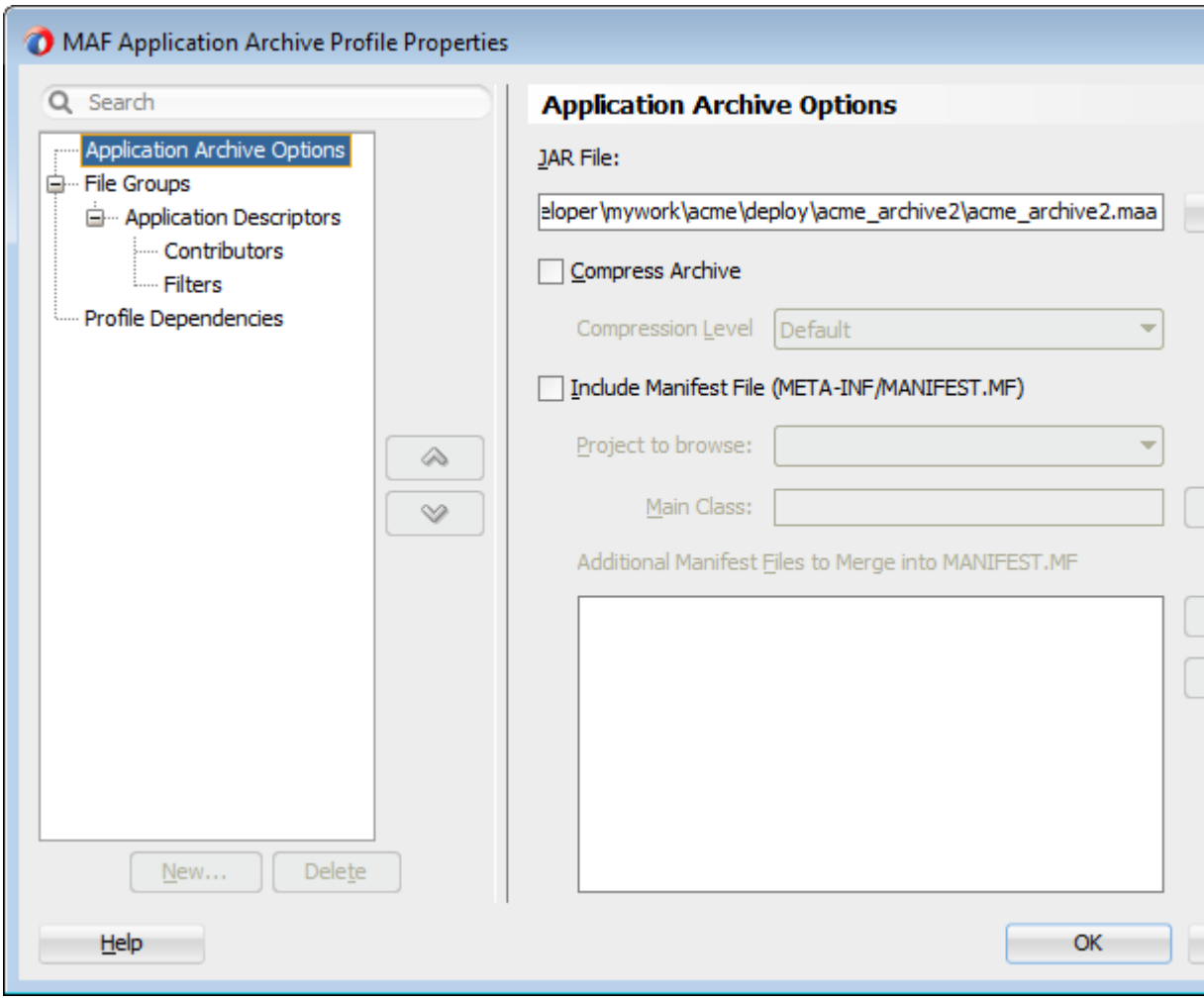
1. Click **Application**, then **Deploy**, then **New Deployment Profile**.
2. In the **Create Deployment Profile** dialog, select **MAF Application Archive** and then click **OK**, as shown in Figure 25-29.

Figure 25-29 Creating an MAA Deployment Profile



3. If needed, enter a name for the Mobile Application Archive in the **Application Archives Options** page, shown in Figure 25-30, or accept the default name (and path). Click **OK**.

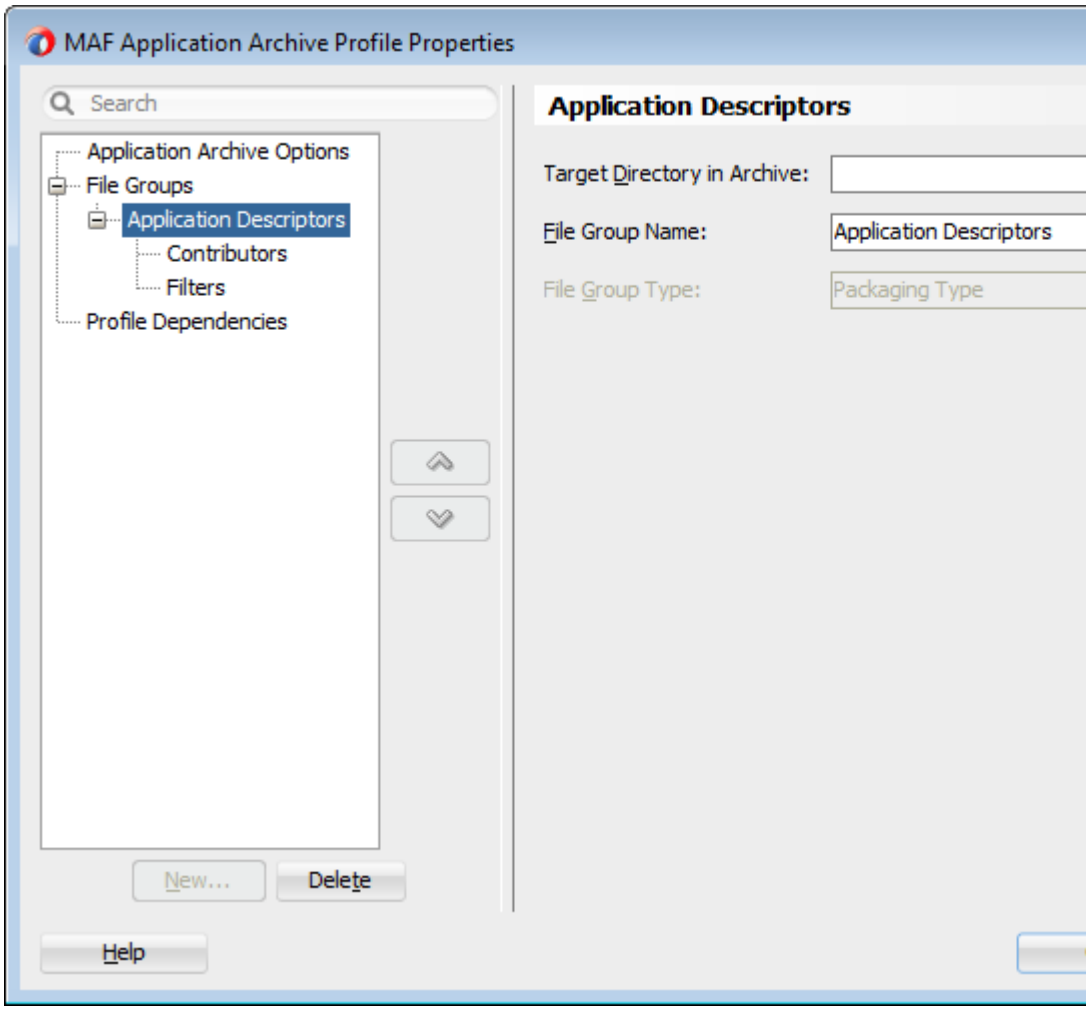
Figure 25-30 Entering a Name and Path for the Mobile Application Archive File



4. If needed, perform the following:

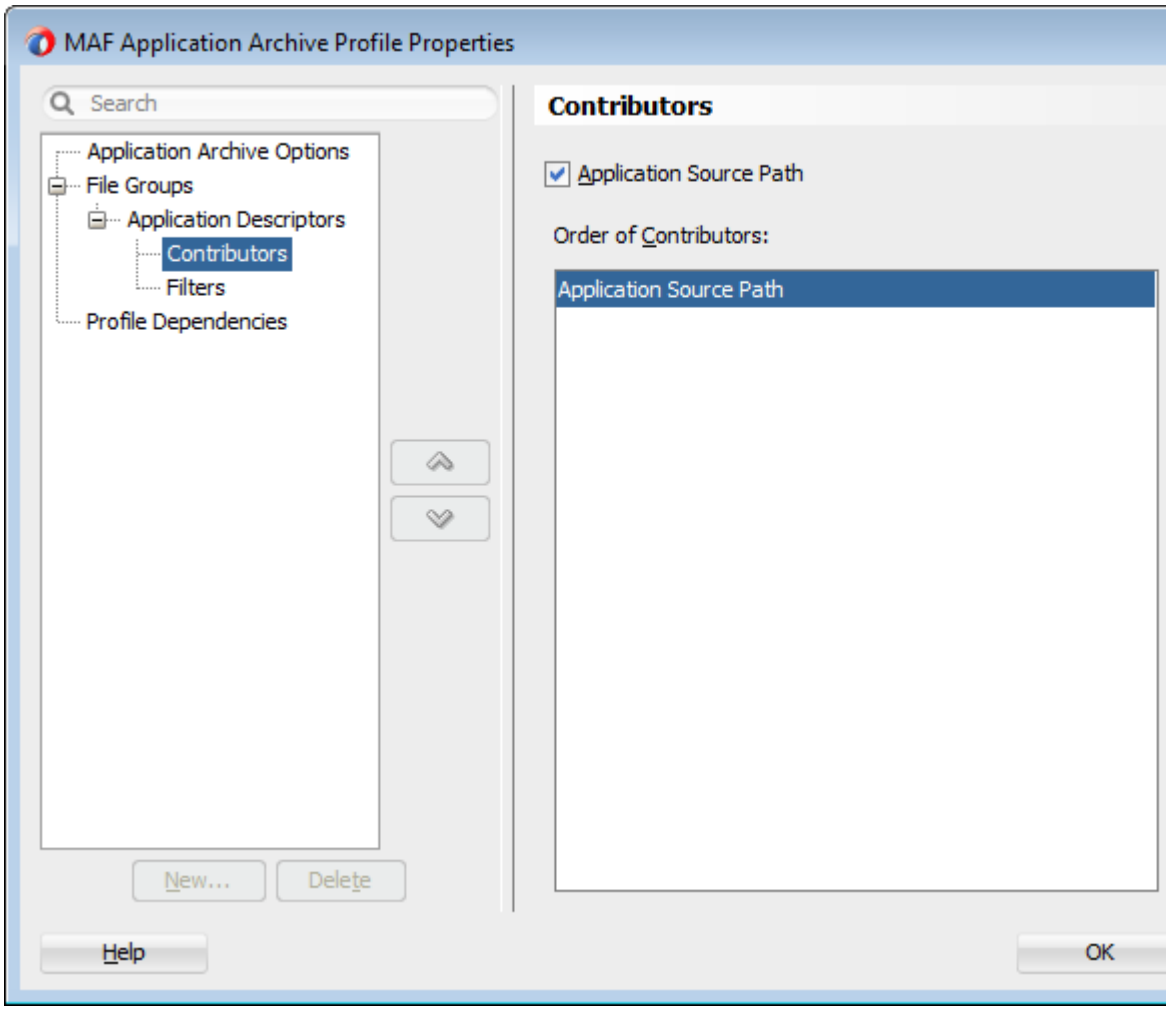
   a. In the **Application Descriptors** page, shown in Figure 25-31, enter the file group name (or accept the default name) used for the contents of the `META-INF` folder (`application_workspace\src\META-INF`).

Figure 25-31 Entering a File Group Name for the META-INF Contents



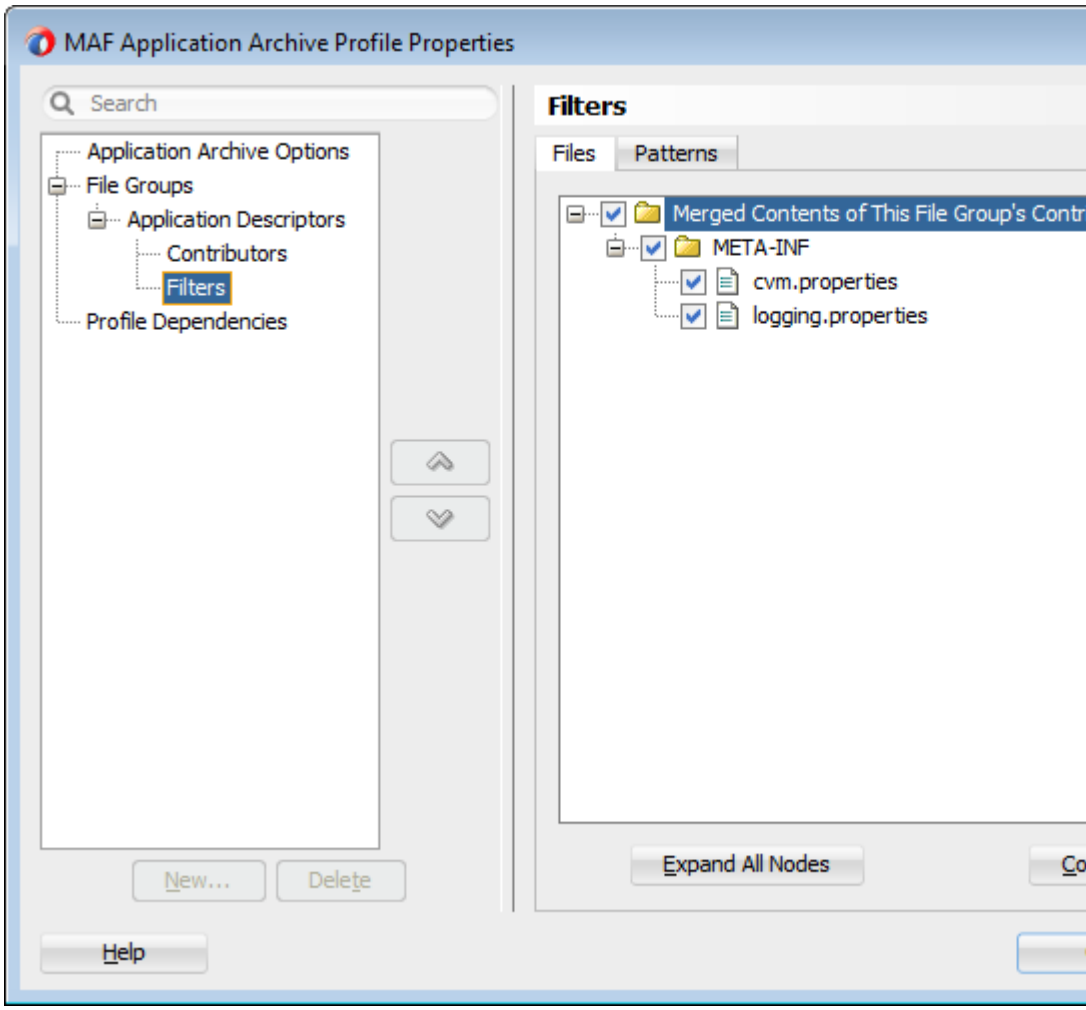
b. Select the **Contributors** sub-page of this **Application Descriptors** page to edit the list of directories and JAR files that provide the contents for the file group.

Figure 25-32 Editing Contributors to the Mobile Application Archive File



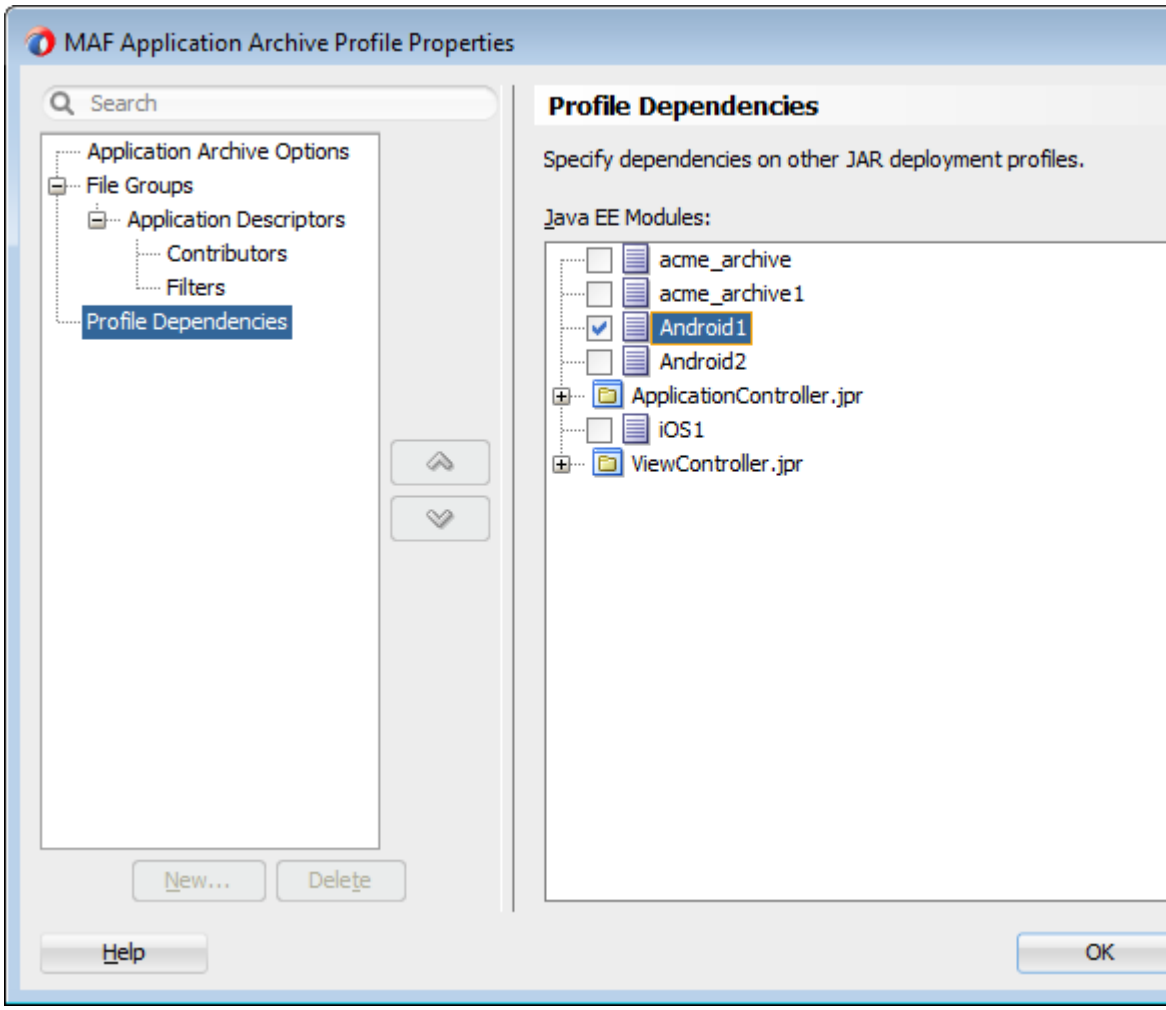
c. Use the **Filters** page, shown in Figure 25-33 to edit the files that will be included in the `.maa` file or set the content inclusion or exclusion rules.

Figure 25-33 Including (or Excluding) Files and Directories



d. Use the **Profile Dependencies** page, shown in Figure 25-34, to specify dependent profiles within the project.

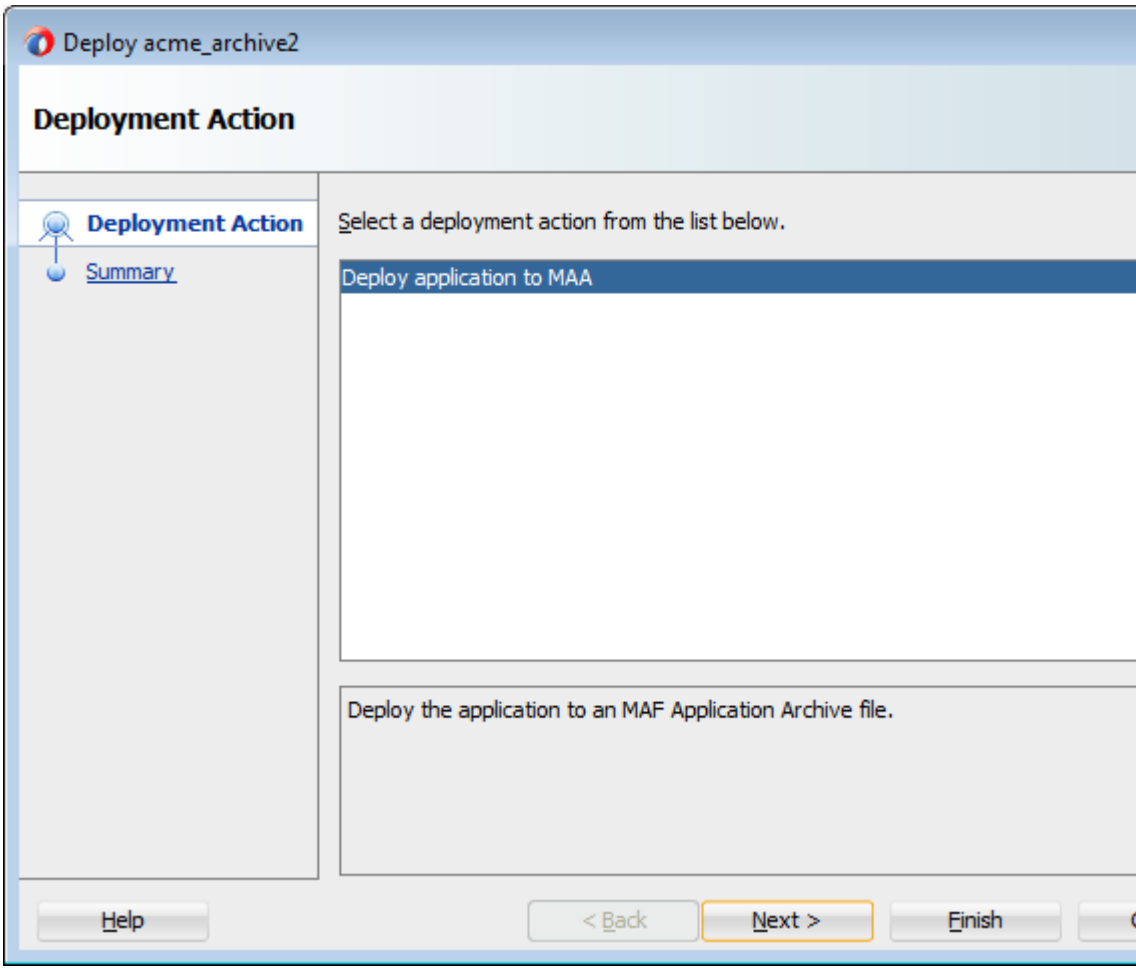
Figure 25-34 Selecting Deployment Profiles



To package a mobile application as a MAF Application Archive file:

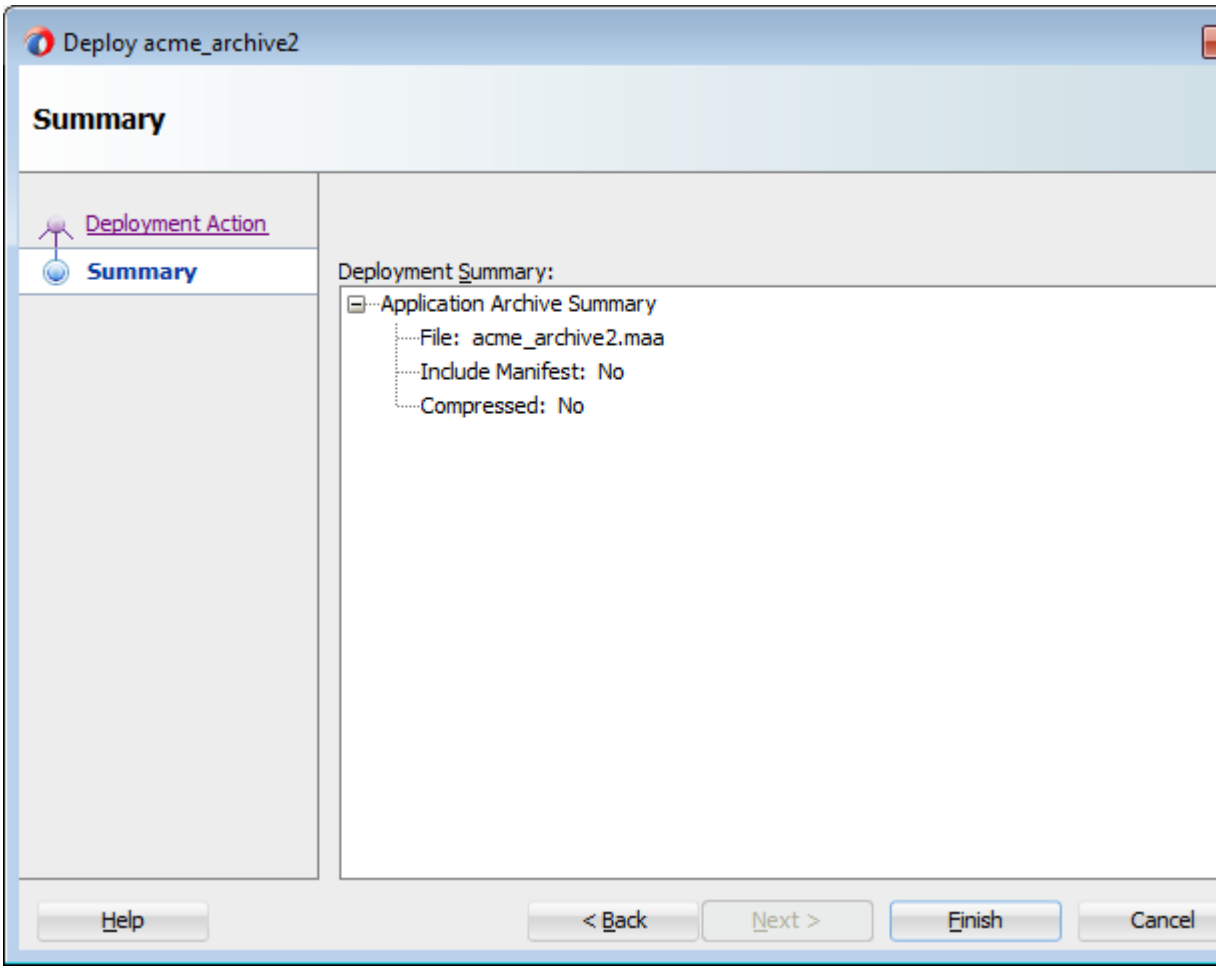
1. Choose **Application**, then **Deploy** and then choose the MAF Application Archive deployment profile.
2. In the **Deployment Action** wizard, select **Deploy application to MAA**, as shown in Figure 25-35.

Figure 25-35 Deployment to a MAF Application Archive File



3. Click **Next** to review the deployment summary, as shown in Figure 25-36.

Figure 25-36 MAF Application Archive Deployment Summary



4. Click **Finish**.

25.9 Creating a New Application from an Application Archive

Use an application archive file to create a new MAF application that facilitates various customizations.

You or others (for example, a colleague or a partner) can create a new MAF application using an application archive file (`.maa`) as a starting point. By deriving a mobile application from an `.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.

The MAF Application Archive (`.maa`) file format enables you to provide third-parties with an unsigned mobile application.

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.

25.9.1 How to Create a New Application from an Application Archive

Follow the steps in the task to use the **MAF Application from Archive File** option and import a `.maa` file into a new mobile application.

You import an `.maa` file into a new mobile application.

To create a new application from an application archive:

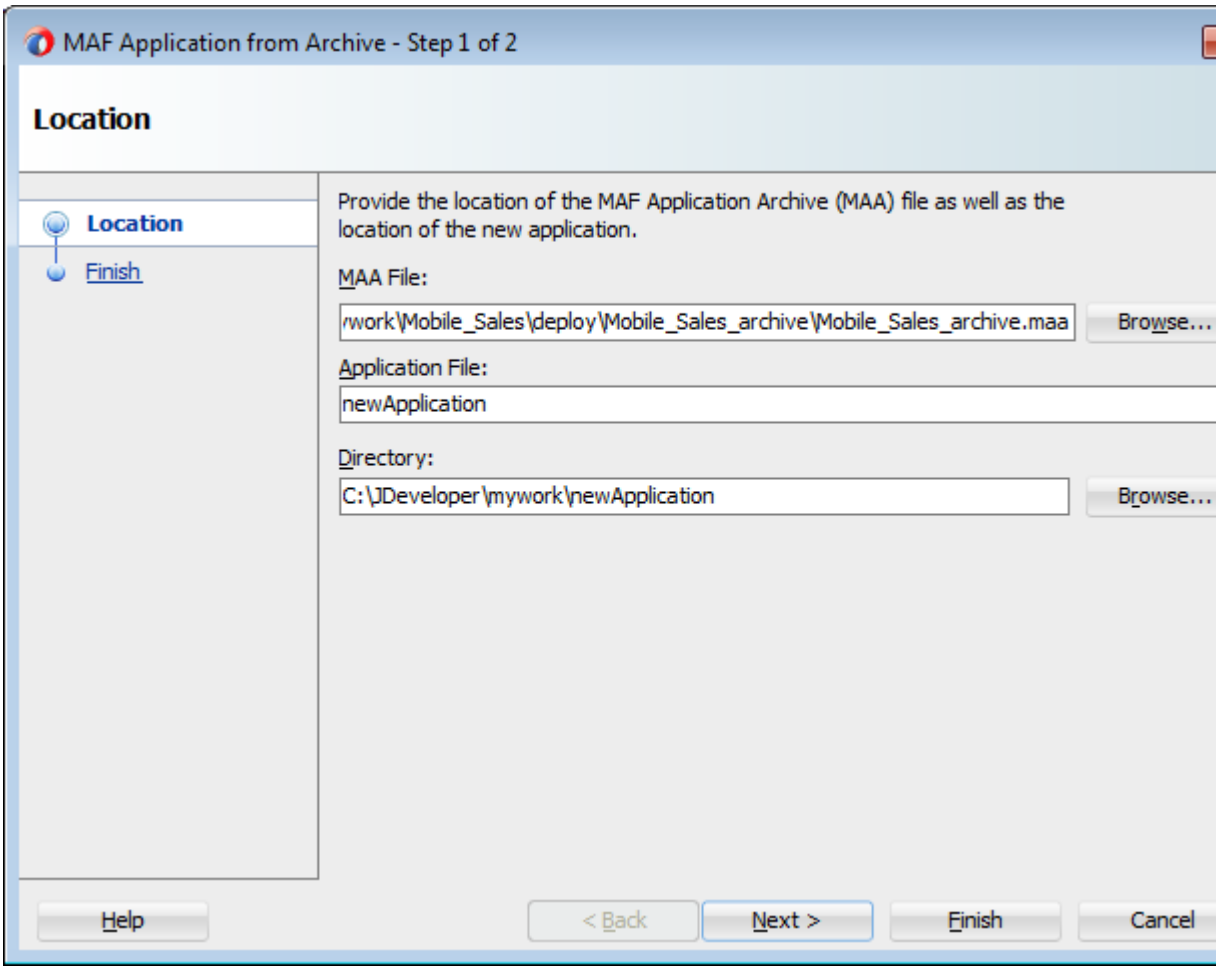
1. Select **File** and then **New**.
2. In the New Gallery, select **Applications** and then **MAF Application from Archive File**.

   Note:

   Alternatively, you can select **File**, then **Import**, and then **MAF Application from Archive File**.

3. In the Location page, select **Browse** in the MAA File field to navigate to the location of the MAA file.
4. If needed, perform the following, or accept the default values:
   a. Enter a name for the mobile application derived from the `.maa` file in the Application File field, as shown in Figure 25-37.
   b. Click **Browse** to retrieve the directory of the mobile application.

Figure 25-37 Entering the Directory Location



5. Click **Next** to review the import summary information and then click **Finish**.

25.9.2 What Happens When You Import a MAF Application Archive File

MAF creates an application folder into which it unpacks relevant directories and their contents. For every JAR file within the Projects directory, MAF creates a project folder, and unpacks the contents of the JAR files into the appropriate project folder.

MAF performs the following after you import an `.maa` file:

1. Creates an application folder.
2. Unpacks the workspace container (`.jws`) file from the `.maa` file to the application file and renames it per the user-specified value.
3. Unpacks the `adf` directory and its contents to the application folder. This directory is renamed `.adf`.
4. Unpacks the `META-INF` directory and its contents and places them in a `src` directory in the application folder.
5. Unpacks the `ExternalLibs` directory and its contents to the application folder.

Note:

While any of the external resources contained in this directory are available in the mobile application that has been packaged as an `.maa` file (and imported into the application), the references to these resources will be invalid for a mobile application derived from the `.maa` file.

6. Unpacks the `resources` directory to the application folder.
7. Unpacks all folders that contain FARs (or other libraries) that are internal to the original mobile application. MAF preserves the original locations of these artifacts.
8. For each JAR file within the `Projects` directory of the original mobile application, MAF performs the following:
   - Creates a project folder under the application directory that corresponds to the name of the JAR file (but without the `.jar` extension).
   - Unpacks the contents of the JAR files into the appropriate project folder. MAF includes the following in these project folders:
     - The original `.jpr` file.
     - The standard directories, such as `META-INF`, `public_html`, `src`, and `adfmsrc`.
     - The contents of the `ExternalLibs` directory.

Note:

While any of the external resources contained in this directory are available in the MAF project that has been packaged with the imported `.maa` file, the references to these resources will be invalid for an existing project, or a project created by importing the `.maa` file.

     - The `classlib` directory, which contains any Java classes packaged in a JAR file.

Note:

If the `.maa` file includes a `classlib` directory, then MAF adds all of the JAR files from this directory as library dependencies in the newly created mobile application.

25.10 Deploying MAF Applications from the Command Line Using OJDeploy

You can deploy iOS or Android applications from JDeveloper without starting the JDeveloper IDE using the OJDeploy command line tool. Command line deployment can serve as a tool for testing, as well as a means of deploying applications using a script.

After you have created iOS or Android deployment files using Deployment Profile Properties editor, you can use OJDeploy to deploy applications in the headless mode to iOS simulators and iOS-powered devices (through iTunes), or as iOS bundles (`.ipa` and `.app` files), or Feature Archive JAR files. Likewise, OJDeploy enables you to deploy applications to both Android emulators and Android-powered devices, or deploy them as an Android application package (`.apk`) file or as Feature Archive JAR files.

Note:

To use OJDeploy on a Mac, add the following line to the `ojdeploy.conf` file:

```
SetSkipJ2SDKCheck true
```

This file is located at: *`jdev_install`*`/jdeveloper/jdev/bin`

At present, you cannot use the OJDeploy command line tool to deploy to the Universal Windows Platform.

MAF downloads and installs the Gradle build tool the first time that you deploy a MAF application to the Android platform. If your development machine is located behind a corporate firewall, configure the Gradle proxy settings so that MAF can successfully download, install, and configure Gradle. See How to Configure Gradle Proxy Settings.

The following commands enable you to deploy MAF deployment profiles:

- `deployToDevice`—Deploys an application to iOS- or Android-powered devices. For iOS applications, this command is used in debugging scenarios where the application is deployed to a device using iTunes. For more information, see How to Distribute an iOS Application to the App Store.
- `deployToSimulator`—Deploys an application to an iOS simulator (as an `.app` file) or Android emulator. You can only deploy a mobile application to an iOS simulator on an Apple computer.
- `deployToPackage`—Deploys an iOS application as an `.ipa` file or an Android application as an `.apk` file. You can only package an application as an `.ipa` file on an Apple computer.
- `deployToFeatureArchive`—Deploys a Feature Archive to a JAR file.
- `deployToApplicationArchive`—Packages a mobile application as a MAF Application Archive (`.maa`) file.

You use these commands in conjunction with the `ojdeploy` command line tool, OJDeploy arguments, and OJDeploy options as follows:

```
ojdeploy deployToSimulator -profile <profile name> -workspace <jws file location>
```

Note:

OJDeploy commands and arguments are case-sensitive.

Table 25-5 lists the OJDeploy arguments that you use to modify the MAF deployment commands.

Tip:

Using the `-help` option with any command (such as `ojdeploy deployToSimulator -help`) retrieves usage and syntax information.

Table 25-5 OJDeploy Arguments for MAF Deployments

Argument	Description
`-profile`	The name of the Android or iOS deployment profile. For example: `ojdeploy deployToSimulator -profile iosDeployProfile ...`

`-workspace`	The full path to the mobile application workspace container (`.jws`) file. For example:
	`... -workspace /usr/jsmith/mywork/Application1/Application1.jws`
	To package a mobile application as a mobile Application Archive:
	`ojdeploy deployToApplicationArchive`
	`-profile applicationArchiveProfile`
	`-workspace /usr/jdoe/Application1/application1.jws`

`-project`	For the `deployToFeatureArchive` command, you must provide the name of the project (that is, a view controller project) that contains the Feature Archive deployment profile. For example:
	`ojdeploy deployToFeatureArchive`
	`-profile farProfileName`
	`-project ViewController`
	`...`

`-buildfile`	The full path to a build file for batch deploy.
`-buildfileschema`	Print XML Schema for the build file.

In addition to the arguments listed in Table 25-5, you can also use OJDeploy options described in the "Command Usage" section of *Developing Applications with Oracle JDeveloper*.

Note:

The following options are not supported:

- `-forcerewrite`
- `-nocompile`
- `-nodatasources`
- `-nodepdependents`
- `-outputfile`
- `-updatewebxmlejbrefs`

Table 25-6 provides examples of how to use the OJDeploy options with the MAF deployment commands.

Table 25-6 OJDeploy Options for MAF Deployments

Option	Description

`-clean`	Deletes all files from the project output directory before compiling. For example:
	`ojdeploy deployToSimulator`
	`-profile iosDeployProfile`
	`-workspace /usr/jsmith/jdeveloper/mywork/Application1.jws`
	`-clean`

`-stdout, -stderr`	Redirects the standard output and error logging streams to a file for each profile and project. For example:
	`ojdeploy deployToSimulator -profile iosDeployProfile`
	`-workspace /usr/jsmith/jdeveloper/mywork/Application1.jws`
	`-clean`
	`-stdout /usr/jsmith/stdout/stdout.log`
	`-stderr /usr/jsmith/stderr/stderr.log`

Table 25-7 lists the macros used with the `deployToApplicationArchive` command:

Table 25-7 Macros Used with MAF Application Archive Packaging

Macros	Description
`workspace.name`	The name of the application workspace container file (without the `.jws` extension).
`workspace.dir`	The directory of the application workspace container (`.jws`) file.
`profile.name`	The name of the profile being deployed.
`profile.dir`	The default deployment directory for the profile.
`base.dir`	Override the current OJDeploy directory using this parameter. You can also override the current OJDeploy directory using the basedir attribute in the build script.

26

Understanding Secure Mobile Development Practices

This chapter describes how Mobile Application Framework provides protection from common security risks identified by the Open Web Application Security Project (OWASP).

This chapter includes the following sections:

- Weak Server-Side Controls
- Insecure Data Storage on the Device
- Insufficient Transport Layer Protection
- Side-Channel Data Leakage
- Poor Authorization and Authentication
- Broken Cryptography
- Client-Side Injection From Cross-Site Scripting
- Security Decisions From Untrusted Inputs
- Improper Session Handling
- Lack of Binary Protections Resulting in Sensitive Information Disclosure

26.1 Weak Server-Side Controls

Mobile applications being vulnerable to attacks on the backend services that store their data, enforce access control using server-side applications instead of using client applications running on mobile devices.

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. See the following:

- Understanding Web Service Security Concepts in *Understanding Oracle Web Services Manager*
- Web Service Security Standards in *Understanding Oracle Web Services Manager*

26.2 Insecure Data Storage on the Device

Mobile application design can make local files accessible to users, thereby exposing sensitive data stored on the local file system of the device. MAF provides an API that encrypts device database and local data stores to secure data on a device.

Shortcomings in the design of a mobile application can make local files accessible to users, thereby exposing sensitive data stored on the local file system of a device. 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.

26.2.1 Encrypting the SQLite Database

Locally stored data is protected by an embedded SQLite database that is not shared by applications. Access to the database is restricted to the application that created it and authenticated users.

The MAF 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 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 How to Encrypt and Decrypt the Database. For a sample application, see the StockTracker sample in the `PublicSamples.zip`, as described in MAF Sample Applications.

Note:

Always use the `GeneratedPassword` utility. Do not hard-code the key.

26.2.2 Securing the Device's Local Data Stores

On both the iOS and Android platforms, store files in the local file system programmatically using the `getDirectoryPathRoot` method of the `adfmfJavaUtilties` class. Files that require security can be encrypted and decrypted using Java cryptographic APIs.

You can store files in the local file system programmatically on both the iOS and Android platforms using the `getDirectoryPathRoot` method of the

`adfmfJavaUtilties` class. 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 Application directory of the device 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 Accessing Files Using the getDirectoryPathRoot Method. See also the File System Basics in *File System Programming Guide*, available from the iOS Developer Library (`https://developer.apple.com/library/`).

26.2.3 About Security and Application Logs

A device that is synchronized with a desktop computer displays log files; ensure that no sensitive data is written to log files.

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 Side-Channel Data Leakage.

26.3 Insufficient Transport Layer Protection

Mobile applications can be secured with SSL or TLS when accessing data over a provider network, although they may use no protocol with WiFi. MAF provides a set of web service policies that support SSL to secure endpoints used by a mobile applications.

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. See 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. See Supporting SSL.

26.4 Side-Channel Data Leakage

Screen shots, key stroke logging, debugging messages, clipboard copying and open-in functionality, temporary directories, and third-party libraries can cause data leakage. Not logging credentials or personal identifiable information, reviewing files when debugging, and removing debugging messages before publishing the application can prevent 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.

26.5 Poor Authorization and Authentication

Weak authentication mechanisms and client-side access control compromise security. MAF application features that require secure access must have it enforced by the server.

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. See How to Designate the Login Page.

All features in a MAF application that require secure access must enable security, as described in 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. See How to Configure Access Control.

26.6 Broken Cryptography

Encryption is compromised by broken implementations and incorrect usage of algorithms. Store the key away from the encrypted data, and use platform-specific file encryption API or other trusted sources to ensure successful encryption.

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 Insecure Data Storage on the Device. Also, apply SSL to create secure web service calls as described in Insufficient Transport Layer Protection. MAF uses Oracle Access Manager for Mobile and Social IDM SDK for secure handling of credentials.

26.7 Client-Side Injection From Cross-Site Scripting

MAF encoding protects applications against Cross-Site Scripting injections, and application sandboxing provides protection against Cross-Site Request Forgery. Disabling the access of application features to the native container can also secure applications.

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`.

26.7.1 Protecting MAF Applications from Injection Attacks Using Device Access Permissions

Secure a MAF application by configuring it in a way that URI access to device capabilities, such as its phone or email, is limited.

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 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 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. See *Java API Reference for Oracle Mobile Application Framework*.

26.7.2 About Injection Attack Risks from Custom HTML Components

Using HTML, which delivers dynamic HTML content through an EL binding, to create a custom user interface component in a MAF AMX page may open an application to injection attacks. HTML and JavaScript must be properly encoded to secure a feature.

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. See How to Use Verbatim Component and How to Use an Output HTML Component. See also Security Decisions From Untrusted Inputs.

26.7.3 About SQL Injections and XML Injections

Validating and encoding all data stored in the local database, and encoding and validating XML and HTML content processed by an application prevents SQL 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.

26.8 Security Decisions From Untrusted Inputs

Prompting for additional authorization, providing additional steps to launch sensitive applications, validating all communication with untrusted third-party applications on the server, and using the JSON encoding API secures applications on both the iOS and Android platforms.

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. See Invoking MAF Applications Using a Custom URL Scheme. See also 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 *Java API Reference for Oracle Mobile Application Framework*.

26.9 Improper Session Handling

Configuring a session timeout value lower than the server-side session timeout, using complex tokens with expiry configured, and configuring a time value for feature session and a timeout that forces user re-authentication are some of the best practices for 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 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.

26.10 Lack of Binary Protections Resulting in Sensitive Information Disclosure

Storing API keys and sensitive business logic on the server, not hard-coding passwords, and not writing sensitive information to the log files protects data such as API keys, passwords, and sensitive business logic.

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 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 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.

27

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:

- Introduction to MAF Security
- About the User Login Process
- Overview of the Authentication Process for MAF Applications
- Configuring MAF Connections
- Configuring Security for MAF Applications
- Allowing Access to Device Capabilities
- Enabling Users to Log Out from Application Features
- Using MAF Authentication APIs
- Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL
- Registering SSL Certificate File Extensions in a MAF Application

27.1 Introduction to MAF Security

MAF provides a number of ready-to-use features that facilitate the development of secure MAF applications. These include a default login page for secured application features, support for anonymous users, and constraints to restrict access to users with specific roles or privileges.

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 MAF application without a successful login, you can enable users to access a MAF 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 MAF 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 MAF 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 MAF 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.

See How to Enable Application Features to Require Authentication and 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. See What You May Need to Know About the Access Control Service.

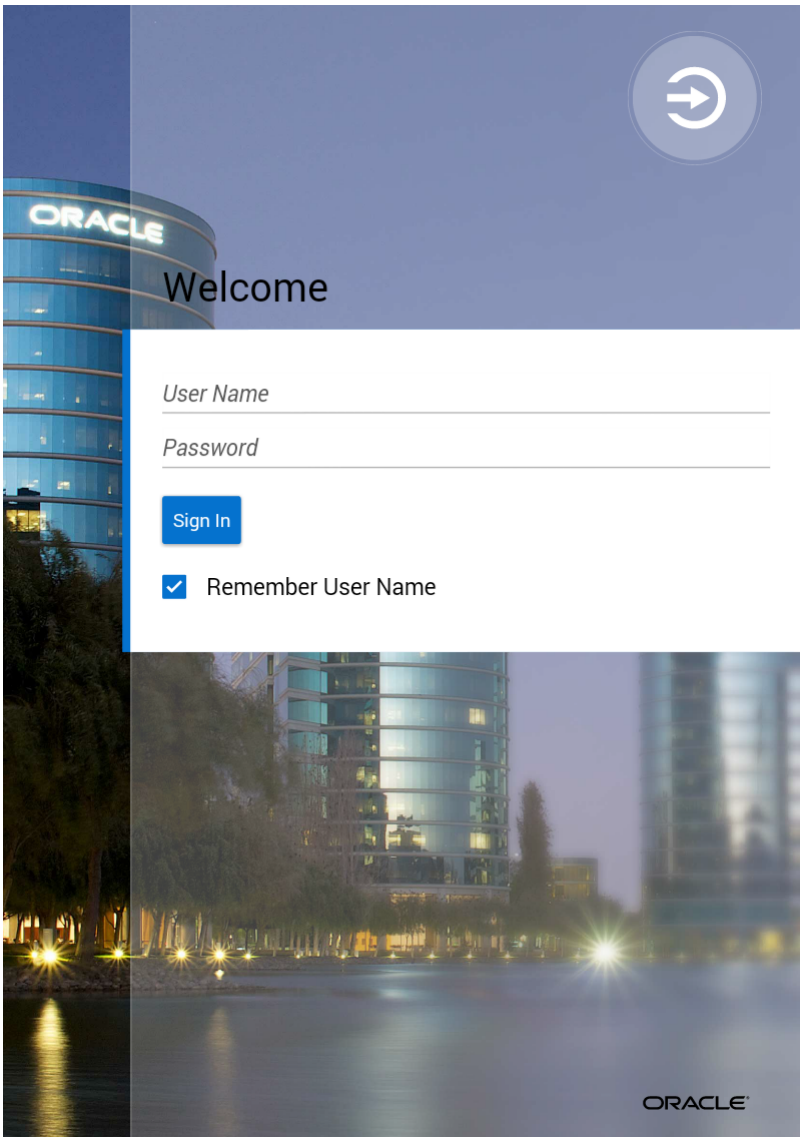
27.2 About the User Login Process

MAF displays the web view, page, or user interface component that a user wants to access after the user has logged in on the web login page and has been authenticated. MAF presents challenges until a login is effected, and the login times out after the configured interval.

From the end-user perspective, the login process is as follows:

1. MAF presents a web view of a login page, shown in Figure 27-1 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 default login page when they launch the application.

Figure 27-1 Default Login Page



Note:

As described in 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 time-out when one of the application features that use the connection to the authentication server has been activated.

27.3 Overview of the Authentication Process for MAF Applications

MAF supports the HTTP Basic, OAuth, and Web SSO authentication protocols, and authenticates application users against a remote login server. Upon a successful remote authentication, MAF stores the credentials locally, so that subsequent authentications become local.

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, if using 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 27-1 summarizes the login configuration options of a MAF application. The connectivity mode depends on the selected authentication protocol.

Table 27-1 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 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.

27.4 Configuring MAF Connections

An application feature can be secured by defining at least one connection to the application login server.

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.

27.4.1 How to Create a MAF Login Connection

Create a connection type for logins, and enable both local and remote authentication. Follow the steps in the task to create a login server connection using the Create MAF Connection dialog.

As Figure 27-2shows, you can use the **Create MAF Connection** dialog to select the connection type and, depending on the connection type, enable both local and remote

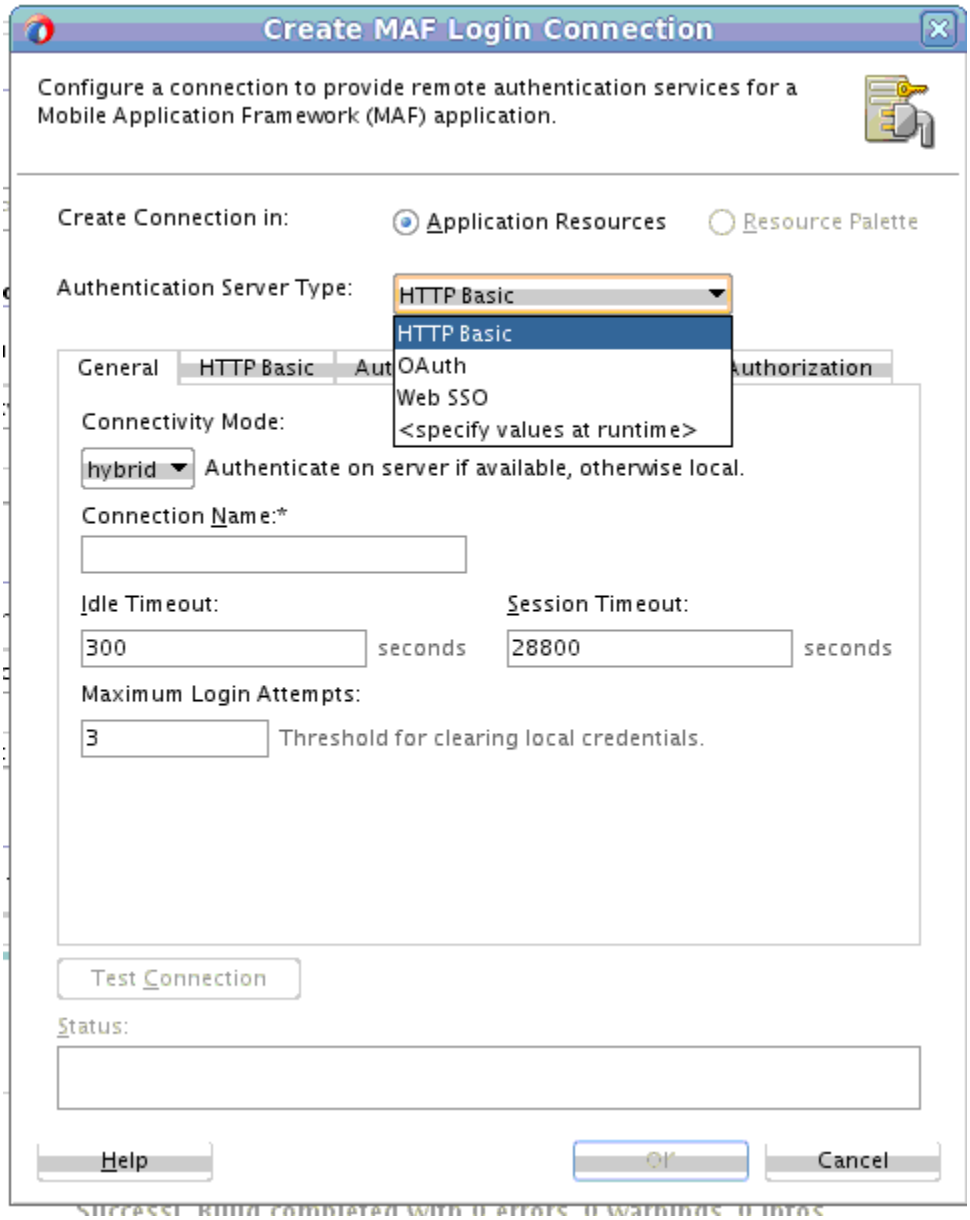
authentication (hybrid). Depending on application requirements, you can configure a connection to servers that support the following authentication protocols:

- 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 27-2 Configuring Authentication

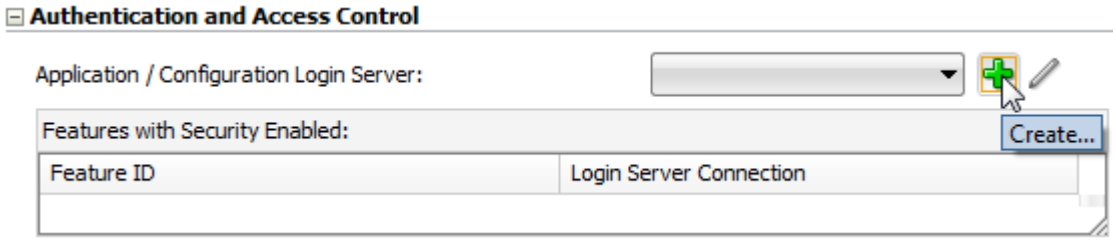


To create a login server connection:

1. Perform one of the following actions.

   - In the Navigator, expand the **Descriptors** node and then **ADF META-INF**, and double-click **maf-application.xml**. Then, in the overview editor for the `maf-application.xml` file, expand the **Security - Authentication and Access Control** section and click **Create**, as shown in Figure 27-3.

Figure 27-3 Adding a Server Connection



- Alternatively, select **Connections** in the **New Gallery** and then **MAF Login Server Connection**.

2. In the **Create MAF Login Connection** dialog, select the desired **Authentication Server Type**.

3. Configure the connection type as described in the following sections.

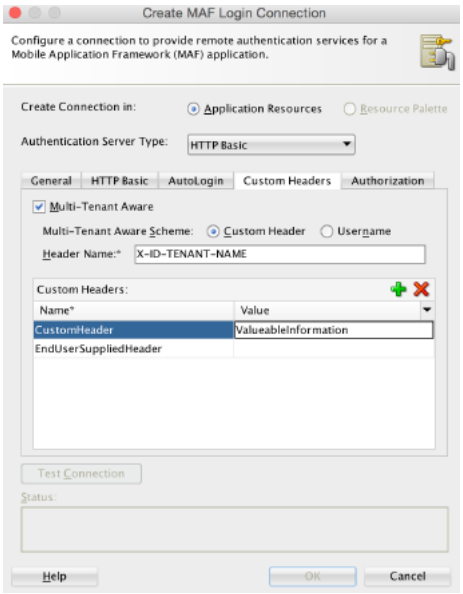
   Note that options that appear in the dialog with an asterisk are required fields. The dialog enables the **Test Connection** button only after all required fields are completed. This button appears only when basic authentication is selected in the dialog.

27.4.2 How to Create a Multi-Tenant Aware MAF Login Connection

A MAF application connection can support a hosted application feature that can be shared by different organizations or tenants. Follow the steps in the task to configure a multi-tenant aware connection to servers.

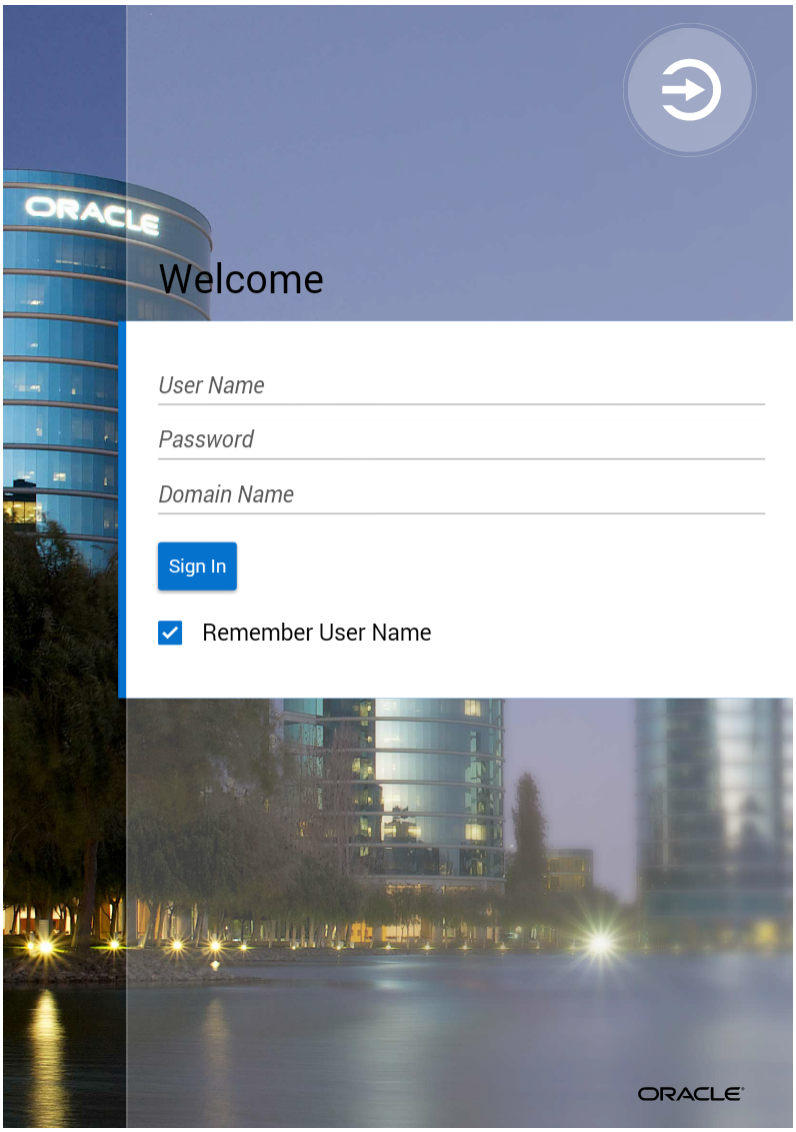
As Figure 27-4 shows, you can use the **Create MAF Login Connection** dialog to create a MAF application connection that supports the notion of multi-tenancy, where an application includes a hosted application feature that can be shared by different organizations (tenants), but can appear as though it is owned by a particular tenant. You can configure a multi-tenant aware connection to servers that support the HTTP Basic authentication protocol.

Figure 27-4 Configuring a Multi-Tenant Aware Connection



As Figure 27-5 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 27-5 Default Login Page for Multi-Tenant Aware Connection

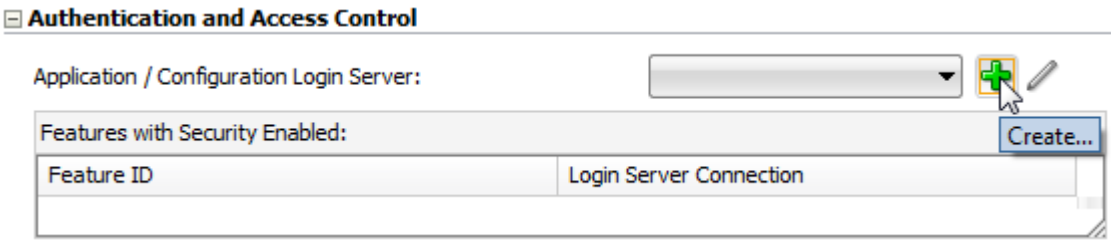


To create a multi-tenant aware login server connection:

1. Perform one of the following actions.

   - In the Navigator, expand the **Descriptors** node and then **ADF META-INF**, and double-click **maf-application.xml**. Then, in the overview editor for the `maf-application.xml` file, expand the **Security - Authentication and Access Control** section and click **Create**, as shown in Figure 27-6.

Figure 27-6 Adding a Server Connection



- Alternatively, select **Connections** in the **New Gallery** and then **MAF Login Server Connection**.

2. In the **Create MAF Login Connection** dialog, select an **Authentication Server Type** that supports multi-tenant login.
3. Click the **Custom Header** tab and configure the following, as shown in Figure 27-4.

   - **Multi-Tenant Aware**—Select to define multi-tenancy awareness for the MAF application connection. See also What Happens When You Create a Multi-Tenant Aware Connection.
   - **Multi-Tenant Aware Scheme**—Select the scheme used to propagate the tenant domain ID to the authentication server. Select **Custom Header** (default) to send as a separate header. The **Username** option supports backward compatibility and sends the tenant ID as part of the user ID (called username mingling).
   - **Header Name**—Enter the tenant header name expected by the authentication server. For example, to solicit the tenant ID from the user during login, enter the multi-tenant header name: `X-ID-TENANT-NAME`. As Figure 27-5 shows, the default login page will prompt the user to enter the domain name
   - **Custom Headers**—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 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 How to Configure Login Credentials Programmatically Prior to Authentication.

4. Configure the connection type as described in the following sections

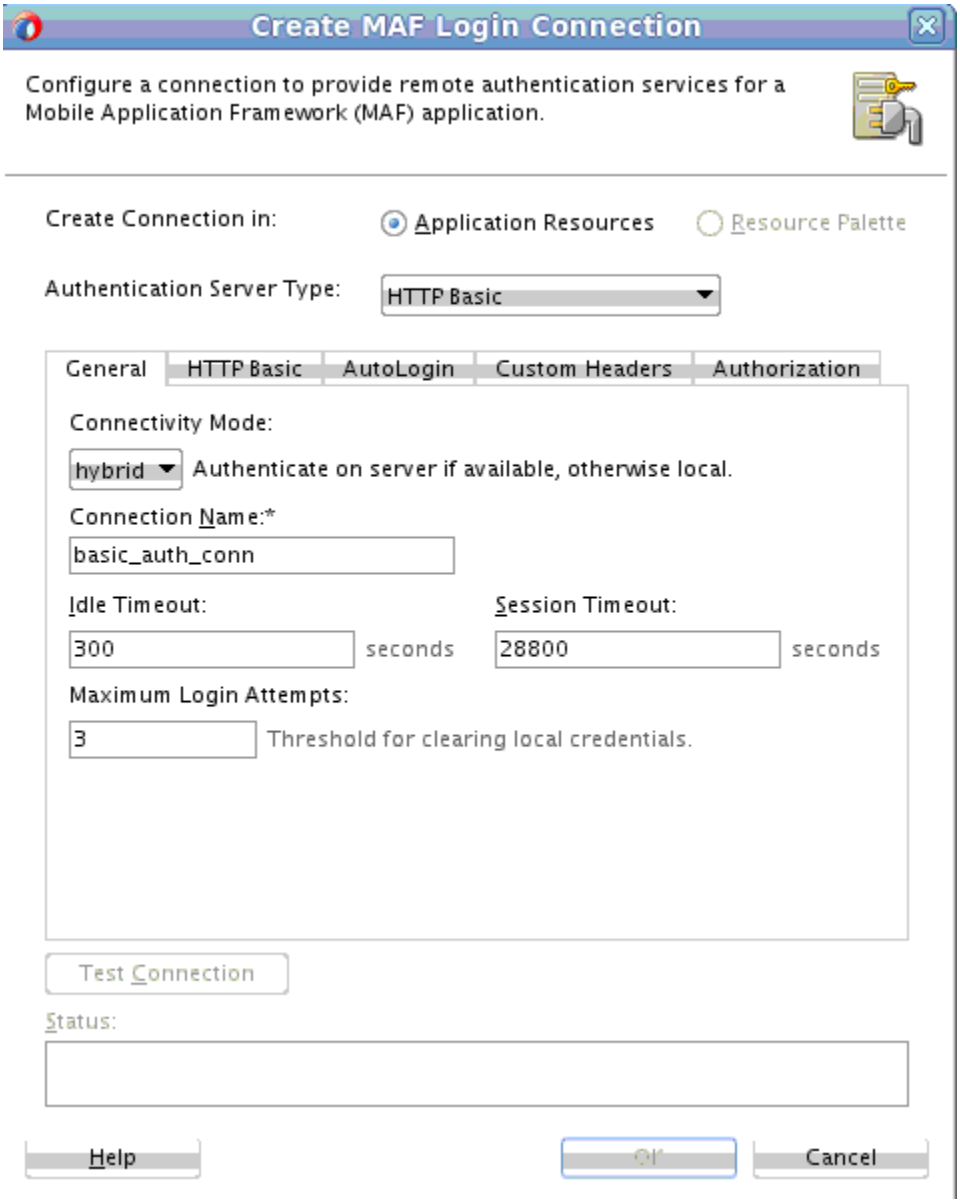
   Note that options that appear in the dialog with an asterisk are required fields. The dialog enables the **Test Connection** button only after all required fields are completed. This button appears only when basic authentication is selected in the dialog.

27.4.3 How to Configure Basic Authentication

Configuration of a connection for basic authentication requires the selection of **HTTP Basic** for **Authentication Server Type**. Follow the steps in the task to configure basic authentication using the Create MAF Login Connection dialog.

As Figure 27-7 shows, you can select the **HTTP Basic** authentication server type in the **Create MAF Login Connection** dialog to configure a connection for basic authentication.

Figure 27-7 Configuring Basic Authentication



To configure basic authentication:

1. In the **Create MAF Login Connection** dialog, select **HTTP Basic** for **Authentication Server Type**.

   For information about opening the **Create MAF Login Connection** dialog, see How to Create a MAF Login Connection.

2. In the **General** tab, define the following:

   - **Connectivity Mode**—Select the type of authentication, as described in Table 27-1.

- **Connection Name**—Enter a name for the connection.
- **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).
- **Maximum Login Attempts**—Set the maximum number of failed login attempts allowed for a user before local credentials will be cleared. By default, MAF grants a user three unsuccessful login attempts before it clears the user's locally stored credentials and contacts the remote login server for subsequent login attempts. Subsequent to contacting the remote server, the user is allowed an indefinite number of login attempts.

  Note that when the user fails login attempts for the number of times specified, the local credentials will be cleared and MAF will thus execute authentication against the server. This ensures that users can login with a new password after an administrator changes their password and it is not yet stored on a device. Where local authentication is allowed, the password will be stored securely on a device when the user successfully logs into the server connection.

Note:

MAF clears locally stored user credentials even when the application feature is configured to use local authentication.

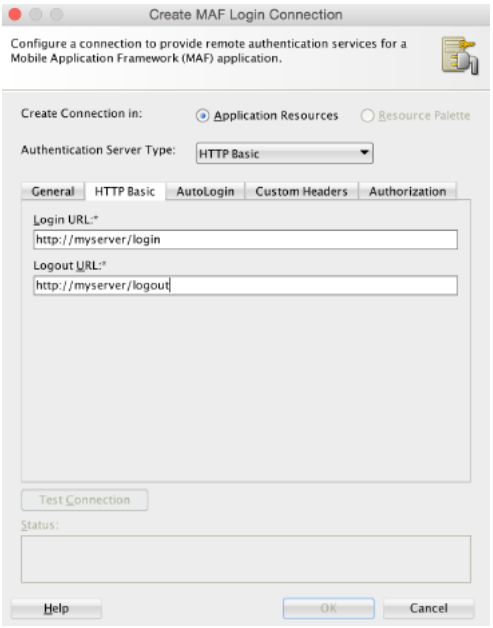
---

3. Click the **HTTP Basic** tab and configure the following, as shown in Figure 27-8.
   - **Login URL**—Enter the login URL for the login page.

     The login URL should not be a login page on the remote server, but rather a page that is secured and presents the HTTP Basic user name/password challenge. The login URL must point to a web resource that does not result in file transfer when requested; it must not point to a file resource.
   - **Logout URL**—Enter the logout URL for the authentication server.

     The logout URL may be the same as the login URL, but alternatively may be a URL to the remote server that performs additional actions on the session, such as invalidating it. The logout URL must point to a web resource that does not result in file transfer when requested; it must not point to a file resource.

Figure 27-8 Configuring Basic Authentication



4. Optionally, click the **Custom Header** tab and configure the following, as shown in Figure 27-4.

   - **Custom Headers**—Enter the name of any custom headers required to perform authentication. See also 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 How to Configure Login Credentials Programmatically Prior to Authentication.

   - **Multi-Tenant Aware**—You can define multi-tenancy awareness for the MAF application connection by selecting this option. For more information, see How to Create a Multi-Tenant Aware MAF Login Connection.

5. Click the **Auto Login** tab and configure the parameters as described in How to Store Login Credentials.

6. Click the **Authorization** tab and configure the parameters as described in How to Configure Access Control.

7. Click the **General** tab, and then click **Test Connection**.

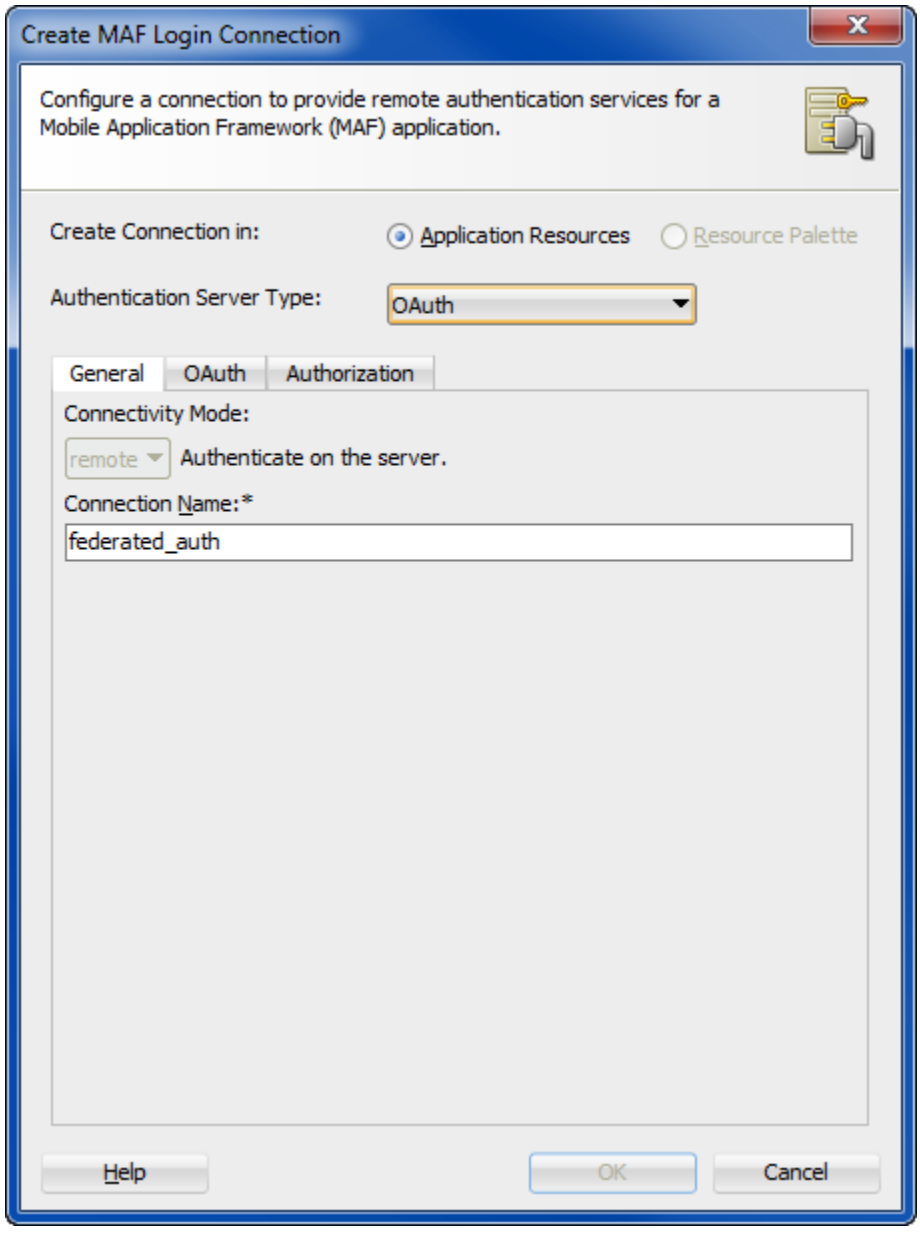
8. Click **OK**.

27.4.4 How to Configure OAuth Authentication

An application can access protected data or services stored on a remote server. Configure the server to use the `OM_PROP_OAUTH_OAUTH20_SERVER` property key, and then follow the steps in the task to configure authentication with an OAuth server.

Use the **Create MAF Login Connection** dialog, shown in Figure 27-9, to configure how to configure the access of an application to protected data or services stored on a remote server.

Figure 27-9 Configuring OAuth



Before you begin:

Configure the server to use the `OM_PROP_OAUTH_OAUTH20_SERVER` property key.

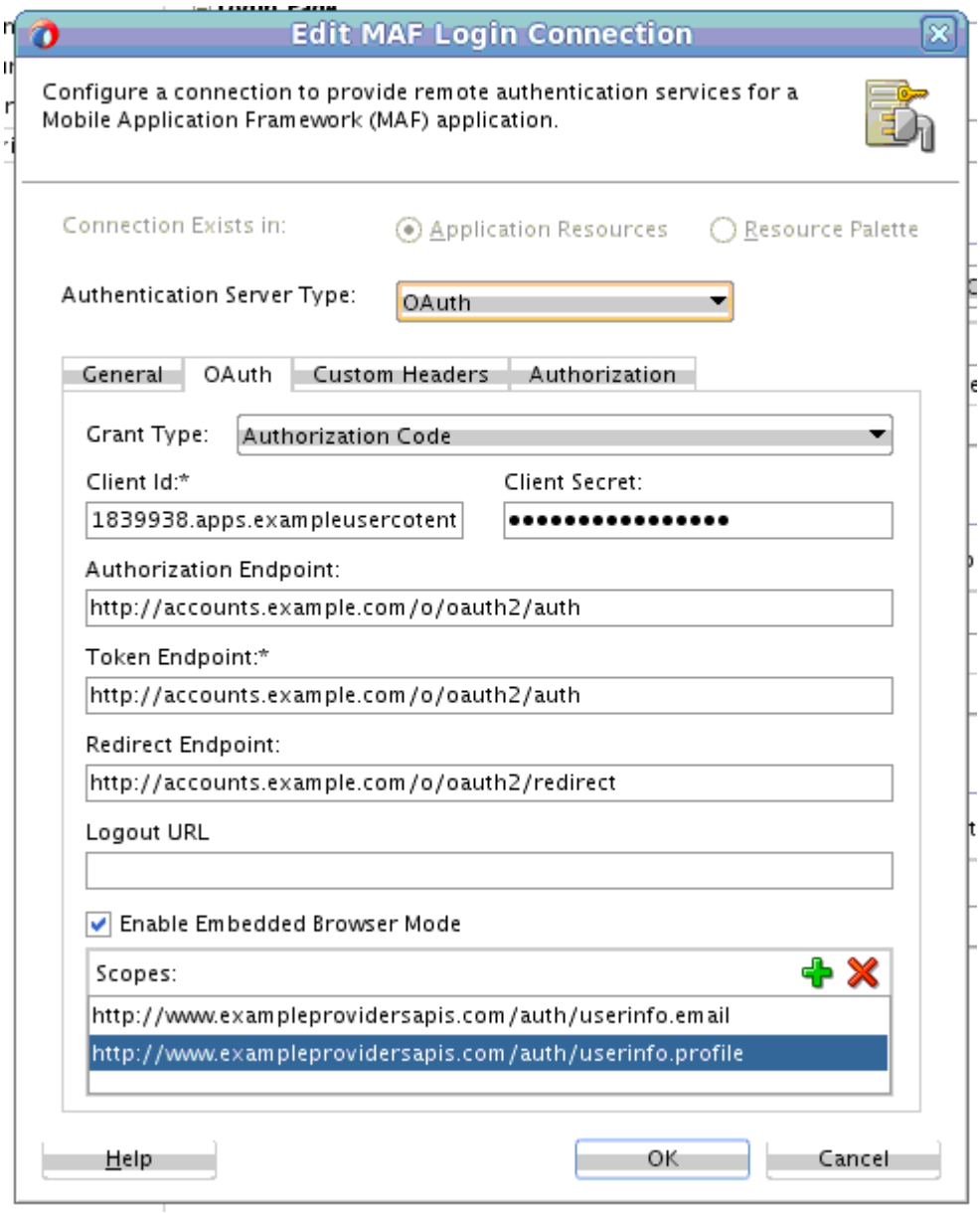
To configure authentication with an OAuth server:

1. In the **Create MAF Login Connection** dialog, select **OAuth** for **Authentication Server Type**.

   For information about opening the **Create MAF Login Connection** dialog, see How to Create a MAF Login Connection.

2. In the **General** tab, configure the following:
   - **Connection Name**—Enter a name for the connection.
3. Click the **OAuth** tab and configure the following, as shown in Figure 27-10:
   - Select the appropriate grant type from the **Grant Type** drop-down list.
     - Select **Authorization Code** when you want the server login page to display.
     - Select **Resource Owner Credentials** when you want the MAF application to display the default login page, or custom login page, when one is configured.
     - Select **Client Credentials** when you want the MAF application to access resources anonymously without requiring an user ID or user credentials.
   - Enter enter the **Client Identifier** and, optionally, enter a connection password value in the **Client Secret** field.
   - Enter the **Redirect Endpoint** of the authorization server and the URIs for the endpoints for the **Authorization Server Endpoint** itself and the **Token Endpoint**.
   - Enter the **Logout URL** to redirect to upon user logout. This field is mandatory and the URL parameters are determined by the specific authentication provider.
   - Select **Enable Embedded Browser Mode** when you want the login page to display within the embedded browser within the application. Deselect to display the login page in an external browser. Note that when SSO is desired, you must deselect this option to force the application to use the external browser.

Figure 27-10 Configuring the Client ID and Endpoints



4. Click the **Authorization** tab and configure the parameters as described in How to Configure Access Control.

27.4.5 How to Configure Single Sign-On in a MAF Application

MAF supports single-sign on (SSO) and federated SSO in MAF applications deployed to all platforms (Android, iOS, and the Universal Windows Platform).

The identity provider that you use to configure SSO can be a third-party identity provider hosted in your corporate domain, such as an instance of Azure Active Directory, or whatever identity provider has already been configured for the backend service that your MAF application connects to such as, for example, Oracle Mobile Cloud Service (MCS).

For MAF applications that you deploy to the Universal Windows Platform that need to support MCS SSO, you need to register a Cordova plugin in your application that

allows access to MCS SSO. Configure entries in the plugin's `plugin.xml` file that allow this access, as demonstrated by the following example:

```
<config-file target="package.appxmanifest" parent="/Package/Applications/Application/
uap:ApplicationContentUriRules">
        <!-- Allow Webviews access to server so the token relay header can be
injected -->
        <uap:Rule Match="http://*.oracle.com:7777/*/*/*/*" Type="include"/>
      </config-file>
```

For more information about creating Cordova plugins and using them in your MAF application, see Introduction to Using Plugins in MAF Applications and Registering Additional Plugins in Your MAF Application.

Use of a token relay service where MAF includes OAuth access tokens in outgoing requests to services secured with the OAuth2 token policy is also supported. You enable this support by selecting the **Parse Token Relay Response** checkbox when you configure the SSO connection. MAF then parses the JavaScript Object Notation (JSON) responses from the login success URL that conform to the access token format proposed by the Internet Engineering Task Force in RFC 6749 - The OAuth 2.0 Authorization Framework. MAF retrieves the OAuth access token from the JSON response and includes it in MAF application requests to secured services. The following example shows a valid JSON response that MAF parses to retrieve the OAuth access token. MAF ignores additional custom fields that may be included in the JSON response.

```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

> **Note:** If you select the **Parse Token Relay Response** checkbox, MAF includes "`format=JSON`" as a query parameter in the login URLs. You do not need to specify this query parameter in the URL.

If you use the identity provider configured for MCS, you provide the client ID for the OAuth Consumer issued by the MCS mobile backend ID that your MAF application connects to as a query parameter in the connection URL that you configure in your MAF application. The following example shows a subset of the entries generated in MAF application's `connections.xml` file for a connection that enables the application to use SSO that is configured in an MCS mobile backend.

```
<Contents>
  <login url="http://yourmcsinstance.com/sso/token?clientID=C490B55...FO51"/>
  <logout url="http://yourmcsinstance.com/sso/appLogout"/>
  <loginSuccessUrl url="http://yourmcsinstance.com/sso/success/token?
clientID=C490B55...FO51">
    <parseTokenRelayResponse value="true"/>
  </loginSuccessUrl>
  <loginFailureUrl url="http://yourmcsinstance.com/sso/appError"/>
  …
</Contents>
```

To configure single sign-on authentication:

1. In the **Create MAF Login Connection** dialog, select **Web SSO** for **Authentication Server Type**.

   For information about opening the **Create MAF Login Connection** dialog, see How to Create a MAF Login Connection.

2. In the **General** tab, configure the following:

   - **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).

3. Click the Web SSO tab and configure the following URLs that enable successful and unsuccessful logins, as shown in Figure 27-11:

   - **Login URL**—Enter the URL that prompts the user to enter credentials when the MAF application navigates to it. The login URL must point to a web resource that does not result in file transfer when requested; it must not point to a file resource. If creating a login connection to MCS, the login URL must include a query parameter set to the client ID specified by the MCS MBE that your MAF application connects to. Obtain the value of this client from MCS.
   - **Logout URL**—Enter a server side URL that logs out the user by terminating the server session. The logout URL must point to a web resource that does not result in file transfer when requested; it must not point to a file resource.

     If configuring a connection to MCS, make sure that the logout URL conforms to the following format so that redirect works as expected after logout.

     ```
     //A production instance of MCS
     https://{hostname}/logout.html?end_url=/mobileui
     //A development instance of MCS
     https://{hostname}/oam/server/logout
     ```

   - **Login Success URL**—Enter a target URL to redirect the user to after the user successfully authenticates.

     The login success URL can be the same as the login URL. For example, if the login URL and login success URL is `http://www.mysite.com`, then when the user points the browser to `http://www.mysite.com`, the browser will redirect to the login page for the site before it redirects upon successful authentication back to `http://www.mysite.com`. Then, when MAF detects the page named by the login success URL, MAF completes the login process and activates the requested feature. Thus, the contents of the login success URL page will not be displayed to the user and user will have access to the MAF feature. As with the URL you enter for **Login URL**, include a query parameter set to the client ID specified by the MCS MBE if you want create a login connection to an MCS MBE.

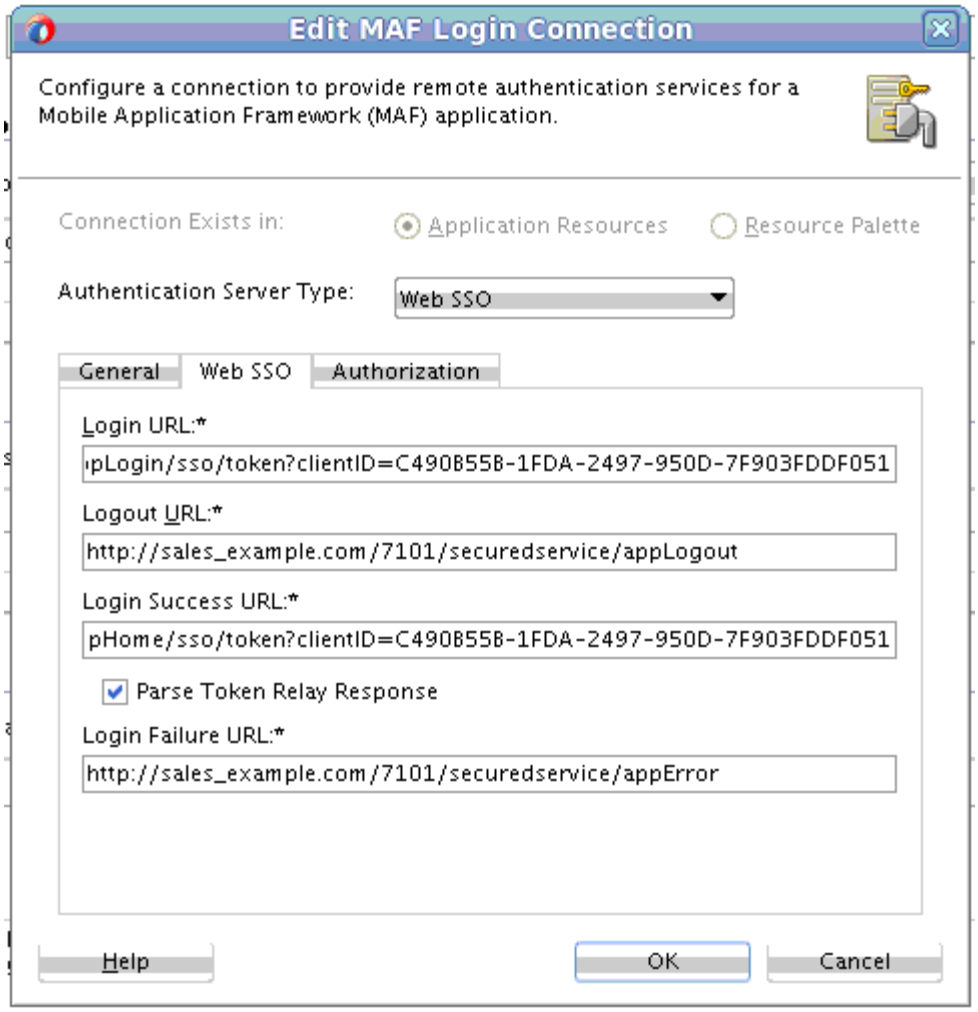
     – **Parse Token Relay Response** – Select this checkbox to parse the login success URL JSON response for the OAuth token. Your MAF application uses the retrieved OAuth token to access secured services. Selecting this checkbox generates `<parseTokenRelayResponse value="true"/>` in your application's `connections.xml` file.

- **Login Failure URL**—Enter a URL to redirect the user to after unsuccessful authentication. Alternatively, when no failure URL exists, enter any URL.

  As the browser loads the login failure URL, MAF first detects the error and returns control to the application. This is useful when the MAF application limits the number of user login attempts. In this case, MAF will redirect the user to the login failure URL after the user fails to authenticate by the last allowed attempt.

  In the case where no failure URL exists, it is permissible to enter any URL. In this case, authentication will be terminated either when the user clicks **Cancel** on the login page or when login times out due to no user action for a given period of time (the inactivity timeout is two minutes).

Figure 27-11 Configuring the Authentication URLs



4. Click the **Authorization** tab and configure the parameters, as described in How to Configure Access Control.

27.4.6 How to Configure a Placeholder Connection for MAF Application Login

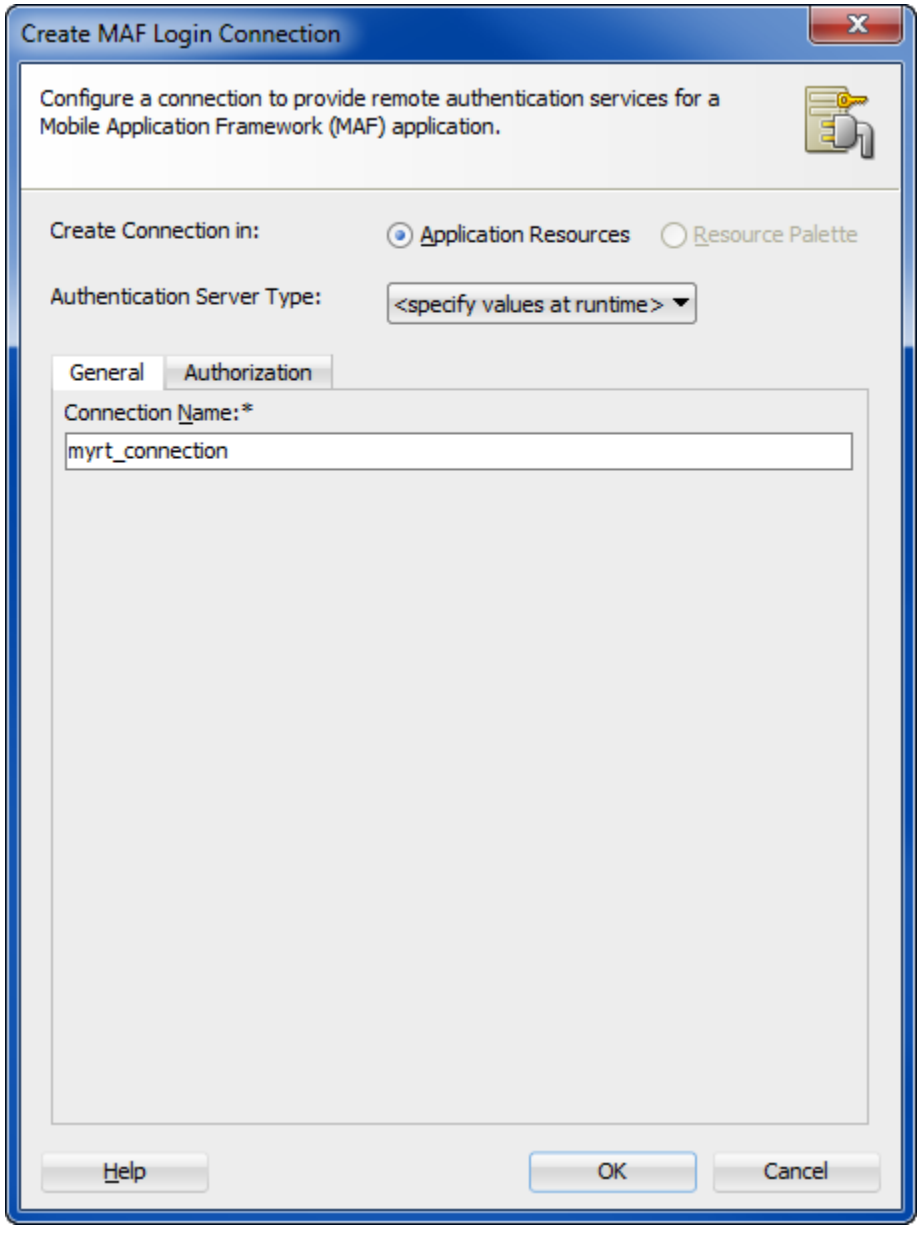
If all connection attributes for a MAF application login are not known at design time, use the `AdfmfJavaUtilities.updateSecurityConfigWithURLParameters`

API to define a placeholder connection. Follow the steps in the task to configure a placeholder connection for definition at runtime.

As Figure 27-12 shows, you can use the **Create MAF Login Connection** dialog to create a named connection during development and populate the login attributes to fully define the connection at runtime. This connection type is particularly useful when the connection attributes are not all known at design time.

Application developers must use `AdfmfJavaUtilities.updateSecurityConfigWithURLParameters` API to fully define the placeholder connection created at design time, as described in How to Update Connection Attributes of a Named Connection at Runtime.

Figure 27-12 Configuring a Placeholder Connection



To configure a placeholder connection for definition at runtime:

1. In the **Create MAF Login Connection** dialog, select **Specify Values at Runtime** for **Authentication Server Type**.

   For information about opening the **Create MAF Login Connection** dialog, see How to Create a MAF Login Connection.

2. In the **General** tab, enter a name for the connection.

   This identifier will be used by the application developer to identify the connection to update, as described in How to Update Connection Attributes of a Named Connection at Runtime.:

3. Click the **Authorization** tab and configure the parameters, as described in How to Configure Access Control.

27.4.7 How to Update Connection Attributes of a Named Connection at Runtime

Use the `overrideConnectionProperty` and `updateSecurityConfigWithURLParameters` methods of the `AdfmfJavaUtilities` class to define or redefine the connection attributes of an existing connection. Use both methods with the `clearSecurityConfigOverrides` and `updateApplicationInformation` APIs of the `AdfmfJavaUtilities` class.

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::=Approvals&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 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 MAF Sample Applications.

27.4.8 How to Store Login Credentials

Opt to have MAF store credentials, user name, and auto login so as to access the MAF application without having to log in. Use the **AutoLogin** option of the Create MAF Login Connection dialog to select credential storing options.

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:

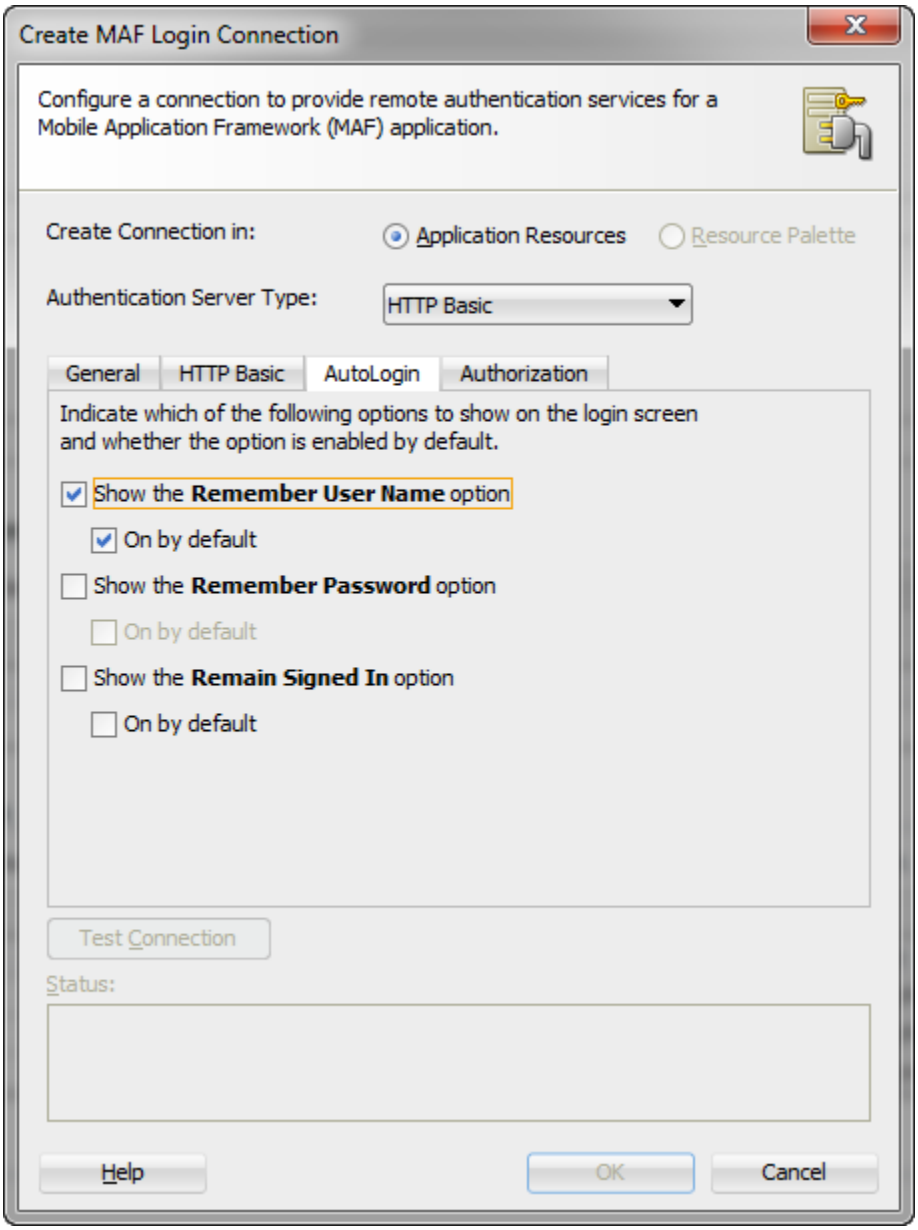
- remember user name—MAF caches the user name and populates the **username** field of the login page. After the user enters the password and confirms by tapping the login button, MAF replays these credentials to the authentication server.
- remember credentials—MAF caches the user credentials and populates the user name and password fields of the login page. 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 27-13 shows, you can use the **AutoLogin** page of the **Create MAF Login Connection** dialog 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 users.

Figure 27-13 Caching User Credentials



27.4.9 What Happens When You Create a Connection for a MAF Application

Application connection information is available in the `connections.xml` file which can be bundled with the application. If the file is hosted for the Configuration Service, MAF checks for updated configuration information every time the application starts.

MAF aggregates all of the connection information in the `connections.xml` file (located in the Application Resources panel of the Applications window under the **Descriptors** and **ADF 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, JDeveloper 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>
```

See Lookup Defined in the connections.xml File in *Developing Fusion Web Applications with Oracle Application Development Framework*.

27.4.10 What Happens When You Create a Multi-Tenant Aware Connection

When you create a multi-tenant aware connection, MAF sets the value of the `<isMultiTenantAware>` element to true in the `connections.xml` file.

After you complete the **Create MAF Login Connection** dialog with the **Multi-Tenant Aware** option enabled, MAF populates the `connections.xml` file with the `<isMultiTenantAware>` element set to true. In multi-tenant connection, 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 27-5. After a successful login (which includes entering the correct tenant ID), MAF stores the tenant ID in the local credential store.

27.4.11 What You May Need to Know About the Login Connection Configuration

When the login URL is defined in the `connections.xml` file or in a program, the URL must point to a web resource that does not result in file transfer upon a request.

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.

27.4.12 What You May Need to Know About Multiple Identities for Local and Hybrid Login Connections

Within an application lifecycle, users can log in and log out using multiple identities in the local and hybrid login connection modes.

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.

27.4.13 What You May Need to Know About Migrating a MAF Application and Authentication Modes

When a MAF application is migrated, ensure that the authentication mode defined in the `maf-feature.xml` file is defined by the `authenticationMode` attribute in the `connections.xml` file.

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. Use the audit rules of JDeveloper, which detect the presence of the `credentials` attribute, to assist you in removing it from `maf-feature.xml`.

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.

27.4.14 What You May Need to Know About Custom Headers

When a MAF login connection is created and custom headers have been defined, the `connections.xml` file is populated with the `customAuthHeaders` element and individual header subelements. You can configure header values at runtime using the `OverrideConnectionHandler` API.

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. For information about using the API to configure headers, see 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.

27.4.15 What Happens at Runtime: When MAF Calls a REST Web Service

When a MAF application calls a REST web service, MAF authenticates the user against the login server if the user has been authenticated locally. MAF executes the application request to the REST web service if the user is authenticated.

After a user is authenticated locally, MAF silently authenticates the user against the login server when the MAF application calls a REST web service. After the user's credentials are authenticated, MAF executes the application 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.

27.4.16 What You May Need to Know About Injecting Basic Authentication Headers

MAF injects a basic authentication header into HTTP requests to enable application features to access secure resources.

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 Authorization tab of the Create MAF Login Connection dialog 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>
```

27.4.17 What You May Need to Know About Web Service Security

Although web services have no login pages, MAF injects the required credentials for access into the header of a web service call. The credentials injected are the credentials that MAF persists after they are used to log in to the authentication server.

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 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 `adfCredentialStoreKey` attribute of the login server. 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 **Properties** window to update the `adfcredentialStoreKey` attribute of the `<Reference>` element with the name configured for `adfCredentialStoreKey` attribute of the login server connection. Alternatively, you can add or update the attribute using the Source editor.

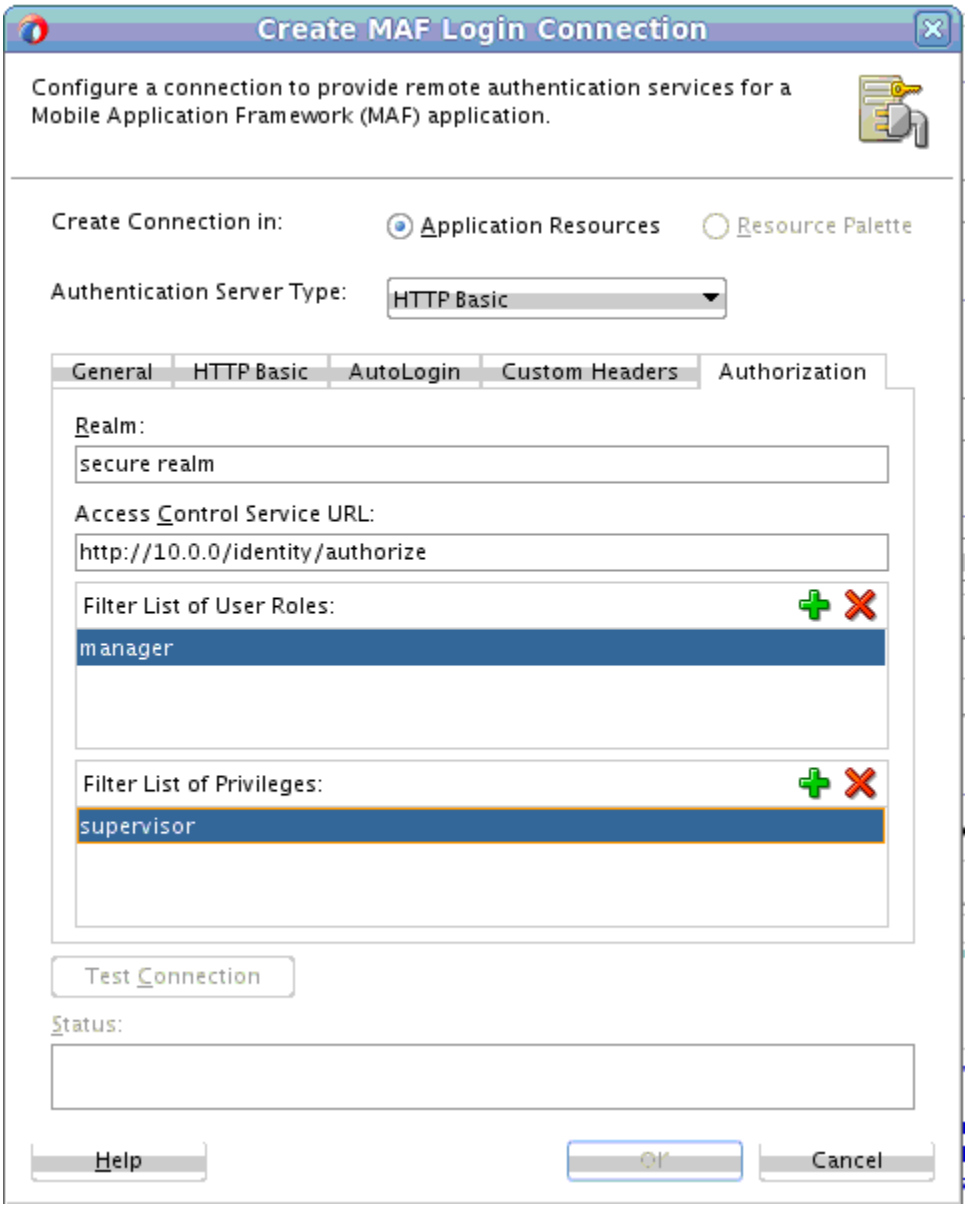
See What You May Need to Know About Credential Injection.

27.4.18 How to Configure Access Control

If an application contains secured components, configure it to implement and host Access Control Service. Follow the steps in the task to configure access control so that only the application features for which the user has authorization becomes available.

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 27-14 shows the Authorization page of the **Create MAF Login Connection** dialog that you would use to configure the MAF application to support access control.

Figure 27-14 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 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 **Create MAF Login Connection** dialog, click the **Authorization** tab.

   For information about opening the **Create MAF Login Connection** dialog, see How to Create a MAF Login Connection.

2. On the **Authorization** page, complete the authorization requirements, as shown in Figure 27-14.

   - **Realm**—Specify a requesting realm, if known at the time of development, for inclusion in the basic authentication header that MAF injects into HTTP requests.
   - **Access Control Service URL**—Enter the URL that is the endpoint for the Access Control Service (ACS).
   - **Filter List of User 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.
   - **Filter List of Privileges**—Enter the privileges checked by the application feature.

27.4.19 What You May Need to Know About the Access Control Service

Access Control Service is a RESTful web service which returns the roles or privileges, or both, for a request message for a given user.

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 following example 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 What You May Need to Know About Credential Injection.

Note:

You must implement and host the ACS service; MAF does not provide this service.

27.4.20 How to Alter the Application Loading Sequence

MAF invokes the Access Control Service after a user successfully authenticates against a login connection that defines the ACS endpoint. When a secured application feature invokes ACS, MAF fetches the user constraints for all the application features associated with the application login connection.

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 27-14. 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 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 *Java API Reference for Oracle Mobile Application Framework*.

27.4.21 How to Configure Login Credentials Programmatically Prior to Authentication

Using the `OverrideConnectionHandler` API in the `oracle.adfmf.framework.api.AdfmfJavaUtilities` class, set connection values programmatically before the MAF application invokes the login connection to authenticate a user. Follow the steps in the task to override the connection values programmatically.

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 Create MAF Login Connection dialog 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 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 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 a `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);
    }
}
```

27.5 Configuring Security for MAF Applications

Use the overview editors for the `maf-feature.xml`, and `maf-application.xml` files, and the Create MAF Login Connection dialog to configure security for MAF applications.

You configure security using the overview editors for the `maf-feature.xml` and `maf-application.xml` files, as well as the **Create MAF Login Connection** dialog. The overview 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.

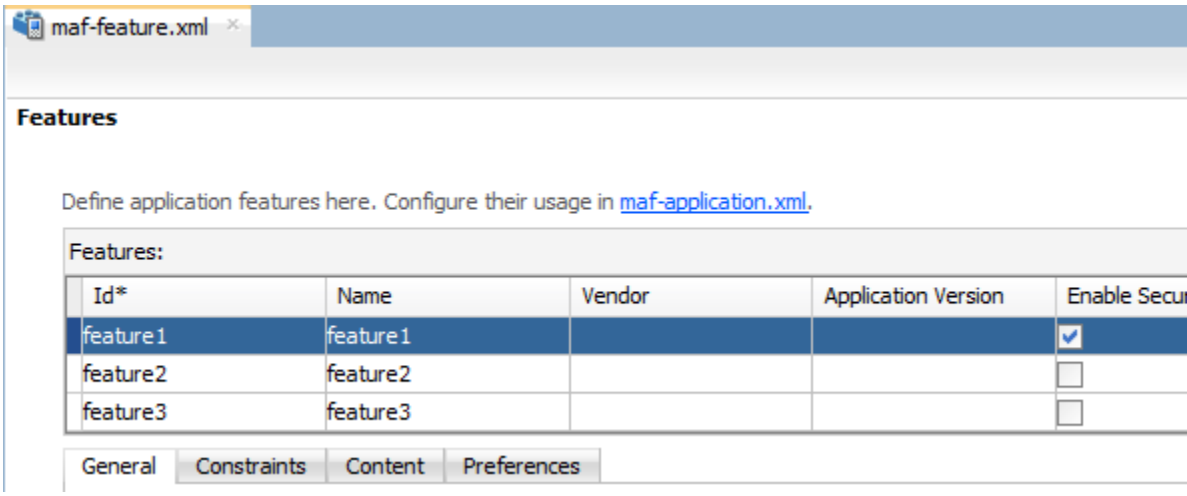
27.5.1 How to Enable Application Features to Require Authentication

Configure security for application features using the Security page of the `maf-application.xml` overview editor. Follow the steps in the task to designate user access for an application feature.

You can define each application feature to participate in security. You perform the security configuration using the Security page of the `maf-application.xml` overview editor.

The `maf-feature.xml` overview editor, shown in Figure 27-15, enables you to designate which application features participate in security.

Figure 27-15 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 Enabling a Core Plugin in Your MAF Application.

To designate user access for an application feature:

1. In the Navigator, in the user interface project, expand **Application Sources** and then **META-INF** folder nodes, and then double-click **maf-feature.xml**.
2. In the overview editor for the `maf-feature.xml` file, select an application feature listed in the **Features** table or click **Add** 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.

27.5.2 How to Designate the Login Page

After defining security for application features, use the Security page of the `maf-application.xml` overview editor to configure the login page, create a connection to the login server, and assign it to the application features for which security is enabled. Follow the steps in the task to designate the login page.

After you designate security for the application features, you use the **Security** page of the `maf-application.xml` overview editor, shown in Figure 27-16, 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.xml` file 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 MAF 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:

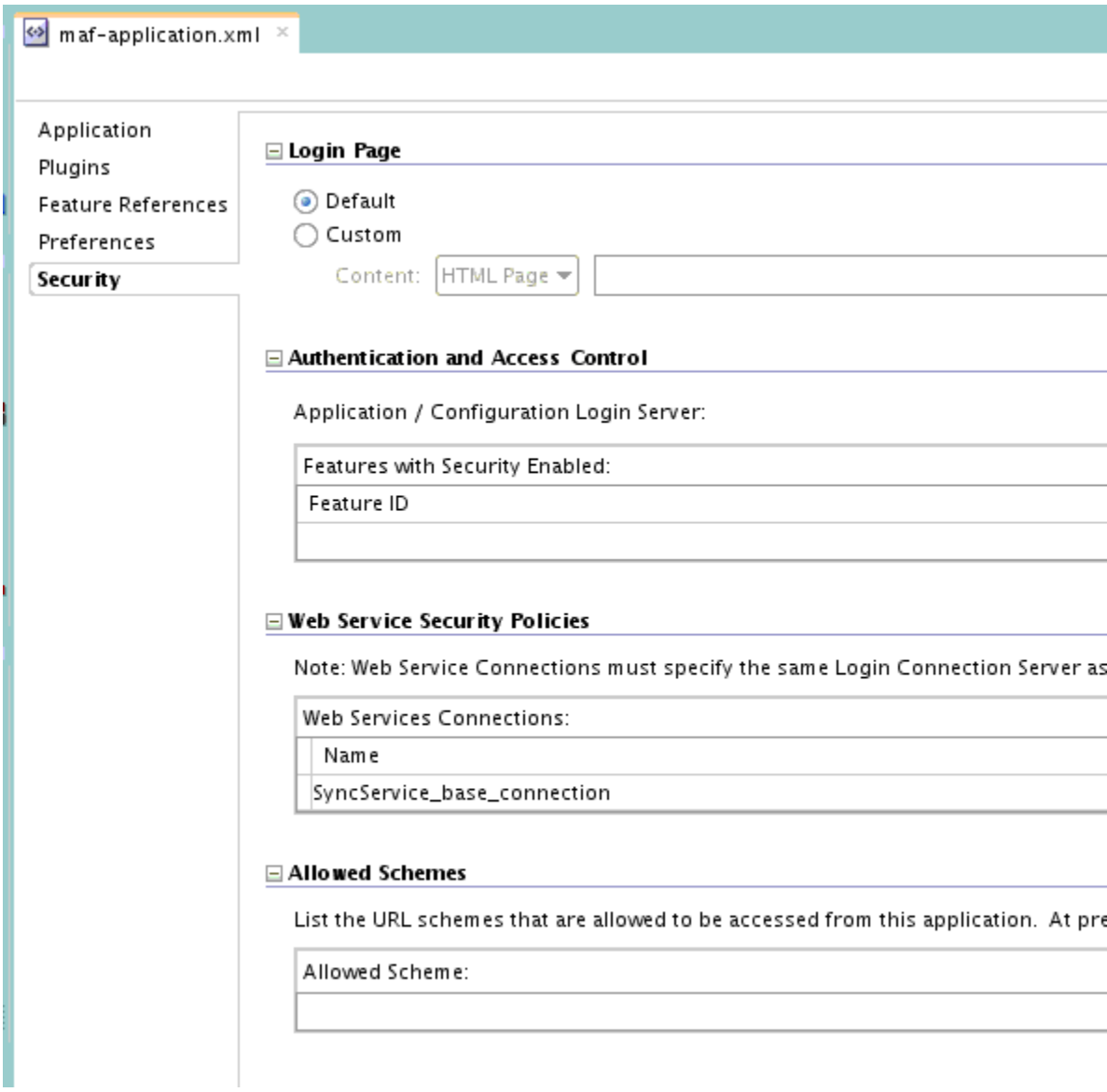
```
<adfmf:featureReference refId="feature1" loginConnRefId="BasicAuthentication"/>
<adfmf:featureReference refId="feature2" loginConnRefId="WebSSO"/>
```

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. See 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 27-16, is denoted as `<application login server>`. The **Properties** window for a Feature Reference notes this default option in its **Login Server Connection** drop-down menu as `<default> (application login server)`. You can select other connections that are defined for the MAF application using the **Create MAF Login Connection** dialog.

Figure 27-16 The Security Page of the maf-application.xml Overview Editor



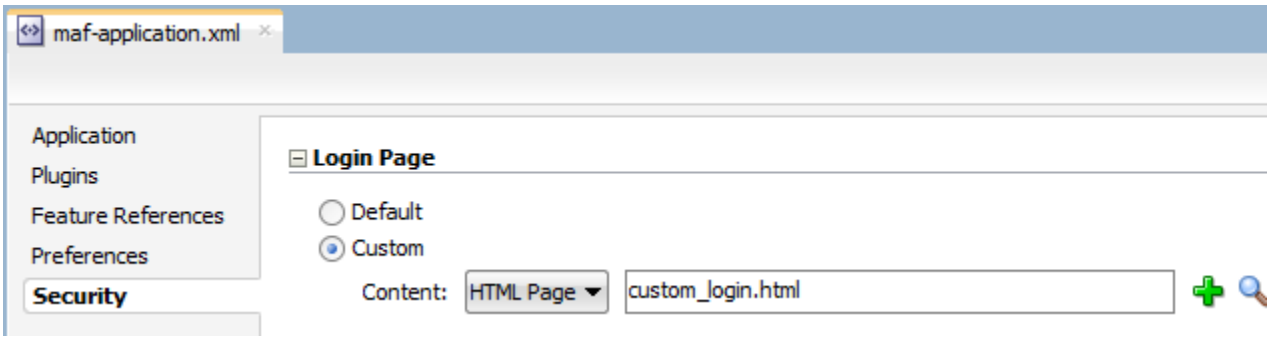
Before you begin:

If the MAF application uses a custom login page, add the file to the `public_html` directory of the application controller project (`JDeveloper\mywork\`*`Application`*`\ApplicationController\public_html`) to make it available from the **Web Content** node in the Application Navigator, as shown in Figure 27-17. See also How to Create a Custom Login HTML Page and What You May Need to Know About Selecting External Resources .

Add constraints for user privileges and roles, as described in About User Constraints and Access Control.

Provision an Access Control Service (ACS) server. See What You May Need to Know About the Access Control Service.

Figure 27-17 Adding a Custom Login Page



To designate the login page:

1. In the Navigator, expand the **Application Resources** panel, expand **Descriptors** and then **ADF META-INF** folder nodes, and then double-click **maf-application.xml**.
2. In the overview editor for the `maf-application.xml` file, click the **Security** navigation tab.
3. On the **Security** page, designate the type of login page:
   - Select **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. See 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. See The Custom Login Page and How to Create a Custom Login HTML.

     Tip:

     Rather than retrieve the location of the login page using the **Browse** function, you can drag the login page from the Application Navigator into the field.

27.5.3 How to Create a Custom Login HTML Page

Modifying the default login page, `adf.login.html`, and artifacts generated by the MAF deployment in the `www` directory creates a custom login page. Follow the steps in the task to create a custom login page, and to retrieve its location.

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 page is 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 within the `public_html` directory of the user interface project, such as `JDeveloper\mywork\`*`application name`*`\ApplicationController\public_html`.
2. Rename the login page.
3. Update the page.
4. In the Security page for the overview editor of the `maf-application.xml` file, select **Custom** and then click **Browse** to retrieve the location of the login page.

27.5.4 What You May Need to Know About Login Pages

When a feature registered with MAF is activated, the login process is executed. Upon successful authentication, the login page retrieves and navigates to the indicated destination.

The entry point for the authentication process to an application feature is the `activate` lifecycle event, described at Using 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.

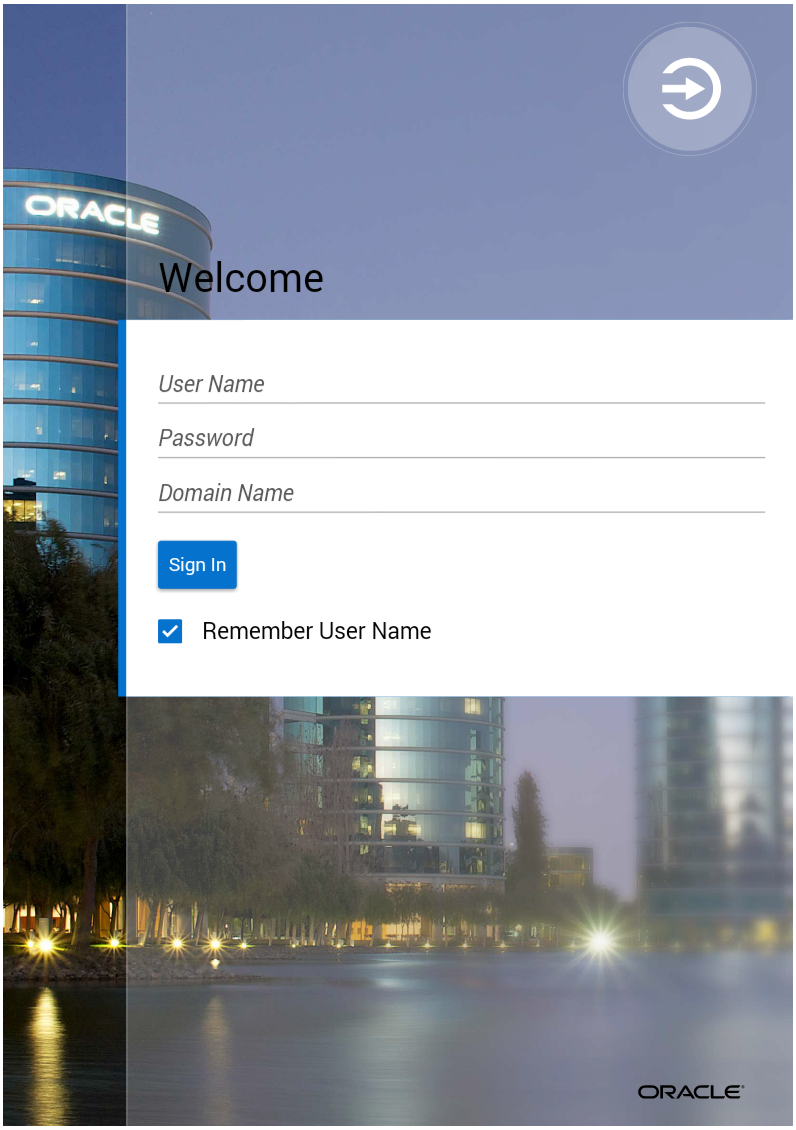
If you want to use Java classes or resources, such as resource bundles, from your login page using the built-in Cordova support, you must place these classes and resources in the `ApplicationController` project.

27.5.4.1 The Default Login Page

The default login page provided by MAF is a cross-platform page, written in HTML, with fields for credentials.

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 27-18 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 27-18 The Default Login Page with Multi-Tenant Domain Field



27.5.4.2 The Custom Login Page

A custom login page has the authentication mechanism and navigation control that is identical to the default login page. When a custom login page is added, JDeveloper adds the `<adfmf:login>` element and populates its child `<adfmf:LocalHTML>` element.

When you add a custom login page for a selected application feature using the overview editor for the `maf-application.xml` file, JDeveloper adds the `<adfmf:login>` element and populates its child `<adfmf:LocalHTML>` element, as shown in the following 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 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 How to Create a Custom Login HTML Page.

The following example 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"/>
```

27.5.5 What You May Need to Know About Login Page Elements

Every HTML login page should include the user interface elements listed in Table 27-2.

Table 27-2 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 checkbox	`oracle_access_auto_login_id`
remember credentials checkbox	`oracle_access_remember_credentials_id`
remember username checkbox	`oracle_access_remember_username_id`

Table 27-3 lists the recommended JavaScript code used by the `OnClick` event.

Table 27-3 JavaScript Used by the OnClick Event

Button	JavaScript
login button	`oracle_access_sendParams(this.id)`
cancel button	`oracle_access_sendParams(this.id)`

27.5.6 What Happens in JDeveloper When You Configure Security for Application Features

For every application feature with security enabled, JDeveloper adds a corresponding feature reference in the Features With Security Enabled table.

After an application feature has been designated to participate in security, JDeveloper updates the **Features With Security Enabled** table with a corresponding feature reference. If each of the referenced application features authenticate against the same login server connection defined in the `connections.xml` file, JDeveloper 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 JDeveloper updates each referenced application feature with a `loginConnReference` attribute (`<adfmf:featureReference refId="feature2" loginConnRefId="Connection2"/>`). See How to Enable

Application Features to Require Authentication. See also *Tag Reference for Oracle Mobile Application Framework*.

27.6 Allowing Access to Device Capabilities

MAF provides a set of Cordova core plugins that defines access to device capabilities. These plugins and any additional plugins that you register, enable the required permissions for device access.

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 enables any device access permissions that the application 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 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`.

27.7 Enabling Users to Log Out from Application Features

A user who logs out of an application feature with secured content can continue to access its unsecured content and features as an anonymous user. MAF permits a user multiple logins to an application using the same identity or different identities.

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 *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:

- 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 *Java API Reference for Oracle Mobile Application Framework*.

27.8 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 27-1 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");
…
```

27.9 Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL

A `cacerts` file in MAF enables deployment by identifying certificates from trusted sources. Follow the steps in the task to add private certificates to the MAF `cacerts` file when an application needs access to resources in a server that uses a self-signed certificate, or when an application needs custom certificates.

MAF provides a `cacerts` certificate file, the Java mechanism for HTTPS handshakes between the client application and the server. JDeveloper creates this file within the Application Resources `Security` folder (located at `JDeveloper\mywork \application name\resources\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. Users of your MAF application can install certificates if you configure your MAF application to facilitate the installation of certificates, as described in Registering SSL Certificate File Extensions in a MAF Application.

Before you begin:

It may be helpful to have an understanding of the contents of the `cacerts` file. See Migrating to New cacerts Files for SSL in MAF in *Installing Oracle Mobile Application Framework*.

You may also find it helpful to understand how JDeveloper creates the `cacerts` file. See About the Application Controller Project-Level Resources.

Refer to Java SE Technical Documentation (`http://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. The following example illustrates adding a single certificate to a `cacerts` file called `new_cert`.

```
keytool -importcert
        -keystore cacerts
        -file new_cert
        -storepass changeit
        -noprompt
```

Repeat this procedure for each certificate. Table 27-4 lists the keytool options

Table 27-4 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 common name (CN) of the certificate must match the hostname exactly.

e. Ensure that the customized certificate file is located within the `Security` directory (`JDeveloper\mywork\`*`application name`*`\resources\Security`) so that it can be read by the JVM.

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.

27.10 Registering SSL Certificate File Extensions in a MAF Application

Follow the steps in the task to register a certificate file extension in an application so as to enable two-way SSL communication. After registration, the user downloads the certificates at runtime, and extracts them for installation in the keystore of the application.

MAF supports the registration of file extensions for both server SSL and client SSL certificates. Users may need to restart their MAF application after installing a certificate.

Server SSL Certificate Extension

A server SSL certificate identifies the remote HTTPS server that your MAF application connects to. Register a server SSL certificate file extension to facilitate the installation of a certificate that is not in the `cacerts` file that MAF provides when you create a MAF application. This scenario frequently occurs when the HTTPS server that your MAF application connects to uses a self-signed certificate rather than a certificate issued by a root certificate authority. MAF currently supports the installation of server SSL certificates in MAF applications deployed to the Android and iOS platforms. This is not supported in MAF applications deployed to the Universal Windows Platform. For this platform, add the certificate to the `cacerts` file of the MAF application, as described in Creating Certificates to Access Servers That Use Self-Signed Certificates for SSL.

Client SSL Certificate Extension

A client certificate identifies a MAF application to a remote server. Register a client SSL certificate extension when you want users to install a client certificate into the keystore of the MAF application to enable authentication using a process known variously as two-way SSL, mutual authentication or two-way authentication. Once the user installs the client certificate, the MAF application can present it to a server so that a two-way SSL communication session performs authentication between the client and the server.

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 user can open the client certificate directly from the email attachment or URL used to distribute the client certificate.

How to Register an SSL Certificate File Extension

You register a certificate file extension in your application by entering the file extension in the appropriate field (depending on your use case) on the Application page of the overview editor of the `maf-application.xml` file.

To configure a MAF application to enable two-way SSL:

1. In the **Applications** window, expand the **Application Resources** panel.
2. In the **Application Resources** panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the **maf-application.xml** file and in the overview editor that appears, click the **Application** navigation tab.
4. In the **Application** page, select the appropriate option:

- Enter the file extension in the **Client SSL Certificate Extension** field if you want to facilitate the installation of a client certificate for use in a two-way SSL session.
- Enter the file extension in the **Server SSL Certificate Extension** field if you want to facilitate the installation of a server certificate that your MAF application uses to connect to a HTTPS server.

What Happens When You Register an SSL Certificate File Extension

JDeveloper writes the value that you enter in the input fields to the `maf-application.xml` file, as shown in Example 27-2.

At runtime, the user downloads the certificate(s) 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 user's device. For example, a MAF application user with an Android device downloads the certificate to the Android `Download` directory. Once downloaded, the user extracts (opens) the certificate in order to install it to the keystore of the MAF application. To complete this step for a client certificate, the user must enter a password provided by the administrator who distributed the client certificate. After the user has installed a client certificate in the keystore of the application, the MAF application can present it to a server to establish a two-way SSL session. A server certificate that an user installs can be used by the MAF application to establish an SSL session with a HTTPS server.

Example 27-2 Certificate File Extensions 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="ssltest" id="com.company.ssltest"
                  appControllerFolder="ApplicationController" listener-
class="application.LifeCycleListenerImpl"
                  client-ssl-certificate-extension="clientcert"
                  server-ssl-certificate-extension="servercert">
                 ….
```

28

Testing and Debugging MAF Applications

This chapter provides information on testing and debugging MAF applications.

This chapter includes the following sections:

- Introduction to Testing and Debugging MAF Applications
- Testing MAF Applications
- Configuring JDeveloper and MAF Applications to Debug Code
- Debugging MAF Applications Deployed on the Android Platform
- Debugging MAF Applications Deployed on the iOS Platform
- Debugging MAF Applications Deployed on the Universal Windows Platform
- Using and Configuring Logging in MAF Applications
- Measuring MAF Application Performance
- Inspecting Web Service Calls in a MAF Application

28.1 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, iOS, or Universal Windows Platform) that MAF supports.

MAF and the respective platforms provide tools that let you connect the JDeveloper 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 JDeveloper to an Android device or to an Android Virtual Device (AVD). The Universal Windows Platform and iOS provide similar tools.

The high-level steps to debug a MAF application include the following tasks:

1. Configure the deployment profile that you use to deploy the MAF application to the test environment so that it deploys the MAF application in debug mode.
2. Configure the MAF application's `maf.properties` file to enable debugging.
3. Deploy the MAF application to the test environment.
4. 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 JDeveloper 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).

28.2 Testing MAF Applications

You can test a MAF application on a mobile device, emulator or simulator.

There are two approaches to test 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.

28.2.1 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 "Test Accessibility on Your Device with VoiceOver" 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.

28.3 Configuring JDeveloper and MAF Applications to Debug Code

JDeveloper 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 JDeveloper, the debugging approach is different: you can use the JDeveloper 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. The following are the steps you need to take to use JDeveloper to debug the Java code in your MAF application:

To test or debug an application:

1. From JDeveloper's main menu, click **Run > Choose Active Run Configuration** to select an active run configuration.

2. In the **Applications** window, right-click on any file that you want to test and choose **Run**.

   Alternatively, choose **Debug** if you want to run the application with debugging enabled.

   Tip:

   You can also open any file in the MAF application in **Source** view, right-click on it, and then select **Run** or **Debug**.

   Note:

   If you are using an existing application that does not have the predefined set of run configurations, you can create new run configurations.

For more information, see What You May Need to Know About the Debugging Configuration.

28.3.1 What You May Need to Know About the Debugging Configuration

When a new MAF application is created, the creation wizard automatically configures the application properties for debugging. This includes the creation of default run configurations that may be used to run or debug the MAF application on an iOS simulator or Android emulator or device. These run configurations allow you to build, deploy, run, or debug a MAF application by clicking the JDeveloper **Run** or **Debug** buttons. When you click the **Run** or **Debug** button in JDeveloper and select a MAF run configuration, the deployment profile associated with the run configuration is executed to build and deploy the application to the targeted device. Once the application has been deployed, it automatically starts. If the **Debug** button was selected, the application will start with the debugger.

For information on how to create and edit run configurations, see Creating and Configuring a Run Configuration.

28.3.1.1 Creating and Configuring a Run Configuration

To create a new configuration or to modify an existing one, complete the **Edit Run Configuration** dialog (see Figure 28-1) as follows:

1. From JDeveloper's main menu, click **Application > Project Properties** to open the **Project Properties** dialog.
2. In the **Project Properties** dialog, select the **Run/Debug** node from the tree on the left.

   Alternatively, choose **Run > Choose Active Run Configuration > Manage Run Configurations**.
3. Create a new configuration or modify an existing one.

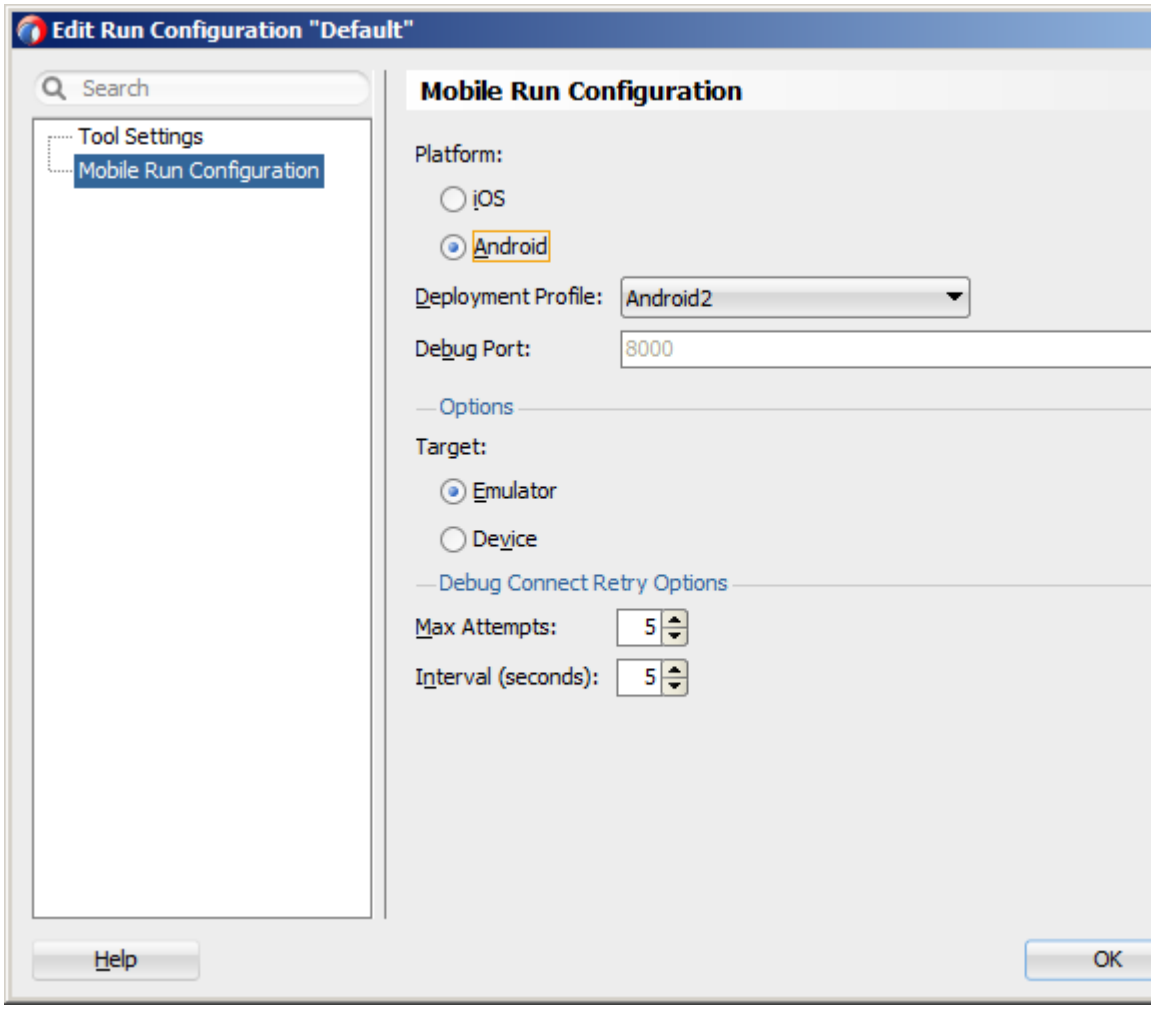
   If using Shared Settings, click **Edit Shared Settings** to open a dialog that allows you to create or edit new MAF run configurations. If using **Project Settings**, click **New** or **Edit**. With shared settings, the run configurations are shared across all projects. **Use Shared Settings** is the default option so you can use the MAF run configurations that exist at the time the project was added.
4. Complete the **Edit Run Configuration** dialog as follows:

- Select **Mobile Run Configuration** from the tree on the left.
- Select your target platform.
- Select a deployment profile.
- Enter the port number, up to five digits long. This number is initially seeded with the value of the `java.debug.port property` contained in the `maf.properties` file and appears as hint text. If a value is not specified for the port, the seeded value is used.
- For iOS, set the following options:
  - **Application Arguments**—Enter arguments that can be passed to the MAF application upon startup for customizing the runtime behavior of the application. For example, `-consoleRedirect=/<path>/<to>/log.txt` directs the log output to the file specified. The path must be an absolute path to receive the log file. The location must be writable for the current user.
  - **Simulator**—Choose which simulator you are deploying the application to. The options depend on which versions of the iOS SDK have been installed.
  - **iOS version**—Choose which version of iOS the simulator should use. The the drop-down menu displays the iOS versions that the selected device supports.
- For Android, set the following options:
  - **Target**—Select the deployment target (Emulator or Device).
  - **Max Attempts**—Choose the maximum number of connection attempts to allow.
  - **Interval (seconds)**— Choose the length of time in seconds between connection attempts.

  Tip:

  If the Android device or emulator is slow or times out, try increasing the Max Attempts or the Interval to allow sufficient time for Java to initialize and to force the Android starter to wait longer or try more attempts before quitting.

Figure 28-1 Mobile Run Configuration Dialog



28.3.2 How to Enable Debugging of Java Code and JavaScript

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 `Descriptors/META-INF` directory under the Application Resources (see Using and Configuring Logging), which corresponds to the `<application_name>/src/META-INF` location in your application file system.

When you execute a MAF run configuration, the following debugging properties are automatically set in the `maf.properties` file:

- `java.debug.enabled`: Set to `true` if doing a debug session; set to `false` if doing a run session.

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`: Set to the port number configured in the MAF run configuration being executed.
- `javascript.debug.enabled`: Set to `true` if doing a debug session; set to `false` if doing a run session. Applies to Android only.

Note:

The `javascript.debug.enabled` property is not required for enabling JavaScript debugging when the MAF application is running on an iOS-powered simulator or iOS-powered device.

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
```

For information on how to use JDeveloper to debug the Java code, see Debugging MAF Applications.

28.3.3 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 JDeveloper.

Note:

You can only debug your Java code and JavaScript (see How to Enable Debugging of Java Code and JavaScript). Debugging of EL expressions or other declarative elements is not supported.

28.4 Debugging MAF Applications Deployed on the Android Platform

Describes how to debug the Java and UI code of MAF applications that you deploy to Android.

To debug Java code, you configure the `maf.properties` file in your application and create a run configuration in JDeveloper. For more information about these tasks, see How to Enable Debugging of Java Code and JavaScript and Configuring JDeveloper 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 How to Debug Java Code on the Android Platform.

To debug UI code (JavaScript, HTML, and CSS), you configure the `maf.properties` file in your application (`javascript.debug.enabled=true`). Once you complete this task, you can debug your UI code, as described in How to Debug UI Code on the Android Platform.

28.4.1 How to Debug Java Code on the Android Platform

To debug a MAF application's Java code on the Android platform using JDeveloper, follow the debugging procedure described in Configuring JDeveloper and MAF Applications to Debug Code.

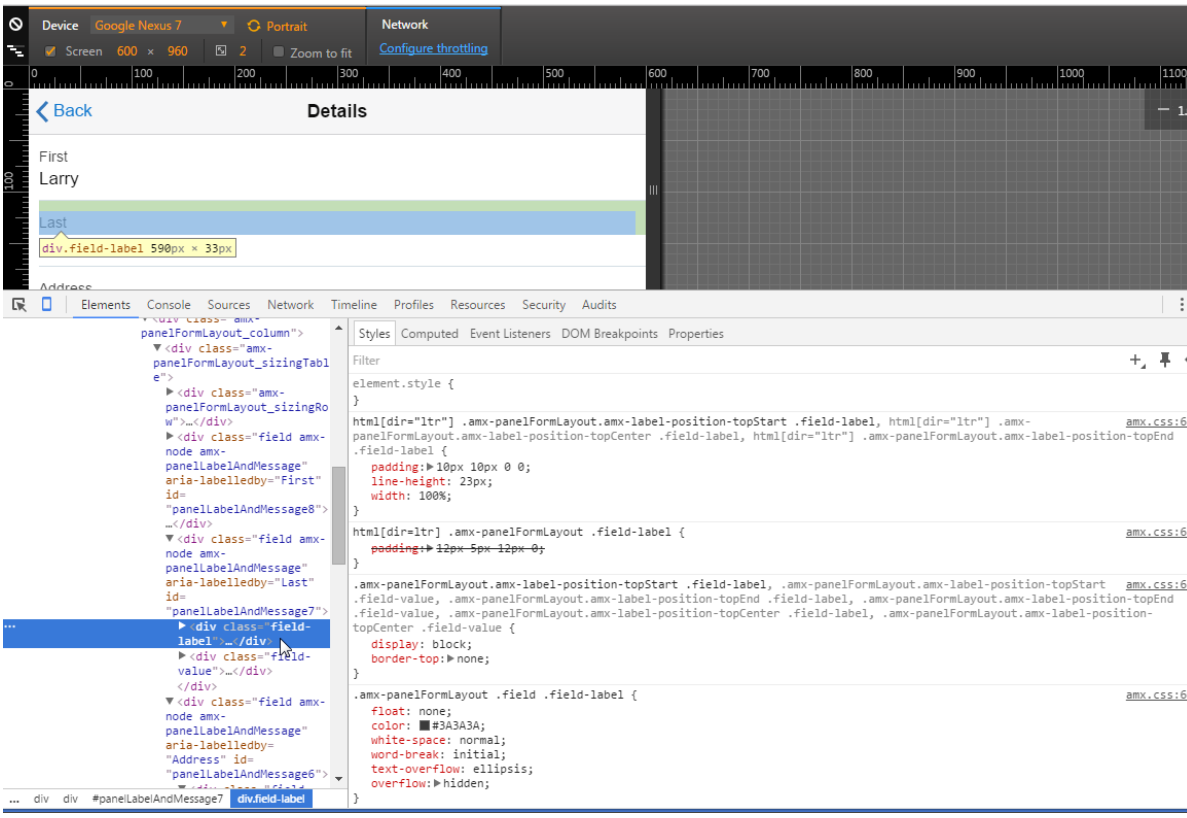
For information on how to configure an Android-powered device or emulator and how to deploy a MAF application for debugging, see How to Deploy an Android Application to an Android Emulator.

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 Using Plugins in MAF Applications.

28.4.2 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 28-2 shows the Chrome DevTools inspecting an AMX page from a MAF application.

Figure 28-2 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` in the current release.

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 How to Enable Debugging of Java Code and JavaScript.

- Deploy in debug mode. For more information, see Deploying MAF Applications.

28.5 Debugging MAF Applications Deployed on the iOS Platform

Describes how to debug the Java and UI code of MAF applications that you deploy to iOS.

Before you can debug Java code, you configure the `maf.properties` file in your application and create a run configuration in JDeveloper. For more information about these tasks, see How to Enable Debugging of Java Code and JavaScript and Configuring JDeveloper 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 How to Debug Java Code on the iOS Platform.

To debug UI code (JavaScript, HTML, and CSS), you configure the `maf.properties` file in your application (`javascript.debug.enabled=true`). Once you complete this task, you can debug your UI code, as described in How to Debug UI Code on the iOS Platform.

28.5.1 How to Debug Java Code on the iOS Platform

To debug a MAF application's Java code on the iOS platform using JDeveloper, follow the debugging procedure described in Configuring JDeveloper and MAF Applications to Debug Code.

For information on how to configure an iOS-powered device or simulator and how to deploy a MAF application for debugging, see the following:

- How to Deploy an iOS Application to an iOS Simulator
- How to Deploy an Application to an iOS-Powered Device
- Registering an Apple Device for Testing and Debugging

28.5.2 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 28-3 and Figure 28-4 show, and then select a **UIWebView** that you are planning to debug, as Figure 28-5 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`. For more information about setting additional build arguments, see Defining the iOS Build Options.

Figure 28-3 Using Develop Menu on Safari Browser for Debugging on iPhone Simulator

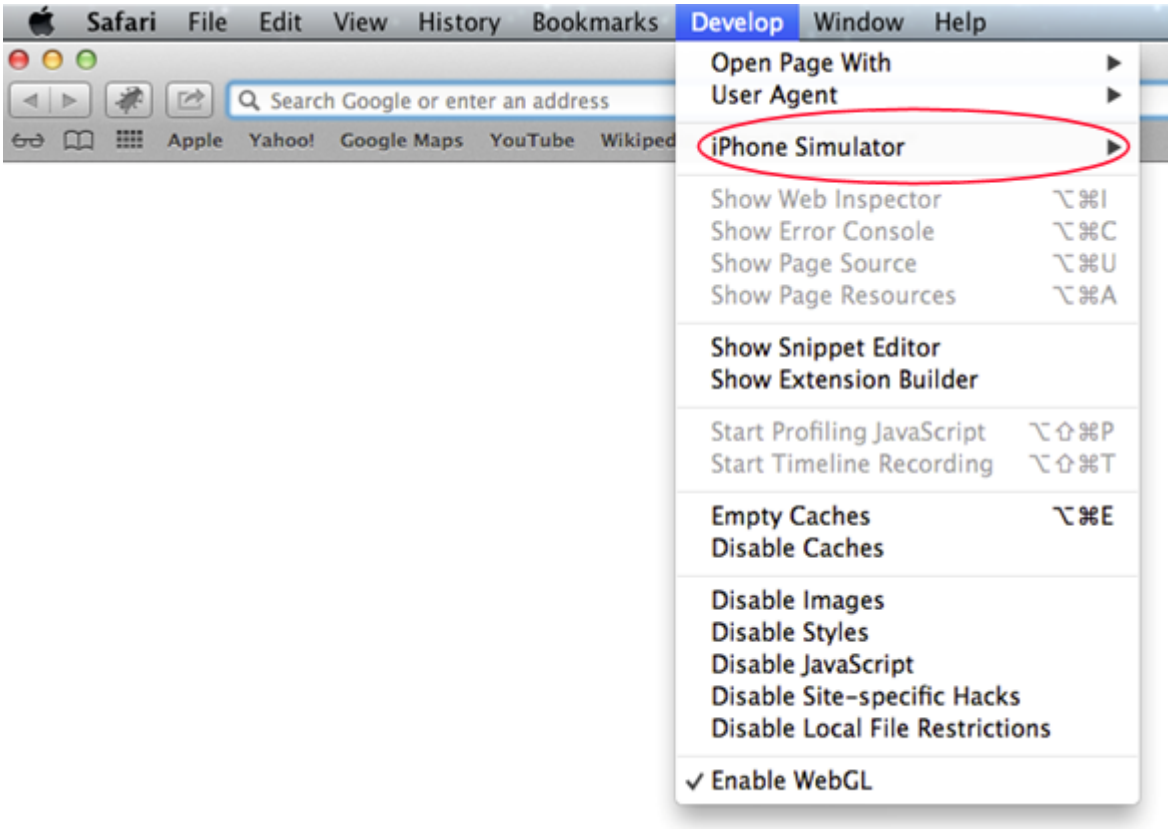


Figure 28-4 Using Develop Menu on Safari Browser for Debugging on iPad Simulator

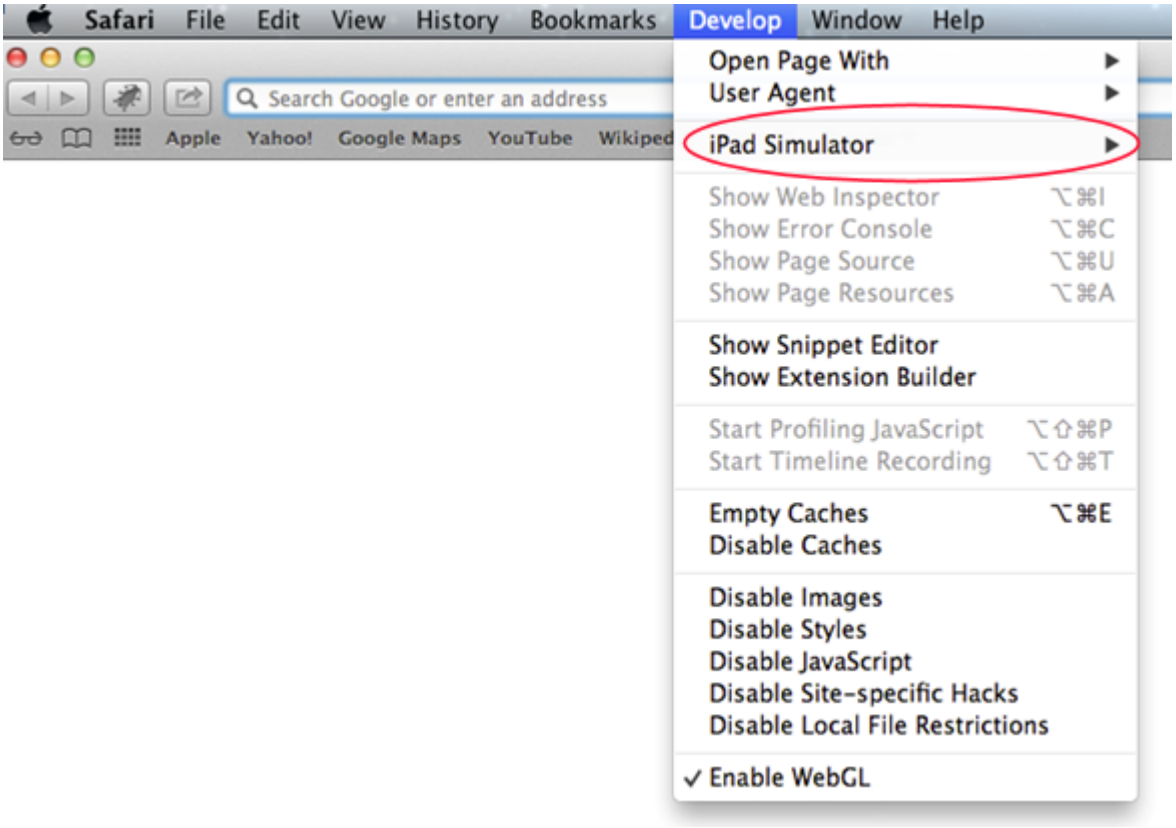


Figure 28-5 Using Develop Menu on Safari Browser to Select UIWebView

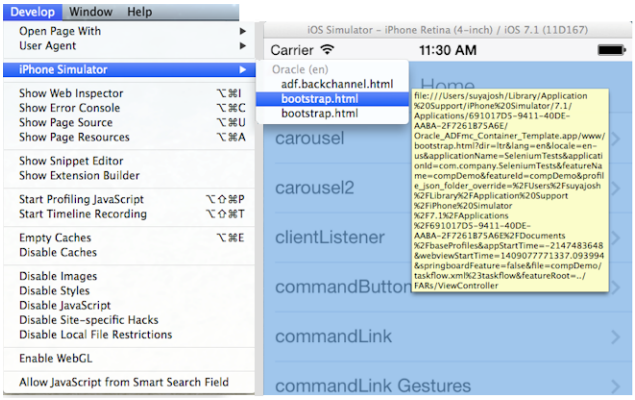


Figure 28-6 and Figure 28-7 show the CSS, DOM, and HTML Safari Remote Web Inspector in action.

Figure 28-6 Remote Web Inspector

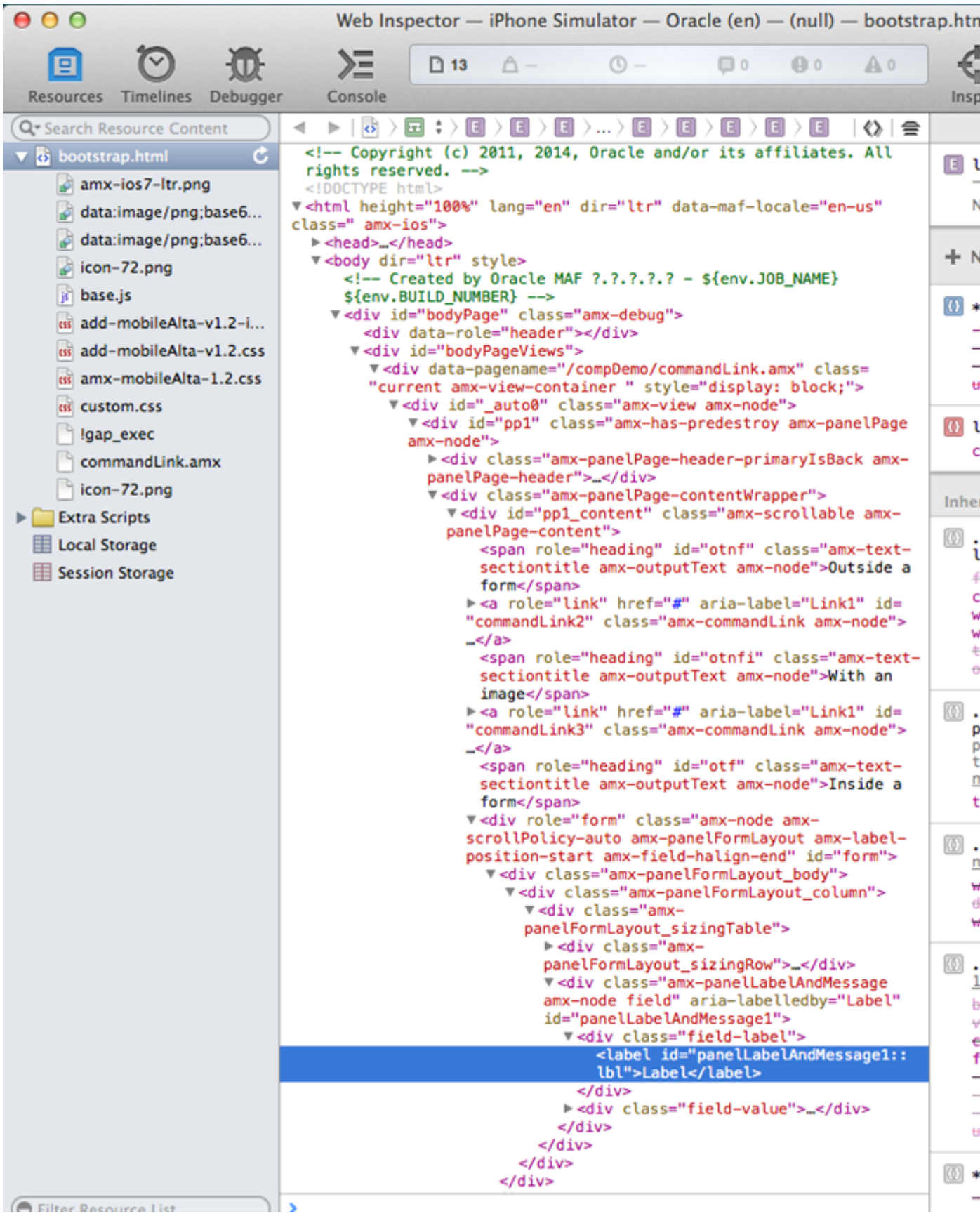


Figure 28-7 AMX Page Analyzed by Remote Web Inspector at Runtime

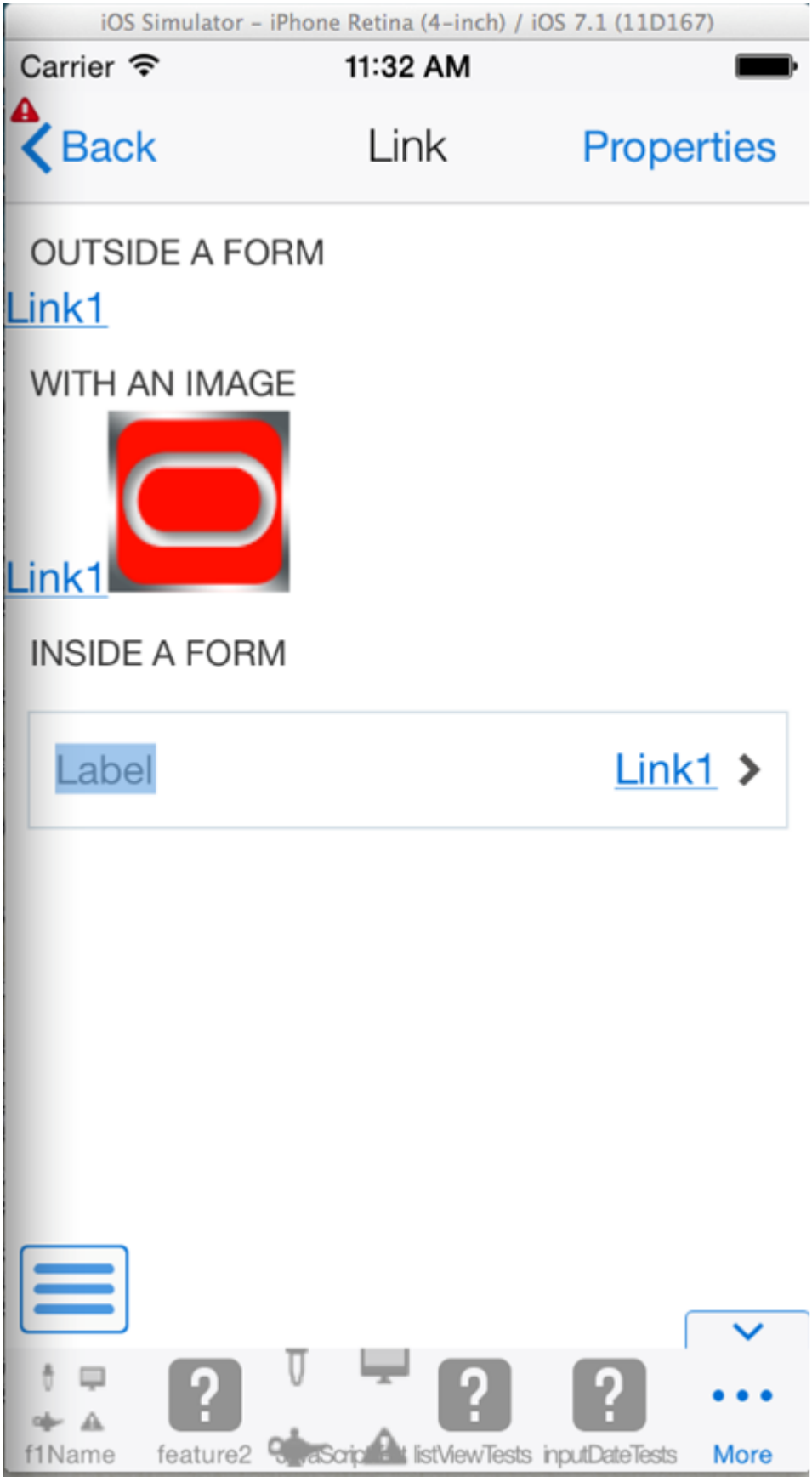


Figure 28-8 and Figure 28-9 show JavaScript debugging using breakpoints inside the Safari browser.

Figure 28-8 JavaScript Debugging in Safari Browser

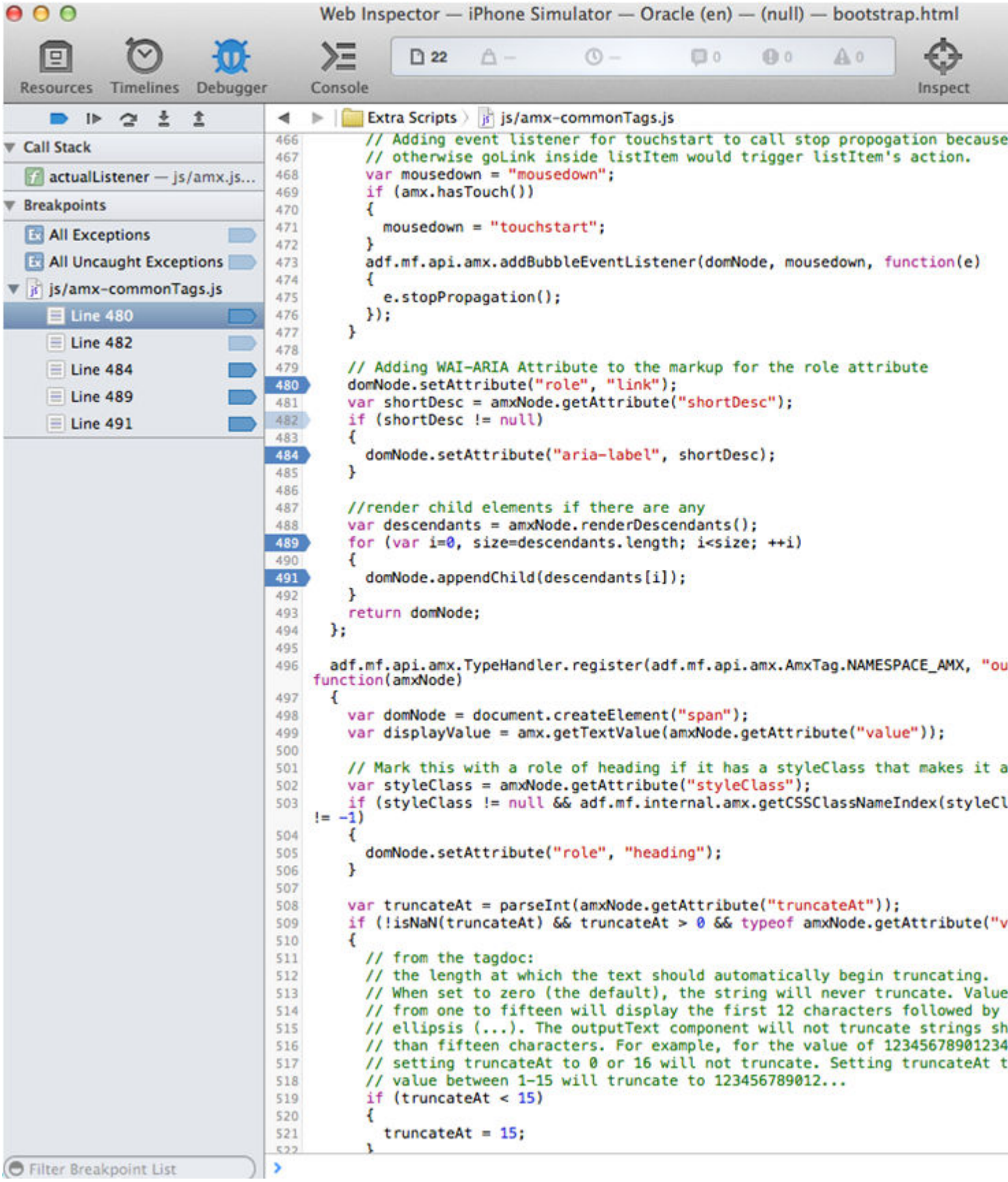
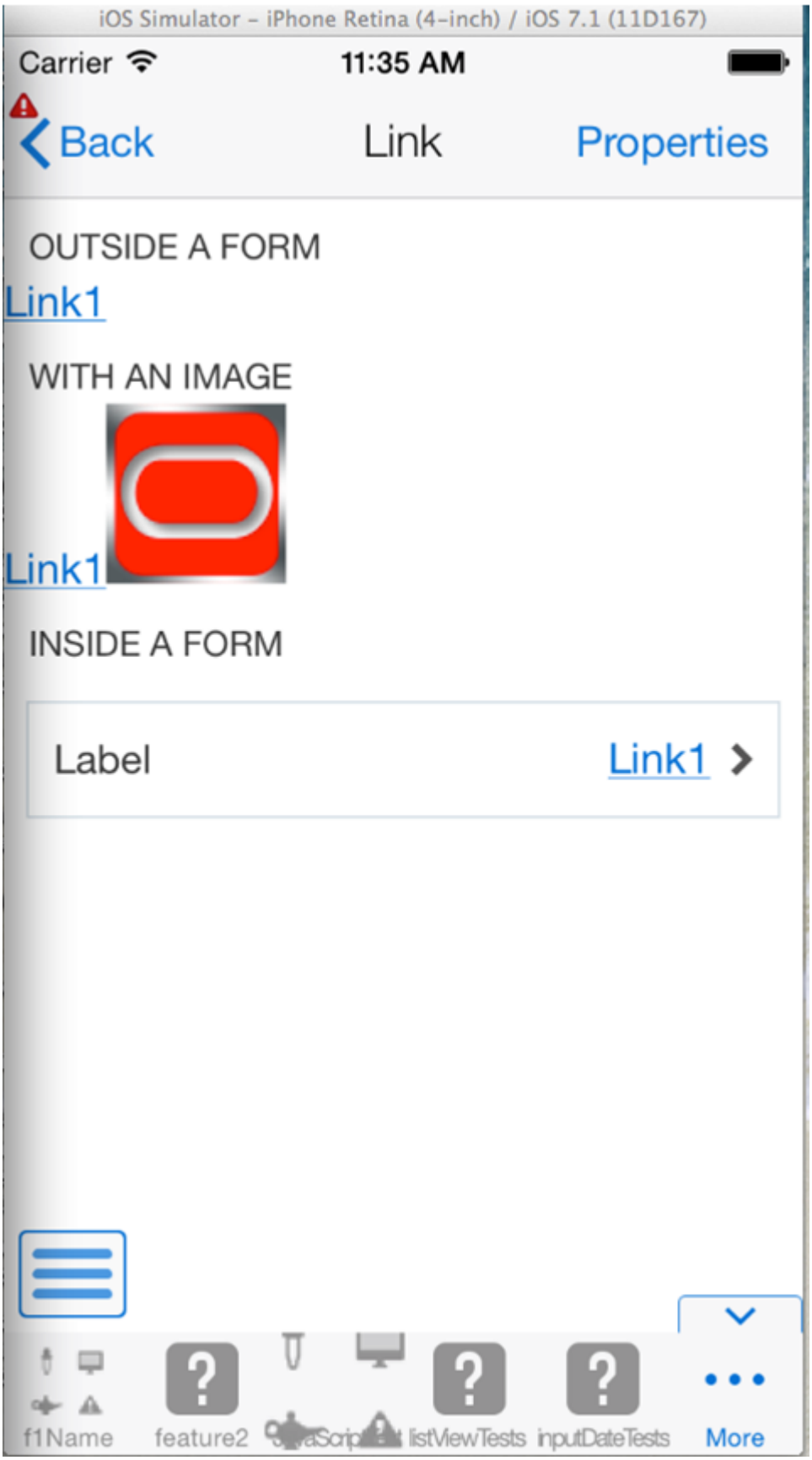


Figure 28-9 AMX Page Debugged at Runtime



28.6 Debugging MAF Applications Deployed on the Universal Windows Platform

Describes how to debug the Java and UI code of MAF applications that you deploy to the UWP.

You can debug the Java code in MAF applications that you deploy to the UWP using JDeveloper's debugging tools. For more information, see How to Debug Java Code on the Universal Windows Platform.

Use Visual Studio to debug the JavaScript, HTML, and CSS code in your MAF application, as described in How to Debug UI Code on the Universal Windows Platform.

28.6.1 How to Debug Java Code on the Universal Windows Platform

Describes how to debug Java code in a MAF application that you deploy to the Universal Windows Platform (UWP).

Perform the following steps so that you can debug Java code in a MAF application that you deploy to the UWP:

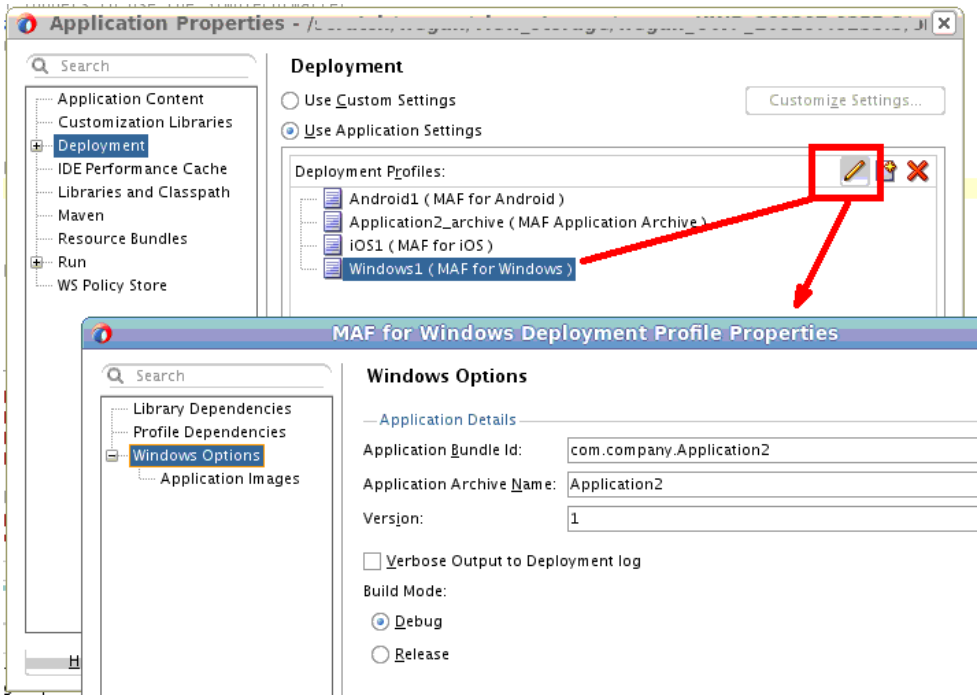
1. Configure the MAF application's `maf.properties` file entries to enable debugging. Ensure that the following values appear in the `maf.properties` file.

   ```
   java.debug.enabled=true
   java.debug.port=8000
   java.debug.mode=client
   java.debug.host=localhost
   ```

   For more information about the `maf.properties` file, see How to Enable Debugging of Java Code and JavaScript.

2. Configure the Windows deployment profile for the MAF application to use **Debug** mode, as shown in Figure 28-10. Click **Application > Application Properties > Deployment** to access this dialog.

Figure 28-10 Enabling Debug Mode for the MAF Application



3. Add a custom project to your MAF application so that it can access the standard JDeveloper application run/debug configuration dialogs. Once you add this custom project to your MAF application, enable the **Remote Debugging** option with a connection that listens for the JPDA protocol. Start a debug listener in the custom project. For more information about these tasks, see How to Enable Remote Debugging of a MAF Application on the Universal Windows Platform.

4. Deploy the MAF application using the local Windows machine option. For more information, see Deploying a MAF Application to the Universal Windows Platform.

Once you deploy the MAF application using the Windows Local Machine deployment option, the application starts and establishes a debug session with JDeveloper.

28.6.1.1 How to Enable Remote Debugging of a MAF Application on the Universal Windows Platform

Add a custom project to your MAF application to expose the Run/Debug configuration panels that the MAF application creation template does not display when you create a MAF application.

Using these configuration panels, you can enable remote debugging for a MAF application that you deploy to the UWP. After you add and configure the custom project, you start the debug listener from the custom project.

To add a custom project to enable remote debugging of a MAF application on the UWP:

1. From JDeveloper's main menu, click **File > New > Project**, select **Custom Project** in the **New Gallery** dialog and click **OK**.

2. Click **Finish** in the **Create Custom Project** to create a custom project named Project 1.

The new custom project (`Project 1`) appears in the Projects panel of your MAF application between the `ApplicationController` and `ViewController` projects.

3. Select the project, and from JDeveloper's main menu select **Application > Project Properties**.

4. In the **Run/Debug** page of the **Project Properties** dialog, select the Default run configuration and click the **Pencil** icon to display the **Launch Settings** dialog where you select **Remote Debugging** checkbox.

5. Navigate to **Tool Settings > Debugger > Remote** and configure the following properties, as illustrated in Figure 28-11:

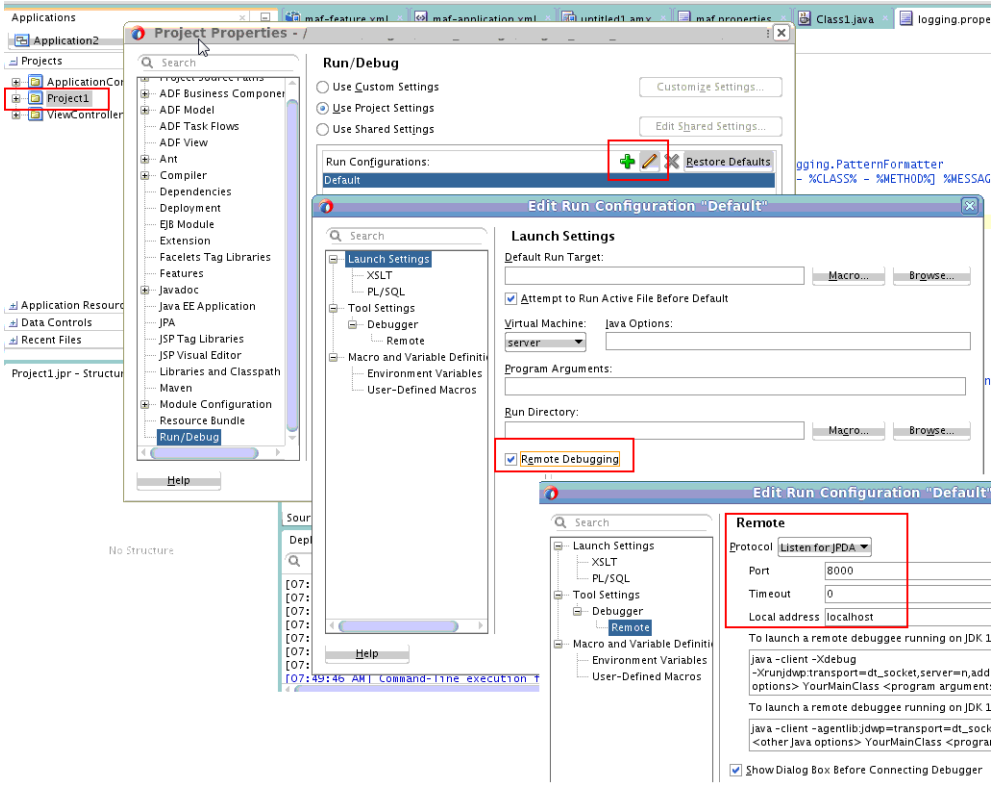
   a. **Protocol:**Select **Listen for JPDA** from the drop-down menu.

   b. **Port:** Enter the port number that you have configured for `java.debug.port` in the `maf.properties` file.

   c. **Timeout:** specify the number of seconds to time out. `0` means do not time out.

   d. **Local address**: `localhost`.

Figure 28-11 Enable Remote Debugging of a MAF Application Deployed to the UWP



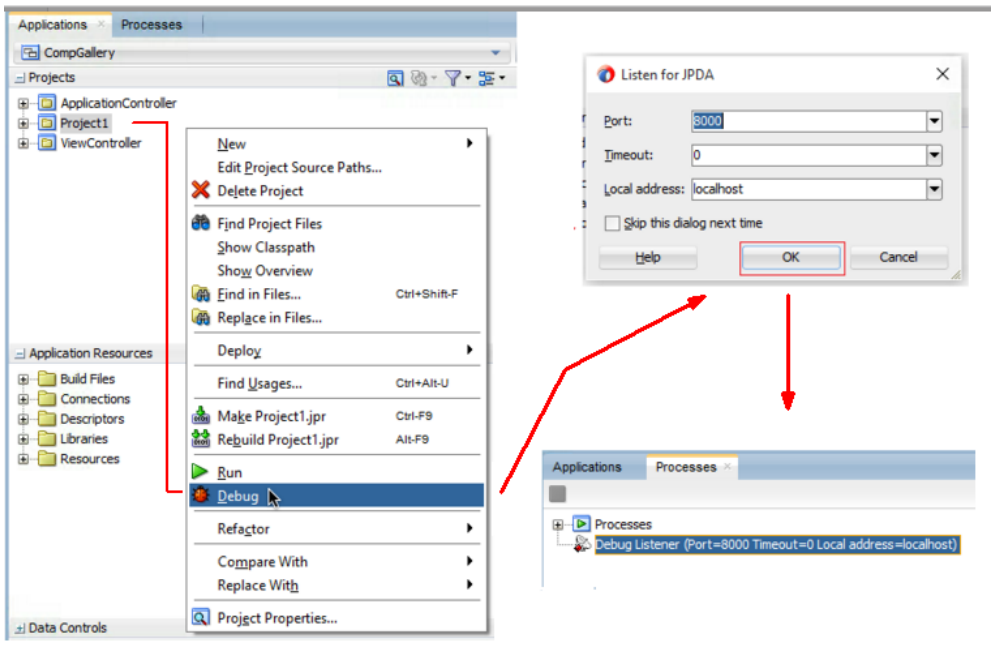
6. Click **OK** to close the open dialogs.

7. In the Applications window's Project panel, right-click the custom project (for example, `Project 1`) and choose **Debug** from the context menu that appears.

8. In the **Listen for JPDA** dialog, review the connection information to confirm it is correct and click **OK**.

9. From JDeveloper's main menu, click **Windows > Processes** to verify that the debug listener starts, as shown in Figure 28-12.

Figure 28-12 Debug Listener Started in MAF Application



Once you deploy the MAF application using the Windows Local Machine deployment option, the application starts and establishes a debug session with JDeveloper.

28.6.2 How to Debug UI Code on the Universal Windows Platform

When developing a MAF application on the Universal Windows Platform (UWP), you may need to debug code that renders the user interface (UI) of your application. The code that renders the UI can include JavaScript, HTML, and CSS. You debug this code using Visual Studio.

For information about installing Visual Studio for use in MAF application development, see the Setting Up Development Tools for the Universal Windows Platform section in *Installing Oracle Mobile Application Framework*.

Before debugging your MAF application, you need to develop and deploy the application. For information on developing and deploying application, see Getting Started with MAF Application Development and Deploying MAF Applications.

To debug UI code in MAF applications deployed on the UWP:

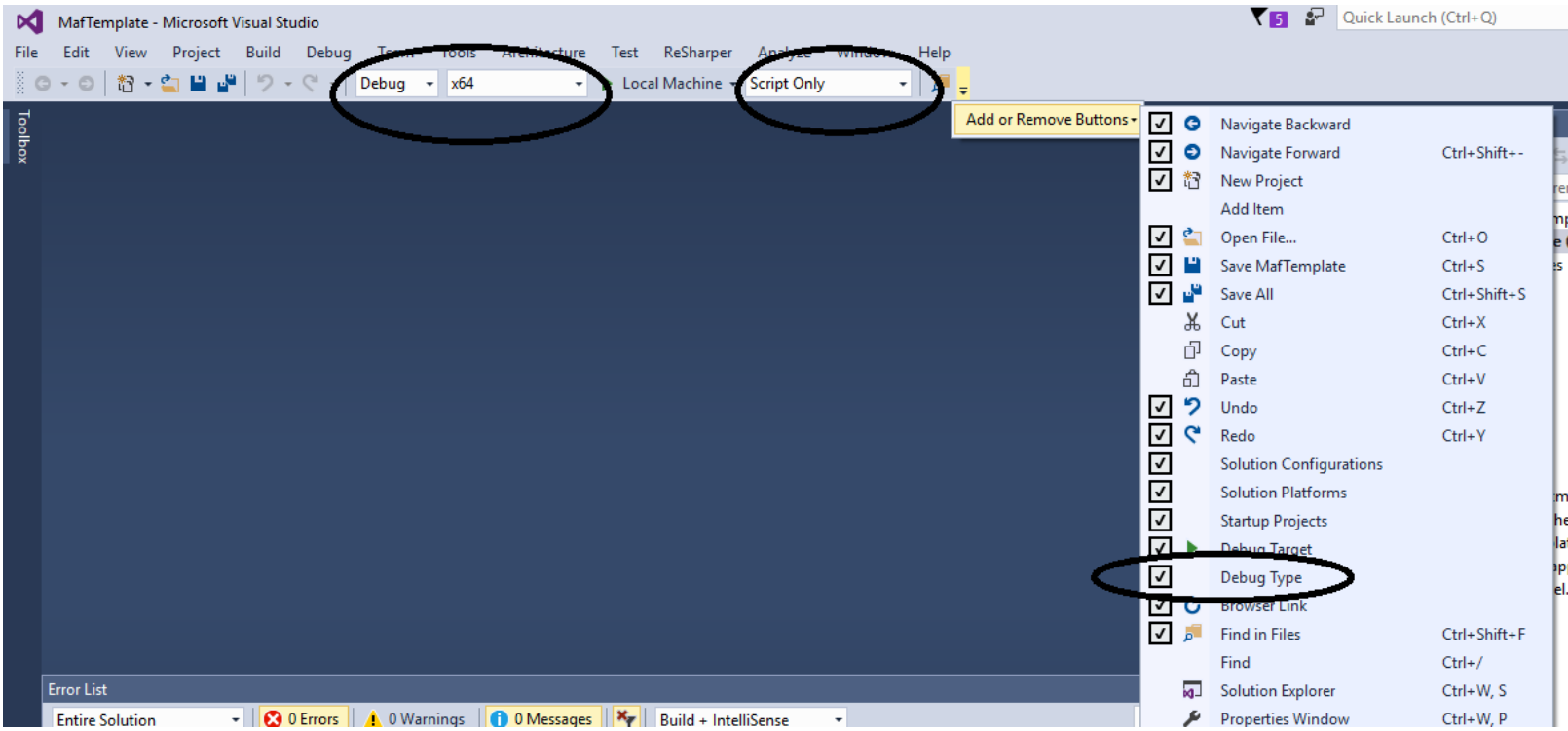
1. Deploy your MAF application in debug mode. For more information, see Deploying a MAF Application to the Universal Windows Platform.

2. Once you deploy your application to your local machine, navigate to the following directory: `C:\path\to\appRoot\deploy\[DeploymentProfileName]\debug\`.

3. Double-click the `MafTemplate.sln` template file that is configured with the application artifacts to open it in Visual Studio.

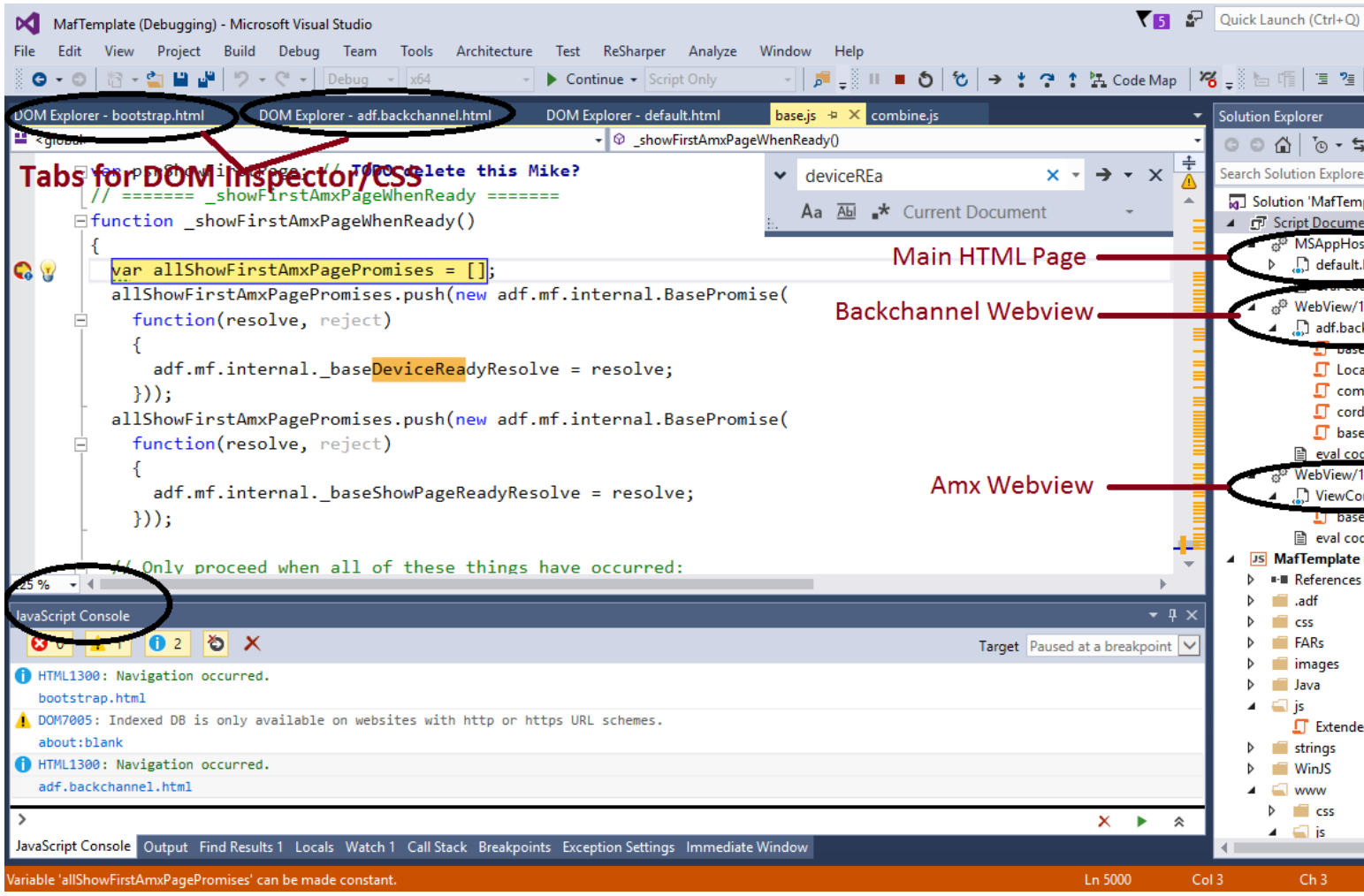
4. In Visual Studio as shown in Figure 28-13, set the value of CPU to x64.

Figure 28-13 Visual Studio IDE



5. If **Script Only** option is selected, you should be able to debug the UI code of AMX as shown in Figure 28-14.

Figure 28-14 Visual Studio Workspace



28.7 Using and Configuring Logging in MAF Applications

You can enable logging on all supported platforms through JavaScript.

For your MAF application, you can enable logging on all supported platforms through JavaScript (see How to Use JavaScript Logging) and embedded code (see 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, 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 typical locations for this file is `/Users/<userid>/Library/Developer/CoreSimulator/Devices/<device_id>/data/Containers/Data/Application/<container_id>/application.log`, where

<device_id> and *<container_id>* references represent long UUID strings created by iOS during the installation of the application. The values of these references cannot be predicted and when multiple simulators or applications are installed, it is difficult to determine which folder corresponds with the simulator used during deployment. When using JDeveloper for deployment, the `-consoleRedirect` option is automatically set to direct the log output to a known location. If you set it on a MAF run configuration for iOS using the **Edit Run Configuration > Mobile Run Configuration** dialog and that run configuration is used to deploy or run the MAF application on an iOS-powered device simulator, the log file is created at the configured location and its contents is displayed on a console in JDeveloper. If you do not set the value using the **Edit Run Configuration > Mobile Run Configuration** dialog or you choose to perform a regular deployment (for example, such that it does not use a MAF run configuration for iOS), the log file is created in the *<appRoot>/<deployRoot>/<iOSProfileName>/*`log`*<appName>*`.log` file and its contents is displayed on a console in JDeveloper. For more information, see Creating and Configuring a Run Configuration.

Note:

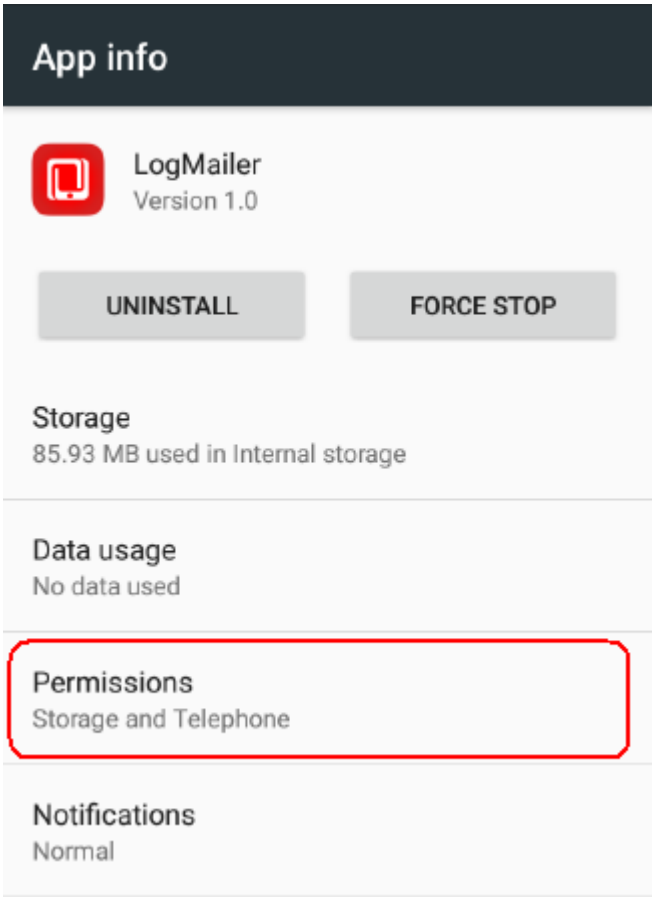
The path must be an absolute path to receive the log file and the location must be writable for the current user.

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. Double-click the `application.log` file.

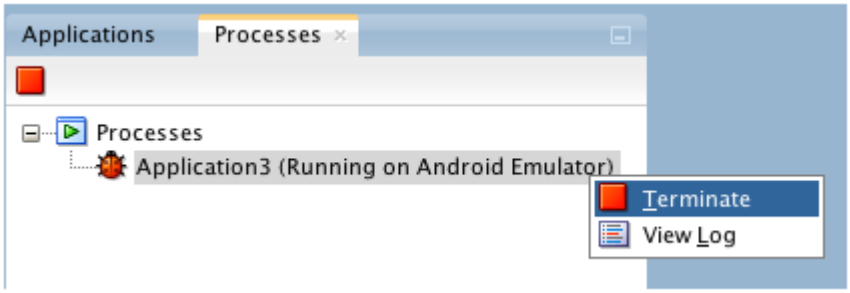
On Android, the output is forwarded to a text file with the same name as the application. The output file location is `/sdcard/Android/data/<app.package>/files/<app.name>.txt`. If this location is not present or is configured as read-only, the log output is rerouted to the application's writable data directory at `/data/data/<package>/files/<AppName>.txt`. On Android devices that use Android 6.0 +, MAF writes the log output to `/data/data/<package>/files/<AppName>.txt` unless the end user grants Storage permission to the MAF application, as shown in Figure 28-15. If the end user grants Storage permission, MAF writes the log output to `/sdcard/Android/data/<app.package>/files/<app.name>.txt`.

Figure 28-15 Granting Storage Permission to MAF Application on Android



The contents of the log file is replicated in the Android Logcat utility (see `http://developer.android.com/tools/debugging/debugging-log.html`). JDeveloper displays the logging output from the Logcat when you use JDeveloper to deploy your MAF application to an Android-powered device or emulator.

- For both iOS and Android, the logging output appears in JDeveloper's run or debug Log page immediately after the deployment. In case of a regular deployment, the deployment log is displayed in the **Deployment Log** page, whereas the application log is displayed in a separate Log page. If you use a Run/Debug run configuration to build, deploy, and launch the MAF application, the **Terminate** option in the **Processes** tab of the Applications window terminates the MAF application along with the process that performs the log redirection. The **View Log** option enables you to see the log page, as the following illustration shows.



If you use a deployment profile to build, deploy, and launch the MAF application, the **Terminate** option in the **Processes** tab terminates only the log redirection process, whereas the MAF application remains running.

- The `logging.properties` file is automatically created and placed in the `Descriptors/META-INF` directory under the Application Resources (see Using and Configuring Logging), which corresponds to the `<application_name>/src/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 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*.

28.7.1 How to Configure Logging Using the Properties File

The following example shows the `logging.properties` file that you use to configure logging.

```
# default - all loggers to use the ConsoleHandler
.handlers=java.util.logging.ConsoleHandler
# default - all loggers to use the SimpleFormatter
.formatter=java.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.

The logger defined in the `logging.properties` file matches the logger obtained from the `oracle.adfmf.util.Utility` class (see Using and Configuring Logging). The logging levels also match. If you decide to use the logging level that is more fine-grained than `INFO`, you must change the `ConsoleHandler`'s logging level to the same level, as the following example 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%
```

28.7.2 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 following example 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 *JSDoc Reference for Oracle Mobile Application Framework*.

28.7.3 How to Use Embedded Logging

Embedded logging uses the `java.util.logging.Logger`, as illustrated in the following example. The `EmbeddedClass` represents a Java class defined in the project.

```
import java.util.logging.Level;
import java.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
```

28.7.4 How to Use Xcode for Debugging and Logging on the 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 JDeveloper, you may choose to do so in exceptional circumstances.

Before you begin:

Deploy the application to the iOS simulator from JDeveloper.

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 JDeveloper console nor Xcode console). Instead, the output is redirected to a file (see Using and Configuring Logging). 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
```

28.7.5 How to Access the Application Log

You can retrieve the application log file for your MAF application from a user's device to analyze issues that occur with your MAF application. One way to accomplish this is to copy the application log file from its on-device location to a directory from where the application can then send the file from the device.

The following example demonstrates how you access the application log file location and copy it to a location from where it can be attached to an email message to send to a recipient who can analyse the content of the log file.

```
import java.nio.file.FileSystems;
import java.nio.file.Path;
import oracle.adfmf.util.Utility;

// Get path to application log file
private static Path getPathToLogFile() {
  String pathStr = Utility.getStorageLocations().getLogFileLocation();
  return FileSystems.getDefault().getPath(pathStr);
}


// Create an instance of device manager to access the device's email functionality
later
DeviceManager dm = DeviceManagerFactory.getDeviceManager();
```

```
//Construct path to application log file
String logFilePath = getPathToLogFile().toString();
...


//1. Determine device location to save a copy of the log file
String mailAccessiblePath =
Utility.ensureTrailingForwardSlash(AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaU
tilities.DeviceOnlyDirectory));
String targetFileNameAndPath = mailAccessiblePath + appName + ".log";

//2. Copy file to a location accessible from a mail client
try {
    Utility.copy(new File(logFilePath), new File(targetFileNameAndPath));
}
catch (IOException e) {
    // If something goes wrong, log the failure to copy.
    Utility.ApplicationLogger.logp(Level.SEVERE, this.getClass().getSimpleName(),
"sendLogAsMail", "Could not copy file " + logFilePath + " to " +
targetFileNameAndPath);
    Utility.ApplicationLogger.logp(Level.SEVERE, this.getClass().getSimpleName(),
"sendLogAsMail", e.getLocalizedMessage());
}

//3. Set the attachment property referenced by sendEmail(...)
this.setMailAttachment(targetFileNameAndPath);

//4. Send mail: open the mail client
dm.sendEmail(mailTo, mailCc, mailSubject, mailBody, mailBcc, mailAttachment,
mailMimeType);
```

Note: If you deploy your application from JDeveloper to a virtual device (iOS simulator or Android emulator), MAF writes the log output to JDeveloper's Log window instead of to the application log file.

Once you obtain the location of the application log file, choose a mechanism to send it from the user's device to the server side. Options to consider include uploading the file using a REST web service to a specific server-side location that is associated with your application. This option may offer a more consistent user experience to the alternative option of transmitting the log file as an email attachment. The latter option can exhibit different behaviors based on the platform device (iOS, Android, UWP) or how the email client is configured on the device.

The level of detail that the MAF application's log file captures depends on the configuration entries in the logging.properties file unless you change the logging level dynamically at runtime by, for example, using an API as demonstrated in the following example:

```
Logger l = Utility.ApplicationLogger;
// Select a new log level. Note that OFF disables logging. In our example, we select
ALL.
// Level newLevel = Level.<ALL | CONFIG | INFO | FINE | FINER | FINEST | OFF |
SEVERE | WARNING>
Level newLevel = Level.ALL;
l.setLevel(newLevel);
```

28.7.6 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 Introduction to MAF Application and Project Files.

28.8 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 28-1 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.ConsoleHan
dler
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.loggi
ng.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 28-1 describes the available performance monitor levels that you can set in the `logging.properties` file.

Table 28-1 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 28-1 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)
```

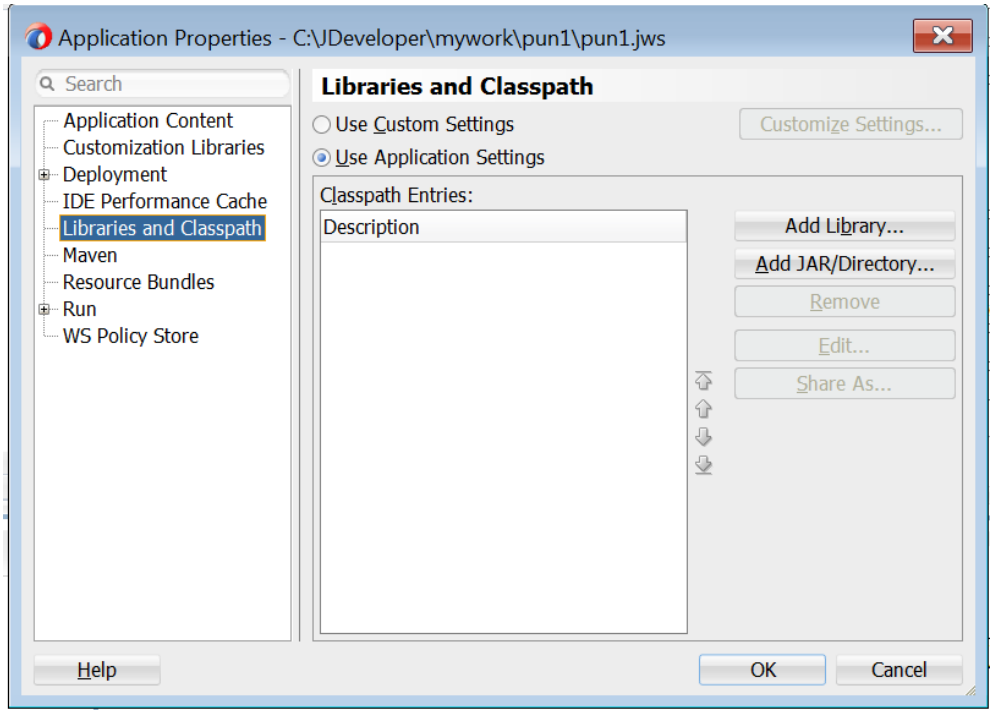
28.9 Inspecting Web Service Calls in a MAF Application

If your MAF application accesses REST services that were configured using the design-time support for the creation of the client data model, you can add a reusable application feature to view the request and response details of every REST call made by the application.

For more information about the client data model, see Creating the Client Data Model in a MAF Application. To add this feature to your application, perform the following steps:

1. In JDeveloper, go to the **Application** menu, and select the **Application Properties...** option. The **Application Properties** dialog is displayed.
2. In the **Application Properties** dialog, click the **Libraries and Classpath** option, shown in Figure 28-16.

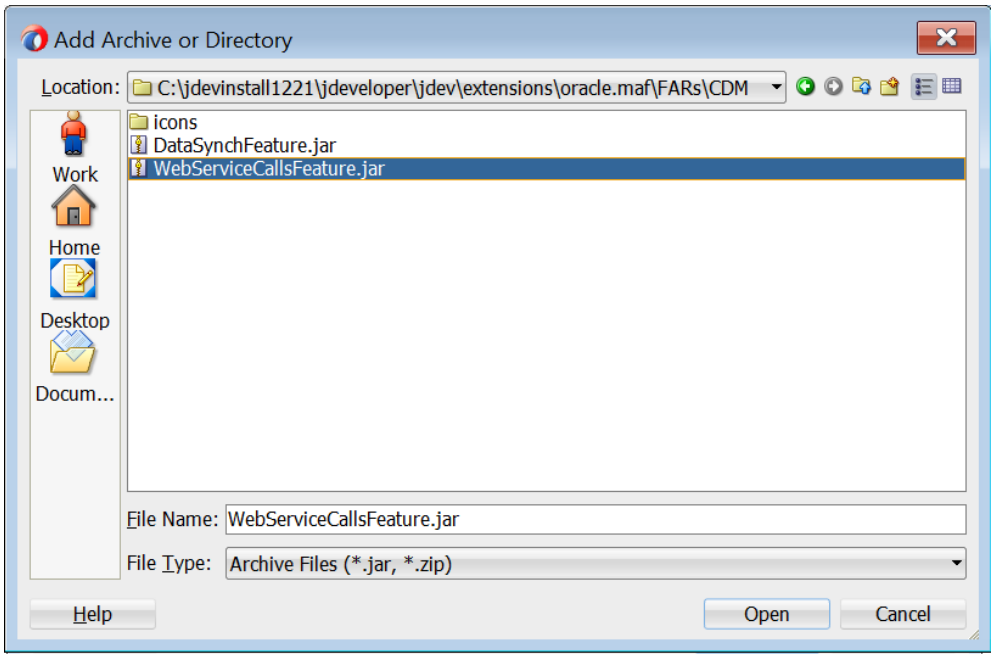
Figure 28-16 Application Properties dialog



3. In the **Application Properties** dialog, click the **Add JAR/Directory** button. The **Add Archive or Directory** dialog is displayed.

4. In the **Add Archive or Directory** dialog, navigate to the directory `./jdevinstall/jdeveloper/jdev/extensions/oracle.maf/FARs/CDM` and select the `WebServiceCallsFeature.jar` file and click the **Open** button, shown in Figure 28-17.

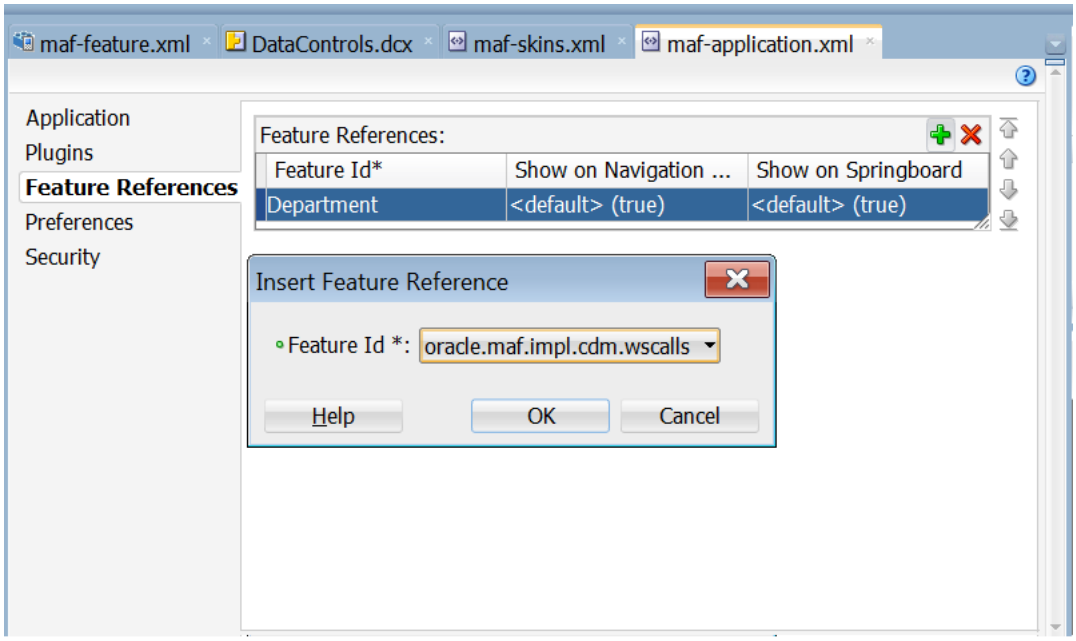
Figure 28-17 Add Archive or Directory dialog



5. The `WebServiceCallsFeature.jar` file appears under **Library and Classpath**. Click **OK** to close the **Application Properties** dialog.

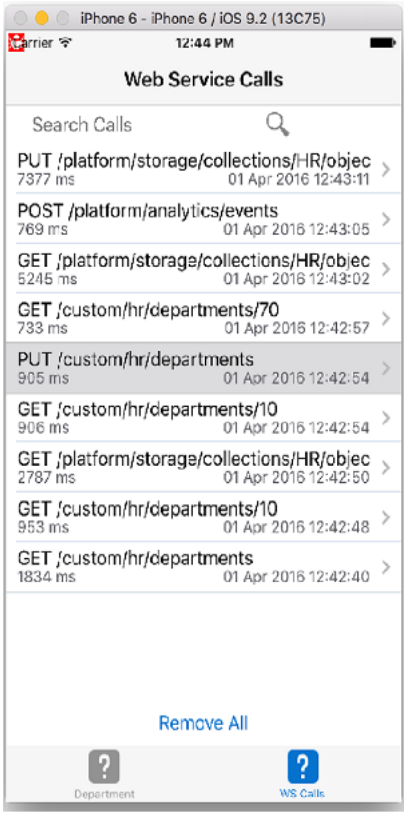
6. Open the `maf-application.xml` file in the editor and select the **Feature References** option. Click the green plus icon on the right and select `oracle.maf.impl.cdm.wscalls` from the **Feature Id** drop-down list, shown in Figure 28-18.

Figure 28-18 maf-application.xml file



7. Deploy your MAF application to view the application feature that allows you to inspect web service calls, shown in Figure 28-19.

Figure 28-19 View Rest Calls



8. You can click on one of the REST call in the list to view all request and response details.

29

Reusing MAF Application Content with a Feature Archive File

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:

- Introduction to Feature Archive Files
- Using FAR Content in a MAF Application
- What Happens When You Add a FAR as a Library
- What Happens When You Add a FAR as a View Controller Project
- What You May Need to Know About Enabling the Reuse of Feature Archive Resources

29.1 Introduction to Feature Archive Files

A Feature Archive file is a JAR file that is packaged with application features for reuse by other MAF applications. A Feature Archive file contains components of application features, and it can be added as an application library or as a view controller project.

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 either add a FAR as an application library or as a view controller project. You cannot customize the contents of a FAR 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, the task flow of a FAR cannot be the target of a task flow call activity. Adding a FAR as a view controller project, however, enables you to customize its artifacts, as described in Customizing MAF Application Artifacts with MDS .

29.2 Using FAR Content in a MAF Application

An application feature is made available to a MAF application by adding it to the class path of the consuming application.

Note:

You can only add the FAR to the application controller project; you cannot add a FAR to the view controller project.

If the FAR defines a connection in the `connections.xml` file that is also defined in the consuming application's `connections.xml`, no connections will be added to the consuming application and the FAR will not be added to the application.

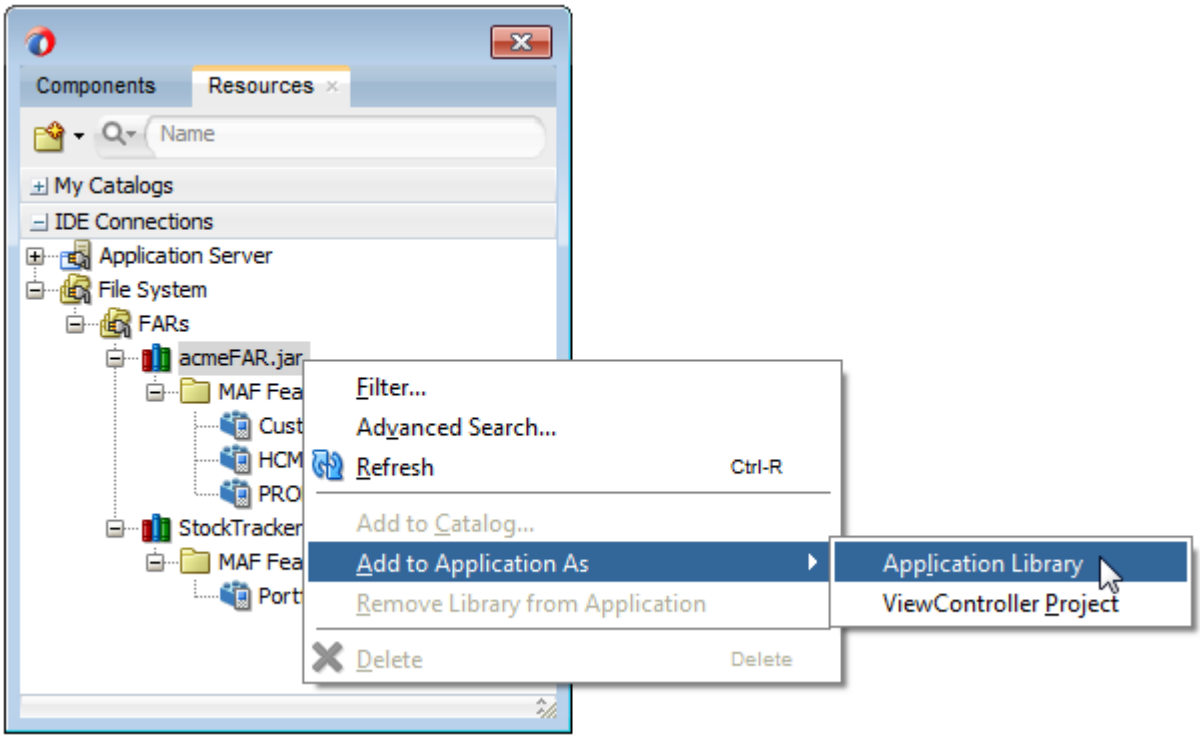
Before you begin:

Deploy the application feature as a Feature Archive file, as described in How to Deploy the Feature Archive Deployment Profile.

How to add application feature content to a MAF application as a library:

1. Open the Resources window, and click **New**, then **IDE Connections**, and then **File System**.
2. Complete the File Systems Connection dialog to create a file connection to the directory that contains the Feature Archive JAR file. For more information, refer to the Oracle JDeveloper Online help.
3. Right-click the Feature Archive file (which is noted as a JAR file) in the Resources window.
4. Click **Add to Application As**, and then **Library** to add the classpath of the consuming application, as shown in Figure 29-1.

Figure 29-1 Adding a FAR to a MAF Application as a Library



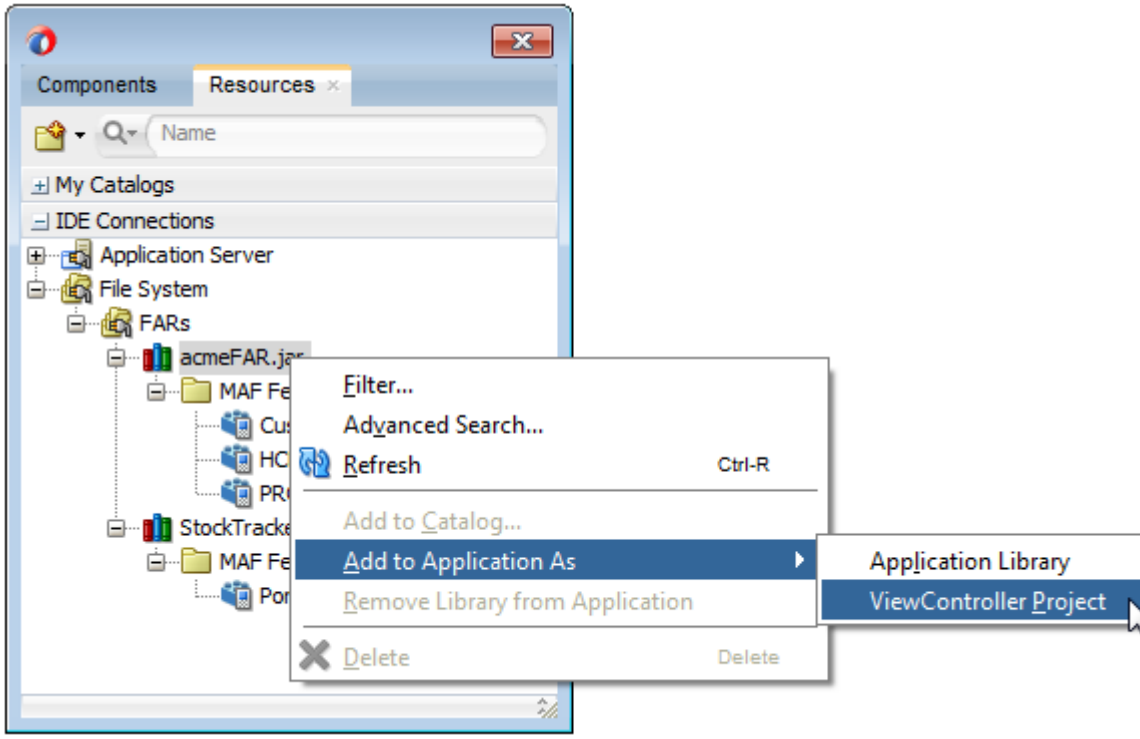
Tip:

Click **Remove Library from Application** to remove the feature archive JAR from the classpath of the consuming application.

How to add a FAR as a view controller project:

1. Open the Resources window and click **New**, then **IDE Connections**, and then **File System**.
2. Complete the File Systems Connection dialog to create a file connection to the directory that contains the Feature Archive JAR file. For more information, refer to the Oracle JDeveloper Online help.
3. Right-click the Feature Archive file (which is noted as a JAR file) in the Resources window.
4. Click **Add to Application As**, and then **ViewController Project** as shown in Figure 29-2.

Figure 29-2 Adding a FAR to a MAF Application as a View Controller Project



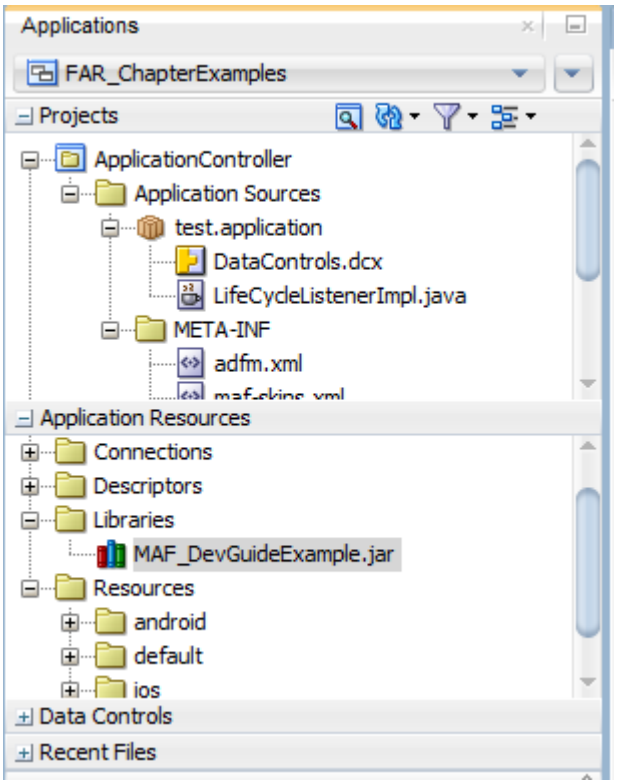
29.3 What Happens When You Add a FAR as a Library

A FAR file can either be added manually to the classpath of the application, or features may be added to Resources to make them available to applications. Use the procedure to add or remove an application feature from the Resources window.

After you add a FAR as a library (or manually to the classpath of the application):

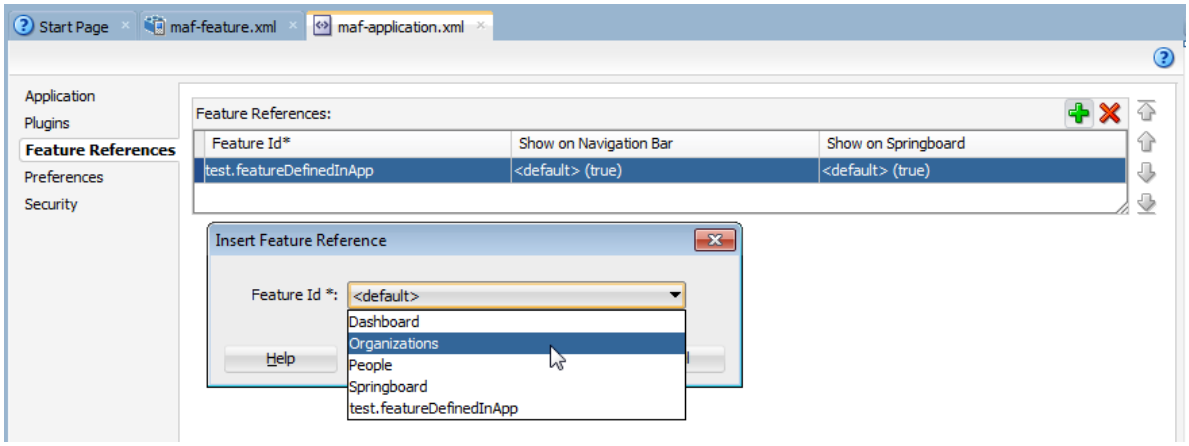
- The contents of the FAR display in the Application Resources under the **Libraries** node, as shown in Figure 29-3.

Figure 29-3 The FAR JAR File in the Application Resources of the Consuming 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 29-4 where the drop-down list IDs of the available application features in the JAR in addition to the one that has already been defined in the application.

Figure 29-4 Referencing the Application Features Defined in Various maf-feature.xml Files



Tip:

Manually adding the Feature Archive JAR to the application classpath also results in the application features displaying in the Insert Feature Reference dialog.

Alternatively, you can add or remove an application feature from the Resources window as follows:

1. Expand the feature archive JAR in the Resources window.
2. From the MAF Features folder, right-click an application feature.
3. Choose **Add Feature Reference to maf-application.xml**, as shown in Figure 29-5, or **Remove Feature Reference from maf-application.xml**, shown in Figure 29-6. Figure 29-5 illustrates adding an application feature called People from `MAF_DevGuideExample.jar`.

Figure 29-5 Adding a Feature Reference

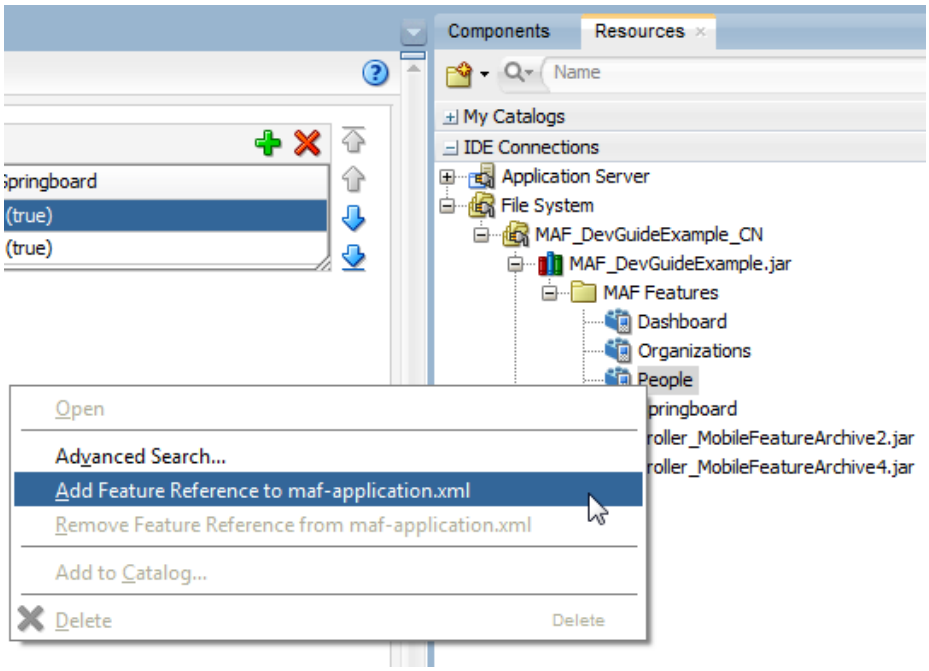
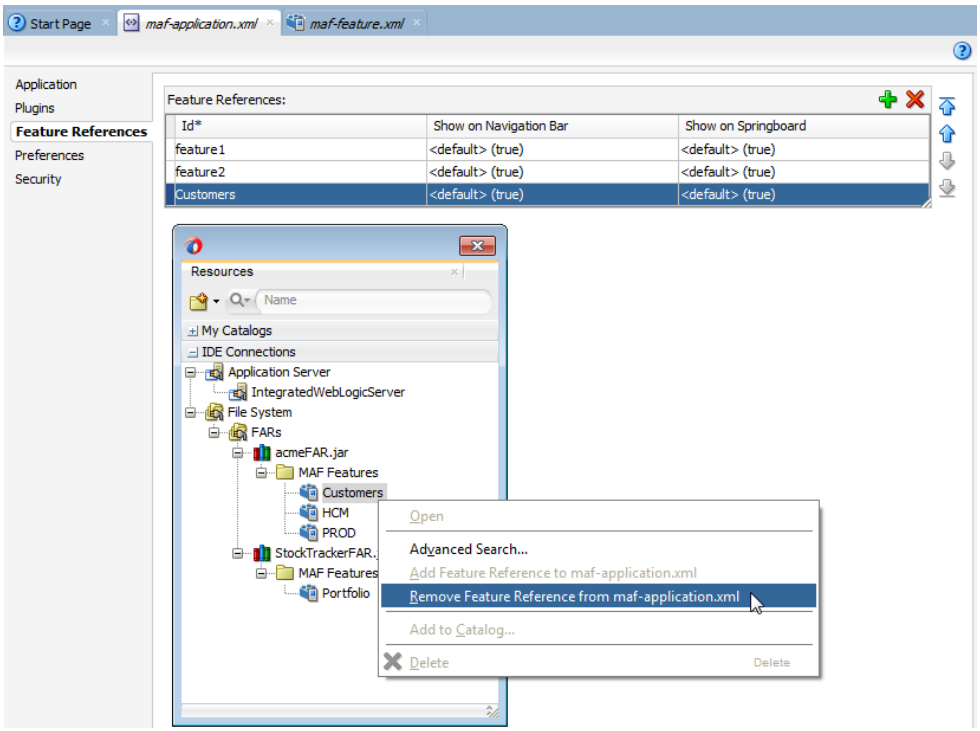


Figure 29-6 illustrates removing an application feature reference from the `maf-application.xml` file.

Figure 29-6 Removing a Feature Reference



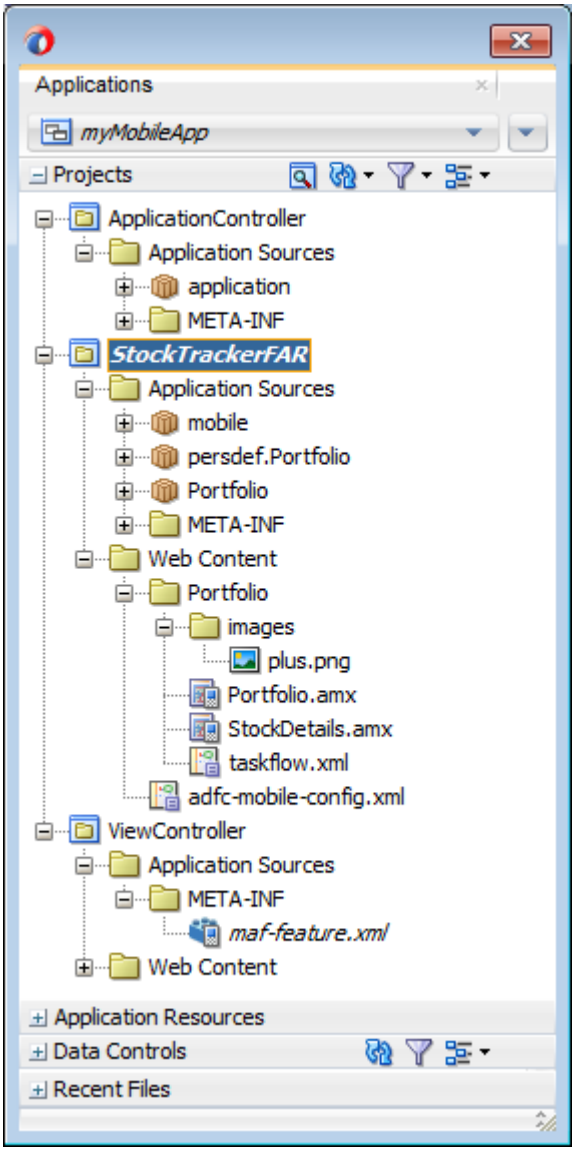
29.4 What Happens When You Add a FAR as a View Controller Project

A FAR, when added as a view controller project, generates a view controller project, causes the creation of a `connections.xml` file, and includes `.class` and JAR files in the FAR available to the view controller project.

When you add a FAR as a view controller project:

- MAF generates a view controller project that bears the same name as the imported FAR. Figure 29-7 illustrates how MAF creates a view controller project (a `.jpr` file) for an imported FAR file called *StockTracker* (which is illustrated as *`StockTrackerFAR.jar`* in Figure 29-2). This view controller project contains the default structure and metadata files of a MAF view controller project, as described in About the View Controller Project Resources. In particular, the FAR view controller project includes the `maf-feature.xml` file. If the MAF application contains other view controller projects, you must ensure that none of these projects include application features with the same ID. See also What You May Need to Know About Enabling the Reuse of Feature Archive Resources.

Figure 29-7 The Imported FAR as a View Controller Project within a MAF Application



- As with a FAR imported as a library, the information in the `connections.xml` file located in the Feature Archive JAR is merged into the `connections.xml` file of the consuming application. MAF will create a `connections.xml` file if one does not already exist in the target application.
- MAF makes any `.class` and JAR files included in the FAR available as a library to the view controller project by copying them into its `lib` directory (such as `C:\jdeveloper\mywork\`*`application`*`\`*`FAR view controller project`*`\lib`). MAF compiles these files into a file called `classesFromFar.jar`.
- Unlike a FAR imported as a library, you can customize the files of a view controller project.

Note:

Because the original resource bundles included in FAR might not be usable in the generated view controller project, you must create new resources bundles within the project as described in Enabling Customizations in Resource Bundles.

- Like a FAR imported as a library, every application feature declared in the `maf-feature.xml` file of the FAR becomes available to the consuming application.

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

Resources of a FAR can be used by an application only if some conditions, such as a unique package hierarchy for the project, and unique package naming system for the feature reference IDs, are satisfied.

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 that 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. Instead of accepting the default names for these package directories, create a unique package hierarchy for the project. Use a similar package naming system for the feature reference IDs too.

30

Reusing MAF Application Content with a MAF Shared Library

This chapter describes how to package content, such as task flows, AMX page fragments, and CSS files, into a MAF shared library (shared library) and distribute it to other MAF applications for reuse.

This chapter includes the following sections:

- Introduction to Reusing MAF Application Content with a MAF Shared Library
- Creating a MAF Application for Shared Library
- Adding Artifacts to a MAF Application for Shared Library
- Deploying a MAF Application for Shared Library
- Consuming a Shared Library in a MAF Application
- Deploying a MAF Application with Duplicate Shared Library Archive File Names
- Using Cordova Plugins in Shared Libraries

30.1 Introduction to Reusing MAF Application Content with a MAF Shared Library

A shared library lets you share content for reuse with multiple MAF applications.

You can, for example, put a task flow in a shared library from where it can be consumed by multiple MAF applications. Other types of content that can be shared include AMX page fragments, images, CSS and JavaScript files plus HTML pages like the custom launch screen for iOS and the custom login page. A shared library differs from a Feature Archive (FAR) in that the FAR shares application features while you use a shared library to share individual artifacts, such as task flows.

If you want to develop a shared library that other MAF application developers can consume in their applications, use the following workflow:

1. Create an application using the MAF Application for Shared Library application template. This template provides the required files and directory structure for a shared library and a deployment profile to deploy it. See Creating a MAF Application for Shared Library.
2. Add the artifacts, such as task flows, which you want to distribute in the shared library. See Adding Artifacts to a MAF Application for Shared Library.
3. Test your shared library by deploying it and verifying that a MAF application can reuse the artifacts that it contains. See Deploying a MAF Application for Shared Library and Consuming a Shared Library in a MAF Application.

4. Distribute the ZIP file that contains the completed shared library.

If you want to consume a shared library created by another MAF application developer, create a file system connection in JDeveloper's Resources window to the directory that contains the extract from the shared library ZIP file that the other MAF application developer distributed. You can then add the shared library to your application. See Consuming a Shared Library in a MAF Application.

30.2 Creating a MAF Application for Shared Library

Add the artifacts that you want to share with other MAF applications to the MAF application for shared library.

Creating a MAF application for shared library is the first step to sharing artifacts with other MAF applications. This creates two projects (`ApplicationControllerLibrary` and `ViewControllerLibrary`) and a `library-content.xml` file where you describe the content of the shared library. Once you create the application for a shared library and the associated projects, perform the following tasks:

- Review the values for the MAF Library Root Directory properties in the `ApplicationControllerLibrary` and `ViewControllerLibrary` projects to ensure that they are unique, as described in How to Set the Shared Library Root Directory.
- Add the artifacts you want to share, as described in Adding Artifacts to a MAF Application for Shared Library.
- Describe the shared library for consumers, as described in How to Describe the Shared Library Content to Consumers.

30.2.1 How to Create a MAF Application for Shared Library

Create a MAF application for shared library using the application wizard that you invoke from JDeveloper's New Gallery.

To create a shared library:

1. In the main menu, choose **File** and then **Application** > **New**.
2. In the New Gallery, in the Items list, double-click **Mobile Application Framework Application for Shared Library**.
3. In the Mobile Application Framework Application for Shared Library wizard, enter application details like the name, directory, and application package prefix. For help with the remaining pages of the wizard, click **Help**.

   The Project 1 Name page of the wizard shows the list of features needed to complete the `ApplicationControllerLibrary` project. The Project 2 Name page of the wizard shows the list of features needed to complete the `ViewControllerLibrary` project.

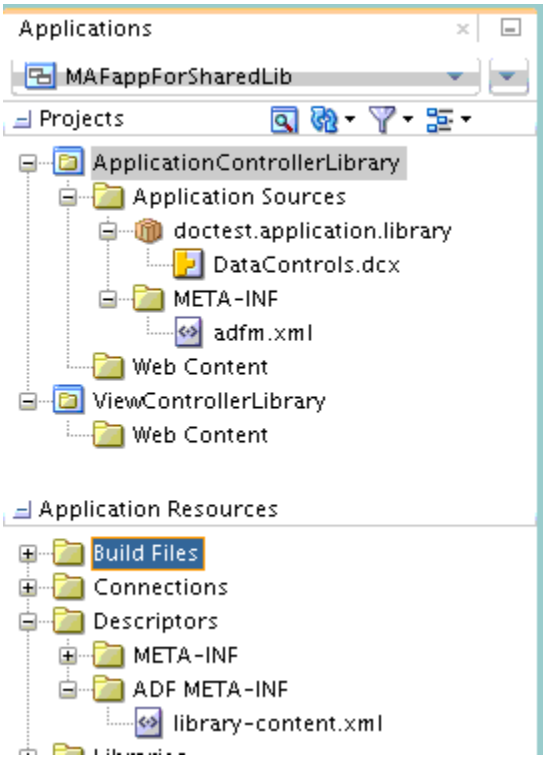
   Ensure that the value you specify for default package in the Configure Java settings page for each project is unique. Doing so ensures a unique value for the library root directory property and avoids deployment issues for shared library consumers, such as those described in Deploying a MAF Application with Duplicate Shared Library Archive File Names.
4. Click **Finish**.

30.2.2 What Happens When You Create a MAF Application for Shared Library

JDeveloper creates a new application with two projects and a variety of other files that these projects require.

Figure 30-1 shows the newly-created MAF application for shared library.

Figure 30-1 Newly-created MAF Application for Shared Library



Example 30-1 shows the content of a newly-created MAF application for a shared library. It contains two projects (`ApplicationControllerLibrary` and `ViewControllerLibrary`) where you add the artifacts that you want to distribute for use with other MAF applications. It also contains a `library-content.xml` file, which is the file that describes the shared library.

Example 30-1 Content of a Newly-Created MAF Application for Shared Library

```
MAFappForSharedLib
+---.adf
|   \---META-INF
|           library-content.xml
+---.data
|   ...
+---ApplicationControllerLibrary
|   |   ApplicationControllerLibrary.jpr
|   |
|   +---adfmsrc
|   |   +---application
|   |   |   \---library
|   |   |           DataControls.dcx
|   |   |
|   |   \---META-INF
|   |           adfm.xml
|   |
```

```
|   \---public_html
|       \---application
|           \---library
|               \---ApplicationControllerLibrary
|                   \---resources
\---ViewControllerLibrary
    |   ViewControllerLibrary.jpr
    |
    \---public_html
        \---mobile
            \---library
                \---ViewControllerLibrary
                    \---resources
```

30.2.3 How to Set the Shared Library Root Directory

Set a value for the MAF Library Root Directory project property that will be unique across all projects in shared libraries that a MAF application consumes.

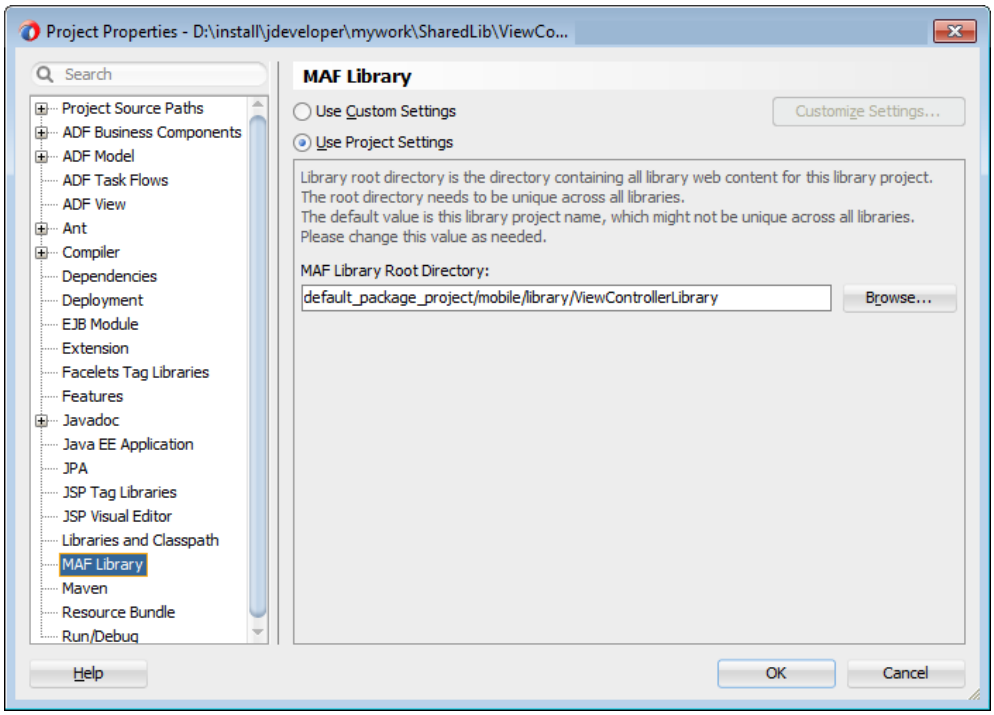
Adopt a naming convention, such as for Java package names, to ensure that the library root directory property is unique. This ensures that your shared library can be consumed by MAF applications that use multiple shared libraries. MAF will not add a shared library to a consuming MAF application if the application already includes a shared library with the same value for the library root directory properties.

To set the library root directory:

1. In the Applications window, double-click the project for which you want to set the MAF Library Root Directory project property (for example, **ViewControllerLibrary**).
2. In the Project Properties dialog, select **MAF Library** to view the MAF Library page.

   The default value for the MAF Library Root Directory project property is a concatenation of the value for the default package in the Configure Java settings page that you specify when you create the MAF application for shared library and the project name.
3. Set a unique value for the property, as shown in Figure 30-2, and click **OK**.

Figure 30-2 MAF Library Root Directory Property



30.2.4 How to Describe the Shared Library Content to Consumers

Use the `library-content.xml` file to provide information for the consumers of the shared library, such as its version number and a description of any dependencies that they need to be aware of to successfully use the shared library.

An example of a dependency that you should describe in the `library-content.xml` file is any custom Cordova plugins that the shared library requires because MAF does not package custom Cordova plugins in the shared library. Shared library consumers must obtain required custom Cordova plugins separately.

To describe the shared library content to consumers:

1. In the Applications Resources panel, expand the **Descriptors** and **ADF META-INF** nodes, and then double-click **library-content.xml**.
2. In the source editor, add a name for the shared library, a version number, and a description of the shared library's purpose.

   Example 30-2 shows a `library-content.xml` file.

> **Note:** MAF inserts the MAF version that you use to create the shared library into the `library-content.xml` file when it deploys it.

Example 30-2 library-content.xml File

```
<library-container
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.oracle.com/adf/mf/sharedlibrary"
         name="DocExample">

 <version>2.1</version>
```

```
  <library-description>
    This library includes a task flow that enables end users to edit employee
details.
  </library-description>

</library-container>
```

30.3 Adding Artifacts to a MAF Application for Shared Library

Add the artifacts that you want to share to the appropriate project in the MAF application for shared library.

The MAF application for shared library template creates two projects (`ApplicationControllerLibrary` and `ViewControllerLibrary`). Add resources to the library project that corresponds to the project in a MAF application where the resource will be consumed. For example, the custom HTML page that you add to display an alternative launch screen to the default MAF launch screen on iOS devices is packaged in the `ApplicationController` project of a MAF application. If you want to distribute this artifact in a shared library, add it to the `ApplicationControllerLibrary` project within the MAF application for shared library.

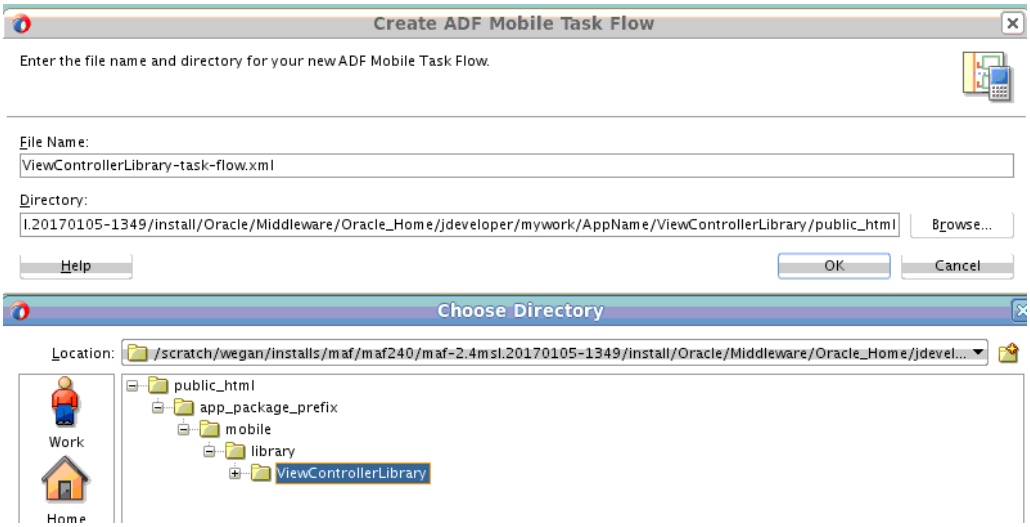
In general, use the `ApplicationControllerLibrary` project to share CSS files, images, and HTML pages, such as a custom login page. Use the `ViewControllerLibrary` project to share AMX page fragments, task flows, CSS files, images, and JavaScript files.

Make sure that you add the dependencies necessary to use an artifact to the shared library. For example, include login connections and Java classes that underlie a data control. MAF packages these dependencies in the shared library when it deploys the shared library so that they are available to the MAF application that consumes the shared library. One exception is Cordova plugins. MAF does not package Cordova plugins in a shared library. For a Cordova plugin, add an entry describing the dependency to the `library-content.xml` file, as described in How to Describe the Shared Library Content to Consumers, so that consumers of the shared library know that they need to obtain the Cordova plugin separately from the shared library. Consumers of the shared library can also refer to the `jar-maf-plugins.xml` file that MAF generates when it deploys the shared library to verify that they have registered all required Cordova plugins in the MAF application.

Use JDeveloper's New Gallery to create the artifacts that you want to add to the shared library. If, for example, you want to add a MAF login connection to the shared library, invoke the New Gallery and enter "`login`" in the New Gallery's search input field to quickly locate the MAF Login Server Connection entry. JDeveloper generates the artifacts and associated files and metadata in the appropriate project. The creation of a login connection in the shared library, for example, results in the creation of a `connections.xml` file in the `ADF META-INF` directory of the shared library. Follow this approach for other artifacts that you want to add to the shared library (task flows, AMX page fragments, and so on).

Create all artifacts that you want to share under the library root directory. Note that JDeveloper does not default to the library root directory when you create a task flow or CSS file in a shared library. Navigate to the library root directory or a sub-directory of the root directory to create these artifacts. If you do not do this, the artifacts that you want to share will not be packaged in the deployed shared library. In Figure 30-3, for example, choose the **ViewControllerLibrary** directory or a sub-directory to create the task flow.

Figure 30-3 Creating a Task Flow in the Library's Root Directory



30.4 Deploying a MAF Application for Shared Library

Deploy a MAF application for shared library to test the shared library you created or to distribute a completed shared library to the application developers who will use it in their MAF applications.

Once you deploy a MAF application for shared library, you can distribute it in a ZIP file to other MAF application developers. These developers extract the ZIP and add a file system connection to the extract directory from JDeveloper's Resources window to consume the shared library in their MAF application. Before you distribute it, you can test your shared library by creating a file system connection to the container directory of the `deploy` directory (`MAFappForSharedLibRootDir/deploy/MafSharedLibraryContainer`) in JDeveloper's Resources window and consuming the shared library in a MAF application that you create for testing. See Consuming a Shared Library in a MAF Application.

MAF provides the following ready-to-use deployment profiles:

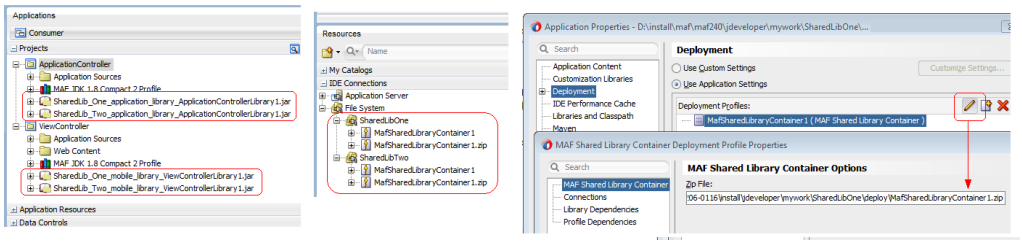
- `MafSharedLibraryContainer1`
- `ApplicationControllerLibrary_MafSharedLibraryArchive1 to JAR file`
- `ViewControllerLibrary_MafSharedLibraryArchive1 to JAR file`

Use the above deployment profiles or create your own deployment profile if, for example, you want to customize the deployment options, such as the name of the ZIP file that JDeveloper produces.

It is good practice to verify the file names produced by the deployment profiles are unique and easily distinguishable by the consumers of the shared library. By default, the `MafSharedLibraryContainer1` deployment profile names the ZIP file that it produces `MafSharedLibraryContainer1.zip`. Similarly, the project-level deployment profiles (`ViewControllerLibrary` and `ApplicationControllerLibrary`) produce JAR files that append `ViewControllerLibrary1.jar` and `ApplicationControllerLibrary1.jar` to the name of the generated JAR files. On the left of Figure 30-4 is the Applications and Resources windows view of a MAF application that consumes two shared libraries (`SharedLibOne` and `SharedLibTwo`) with default file names for the

generated JAR files and ZIP file. The right of Figure 30-4 shows the MAF Shared Library Container Options page where you can edit the default name for the generated ZIP file. Editing the deployment profiles that generate the ZIP and JAR file names may make it easier for end users to distinguish between shared libraries and avoid deployment issues. For more information about editing deployment profiles, see Working with Deployment Profiles.

Figure 30-4 Shared Libraries with Default File Names and Deployment Profile to Edit Default File Names



30.4.1 How to Deploy a MAF Application for Shared Library

Deploy the MAF application for shared library using a MAF Shared Library Container deployment profile that deploys the shared library to a ZIP file.

To deploy a MAF application for shared library:

1. From JDeveloper's main menu, select **Application**, then **Deploy**, and then **MafSharedLibraryContainer1**.

   Where **MafSharedLibraryContainer1** is the name of the deployment profile.

2. Select the **Deploy to MAF Shared Library Container ZIP file** option from the list in the dialog that appears and click **Finish**.

30.4.2 What Happens When You Deploy a MAF Application for Shared Library

JDeveloper uses the `MafSharedLibraryContainer` deployment profile to create a ZIP file (default name `MafSharedLibraryContainer1.zip`) that can contain the following artifacts:

```
//Contents of MafSharedLibraryContainer1.zip
|   ApplicationControllerLibrary_MafSharedLibraryArchive1.jar
|   ViewControllerLibrary_MafSharedLibraryArchive1.jar
|
\---META-INF
        jar-maf-plugins.xml
        library-content.xml
        MANIFEST.MF
        oracle.adf.common.services.ResourceService.sva
```

The JAR files in the ZIP contain the resources from each of the respective projects within the MAF application for shared library. However, you should distribute the ZIP file to shared library consumers.

30.5 Consuming a Shared Library in a MAF Application

Create a File System IDE connection to a shared library so that you can reuse the artifacts it contains in your MAF application.

Review the content of the `library-content.xml` file to determine if there are dependencies that you need to implement to use the shared library successfully in your MAF application. For example, you may need to obtain and register a Cordova plugin in your MAF application. A shared library author should have identified such dependencies in the `library-content.xml` file before distributing the ZIP file that contains the shared library. You can view the content of the `library-content.xml` file after you create a File System IDE connection to the shared library's directory.

Once you have added the shared library to your application, you can use artifacts like AMX page fragments from the shared library in AMX pages of the MAF application that consumes the shared library.

30.5.1 How to Consume a Shared Library in a MAF Application

Add a File System IDE connection to the directory that contains the extract from the shared library ZIP file. This makes the artifacts in the shared library available to consume by your MAF application from JDeveloper's Resources window.

If you are a shared library author who is testing a shared library, create the File System IDE connection to the child container directory of the `deploy` directory of the MAF application for shared library (`MAFappForSharedLibRootDir/deploy/MafSharedLibraryContainer`) and consume it in the MAF application that you created to test the shared library.

To consume a shared library in a MAF application:

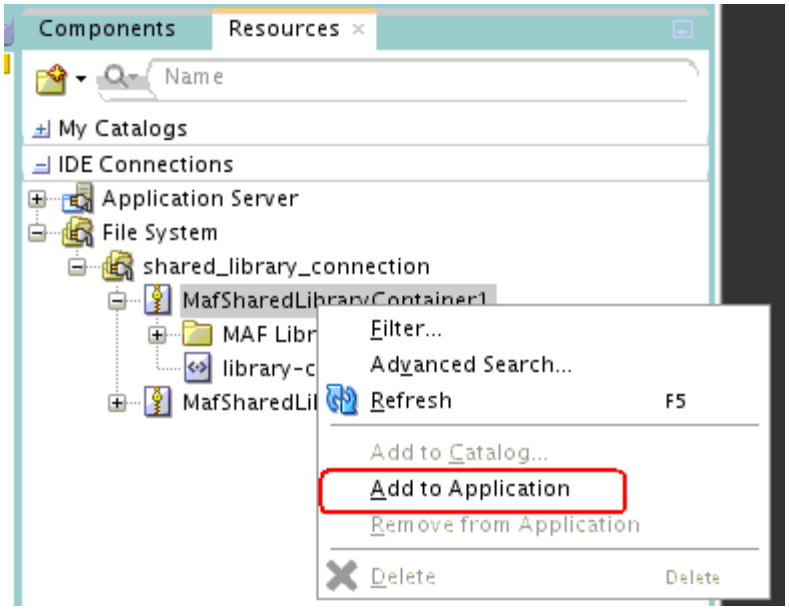
1. In the Resources window, select **New**, then **IDE Connections**, and then **File System**.
2. Complete the File Systems Connection dialog to create a file connection to the directory that contains the extracted shared library ZIP file. See the Oracle JDeveloper Online help for more information.
3. Select a project in your MAF application, for example **ApplicationController**.

   Note: If you have multiple ViewController projects in your MAF application, select the ViewController project to which you want to add the shared library resources. If you do not select a ViewController project, MAF adds the shared library resources to the first ViewController project in the Applications window.

4. Right-click the MAF Library, as shown in Figure 30-5, and select **Add to Application**.

   Tip: You can remove a shared library from a MAF application by selecting the same directory and choosing the Remove from Application option that will be enabled when a MAF application consumes a shared library.

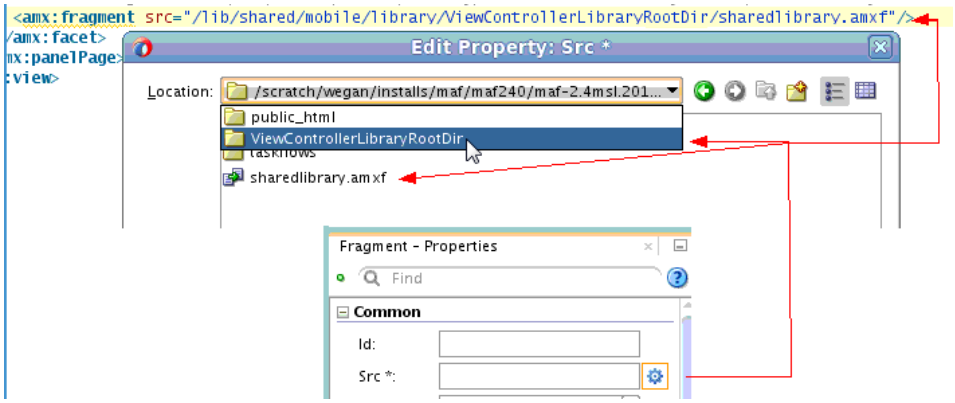
Figure 30-5 Shared Library in JDeveloper's Resources Window



5. Use the appropriate dialog picker(s) from within the MAF application to reference the resources from the shared library that you want to use in the MAF application.

   a. To reference an AMX page fragment from a shared library, add an AMX page fragment to your AMX page and set its `src` attribute to the location of the page fragment in the shared library, as shown by Figure 30-6.

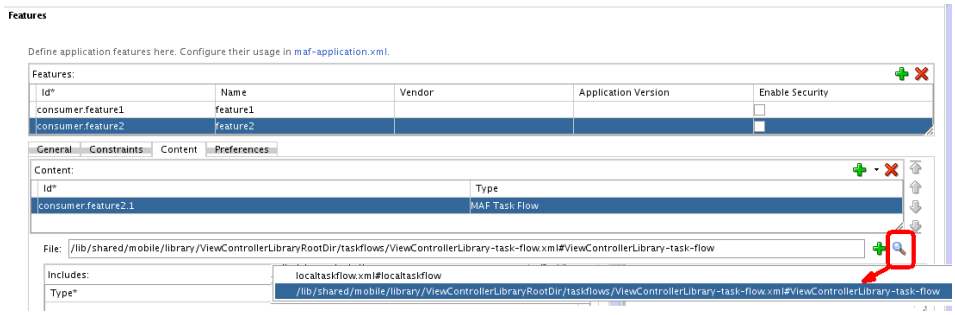
   Figure 30-6 Using an AMX Page Fragment from a Shared Library



   b. To use a task flow from a shared library, you can reference it from the Content tab of the `maf-feature.xml` file's overview editor or add a task flow call activity to a task flow.

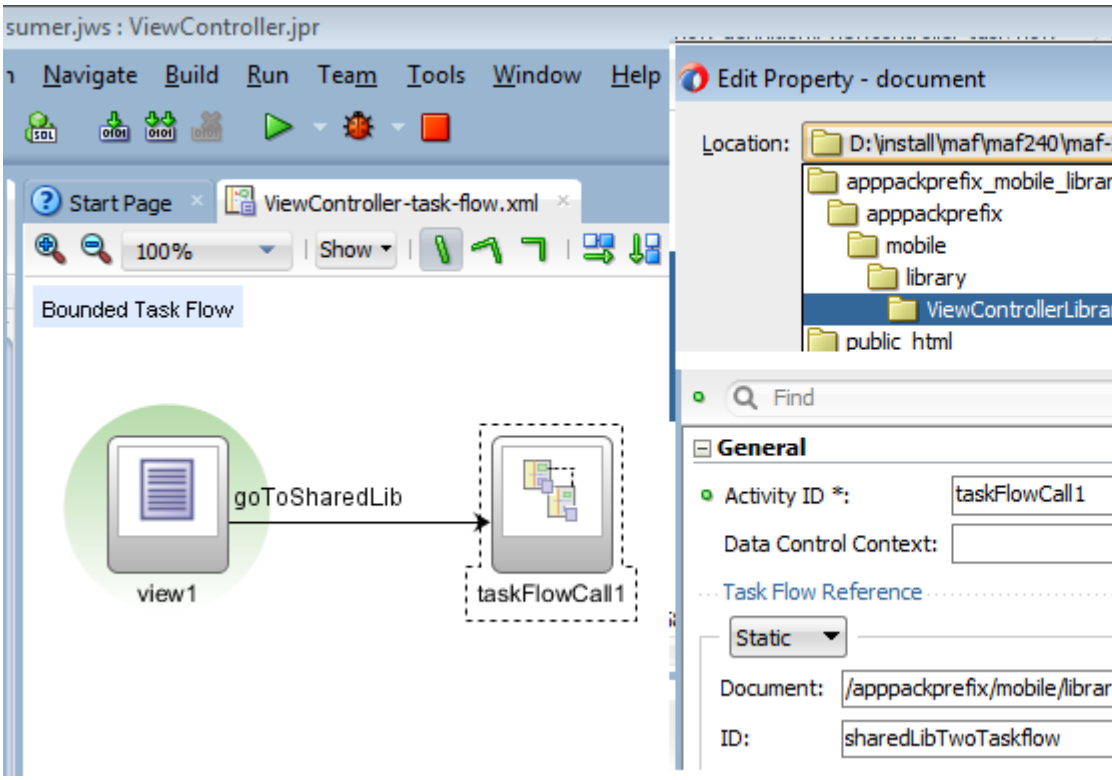
      To reference it from the Content tab of the `maf-feature.xml` file's overview editor, click on the **Browse** icon in the Content tab of the `maf-feature.xml` file's overview editor and navigate to select the task flow from the shared library, as shown by Figure 30-7.

Figure 30-7 Using a Task Flow from a Shared Library



To use a task flow call activity, add the task flow call activity to your task flow and use the Edit Property dialog that you invoke from the Properties window to navigate to the task flow in the shared library, as shown by Figure 30-8.

Figure 30-8 Calling a Task Flow from a Shared Library with a Task Flow Call Activity



One use case where you cannot use dialog pickers is to reference images from a shared library in a CSS file in your MAF application. Reference the image(s) using a relative file path from the current CSS file to the image file in the shared library.

To construct the relative path, navigate from the current CSS file to the Web Content folder (the `public_html` on your file system). Then append the value of the shared library project's library root directory property and any additional sub-directories plus the file name of the image that you want to reference.

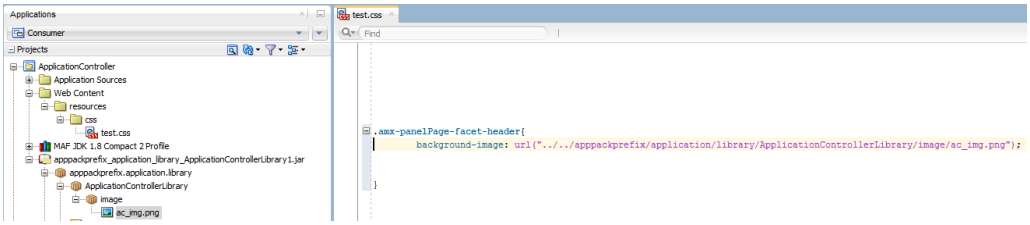
Figure 30-9 shows an example where you reference the `ac_img.png` image file in an `image` sub-directory of a shared library project that has a library root directory property of `apppackprefix/application/library/`

`ApplicationControllerLibrary`. The CSS file that references the image is two directories away from the `public_html` directory. As a result, the relative file path that you use for the image in the CSS file is the following:

```
"../../apppackprefix/application/library/
ApplicationControllerLibrary/image/ac_img.png"
```

> **Tip:** Select **View** > **Application Projects** > **Show Libraries** in JDeveloper to view the shared libraries in the Applications window, as shown in Figure 30-9.

Figure 30-9 Referencing an Image from a Shared Library in a CSS File

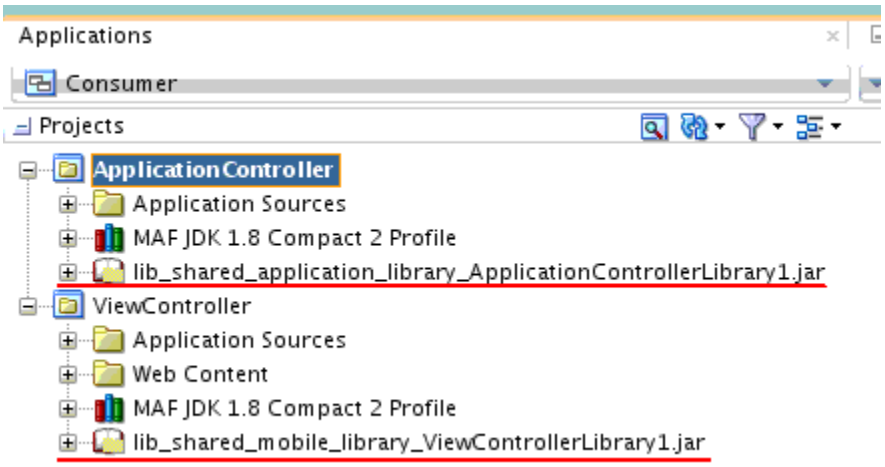


30.5.2 What Happens When You Consume a Shared Library in a MAF Application

MAF adds the content of the `ApplicationControllerLibrary` project from the shared library to the `ApplicationController` project in the MAF application. Similarly, the content of the `ViewControllerLibrary` project from the shared library is added to the `ViewController` project in the MAF application.

MAF adds the JAR files in the shared library to the consuming MAF application. You can view these JAR files in the consuming MAF application's Applications window, as shown by Figure 30-10. Select **View** > **Application Projects** > **Show Libraries** in JDeveloper to view these libraries.

Figure 30-10 Shared Libraries in a Consuming MAF Application



MAF will not add the shared library to the MAF application if the shared library was created with a different version of MAF to the version that you are currently using to develop your consuming MAF application. That is, a shared library developed with release 2.4.0.0.1 will work with a MAF application developed with release 2.4.0.0.2, but not with release 2.4.0.1.1. Shared library authors need to upgrade the shared library to support the release of MAF that you use to build your consuming MAF application. MAF will also not add the shared library to the MAF application if its existing library

root directory matches the library root directory of an existing shared library in use by the MAF application. For information about changing the value of a shared library root directory, see How to Set the Shared Library Root Directory.

Connections that you define in the shared library are merged with connections in the MAF application that consumes the shared library. MAF adds a comment, similar to the following example, that identifies each connection that it adds to the MAF application's `connections.xml` file from the shared library.

```
<!--
 The following connections were added by MAF Shared Library 'DocExample': test
-->
```

The comment identifies the source of the connection and the name of the connection. In the previous example, `'DocExample'` refers to the name of the shared library, the value that is specified for the `name` attribute in the `library-content.xml` file, and `test` refers to the name of the connection. If no value is specified for the `name` attribute in the `library-content.xml` file, MAF uses the name of the container ZIP (for example, `MafSharedLibraryContainer1`). This comment can be useful in helping you to identify connections that you can remove from the MAF application's `connections.xml` file if, for example, the MAF application no longer uses the shared library that led to the merge of the connection into the `connections.xml` file or, as discussed later, you want to upgrade to a newer version of the shared library. JDeveloper also displays a message in its Log window that identifies the connections added from the shared library and if further configuration is required before the connection can be used, as shown by the following example:

```
[04:42:06 AM] These connections were added by the Add to Application operation.
[04:42:06 AM] {
[04:42:06 AM]    test - new, incomplete LoginConnection connection will be added.
Configure before use.
[04:42:06 AM] }
```

If MAF encounters duplicate connections defined in the shared library and in the consuming MAF application, it will not merge connection information from the shared library and removes the shared library from the MAF application. The consuming MAF application's `connections.xml` file remains unchanged. For the scenario where you want to upgrade to a newer version of the shared library that your MAF application consumes, remove the pre-existing connections from your MAF application or coordinate with the shared library author to remove the pre-existing connections from the shared library so that MAF does not prevent the upgrade due to the existence of duplicate connections in the MAF application and the shared library.

30.6 Deploying a MAF Application with Duplicate Shared Library Archive File Names

Describes how to deploy a consuming MAF application with shared libraries that use the same file names for the JARs that package the shared library content.

Assume that you have a consumer application that consumes two shared libraries (`SharedLib_One` and `SharedLib_Two`). These shared libraries have unique values for the root directory property. For example, `test/mobile/library/ViewControllerLibrary_SharedLib_One` is the value for the `ViewControllerLibrary` project of the `SharedLib_One` shared library. The values for the root directory in the other projects are similarly unique in that they append the name of the shared library application (`SharedLib_One` or `SharedLib_Two`). However, the author of these shared library applications chose the same value (`test`) for Application Package Prefix when creating both shared libraries.

The files names for the ZIP and JAR files that the deployment profiles of these shared libraries generate are left to the default values, so both shared library applications are packaged in ZIP and JAR files with the same file names:

```
|   MafSharedLibraryContainer1.zip
...
    |   test_application_library_ApplicationControllerLibrary1.jar
    |   test_mobile_library_ViewControllerLibrary1.jar
```

As each project within both shared library applications has a unique value for the root directory property, the consumer application can successfully add the two shared libraries. However, an attempt to deploy the consumer application after adding the two shared library applications fails with the following message appearing in JDeveloper's Log window.

```
[11:45:00 AM] ----  Deployment started.  ----
...
[11:45:07 AM] Deployment cancelled.
[11:45:07 AM] ----  Deployment incomplete  ----.
[11:45:07 AM] Failed deployment to Android emulator. Encountered exception: Cannot
deploy the
              MAF Shared Libraries included in the application because one or more
MAF Shared Library
              archive files have the same name and are configured to be included in
the same archive
              destination directory.
```

Deployment fails because the project-level FAR deployment profiles invoked by the application-level deployment profile package the shared library JARs within a `lib` directory in the generated deployment JAR. This is `ViewController_MobileFeatureArchive1.jar` in the case of a `ViewController` project. When MAF encounters two shared library JARs with identical file names, it stops the deployment rather than overwrite one shared library JAR with another in the `lib` directory.

The above scenario could have been avoided if the shared library authors adopted a unique naming convention for the Application Package Prefix value when they initially created the shared libraries or edited the deployment profiles to generate file names other than the default values shown above. You (the application developer for the consuming MAF application) can work around this issue and successfully deploy your application by editing the project-level FAR deployment profile to create a second directory (for example, `lib2`) where MAF packages the second shared library JAR that has the same file name. This configuration change results in a successful deployment where MAF packages both shared library JARs in the following directory structure inside the project-level deployment JAR. The following example assumes a deployment to Android for the `ViewController` project.

```
ViewController_MobileFeatureArchive1.jar
    +---lib
    |   |   test_mobile_library_ViewControllerLibrary1.jar
    +---lib2
    |   |   test_mobile_library_ViewControllerLibrary1.jar
    +---META-INF
    ...
    |
    \---public_html
    ...
```

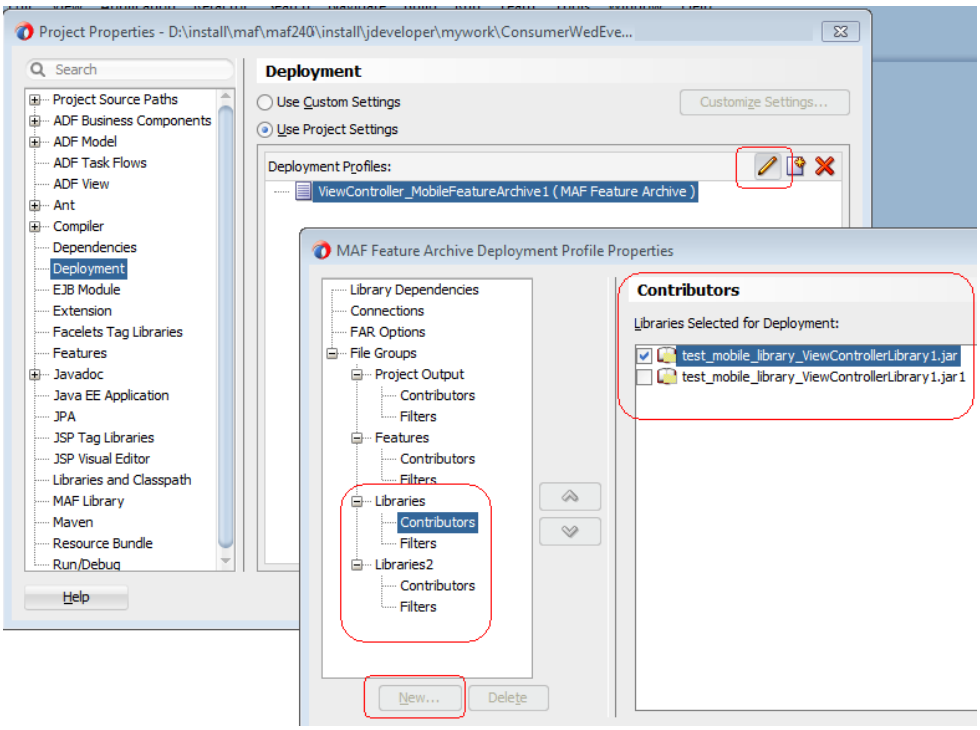
Perform this configuration change for both the `ApplicationController` and `ViewController` project-level deployment profiles.

To edit the project-level deployment profile:

1. In the Applications window, double-click the project and select the Deployment node in the Project Properties dialog that appears.
2. Select the deployment profile for the project and click the edit icon to open its properties dialog.

   The default name is *`ProjectName`*`_MobileFeatureArchive1 (MAF Feature Archive)`.
3. Select the File Groups node in the dialog that appears and click **New**.
4. In the Create File Group, enter a name, for example `Libraries2`, select the **Libraries** radio button, and then click **OK**.
5. In the newly-created `Libraries2` file group, set the **Target Directory in Archive** property to a unique value (for example, `lib2`).
6. In the Contributors node, clear the check box for the first library.
7. In the Contributors node for the first Libraries file group, clear the check box for the second library, as shown in Figure 30-11.

Figure 30-11 Creating a Second File Group Library Definition for a Shared Library Archive with Duplicate File Names



8. Click **OK** to close the dialogs and click **Build** > **Clean All and Refresh Application** prior to deploying your MAF application again.

30.7 Using Cordova Plugins in Shared Libraries

A deployed shared library does not include the Cordova plugins that it uses. You must obtain the Cordova plugin separately from the shared library and register it in the consuming MAF application.

MAF packages a modified copy of the `maf-plugins.xml` file with a filename of `jar-maf-plugins.xml` file in the `META-INF` directory of the shared library. It sets the path attribute to an empty string for all Cordova plugins in the `jar-maf-plugins.xml` file. The content of the `jar-maf-plugins.xml` file is not merged with the consuming MAF application's `maf-plugin.xml` file when a MAF application consumes the shared library. Use the `jar-maf-plugins.xml` file to verify that you have registered all Cordova plugins that the shared library requires in the consuming MAF application's `maf-plugins.xml` file.

For more information about Cordova plugins, see Using Plugins in MAF Applications .

31

Integrating MAF Applications with EMM Solutions

This chapter describes the AppConfig Community that provides tools and best practices to manage mobile applications, the MAF approach to enterprise mobile applications, and the management of MAF applications with EMM solutions from AirWatch, MobileIron, and Blackberry.

This chapter includes the following sections:

- Introduction to the AppConfig Community
- About the MAF Approach to Enterprise Mobile Applications
- Access Control for MAF Applications with EMM Solutions
- How to Manage MAF Application Configurations with EMM Solutions
- Managing MAF Applications with the AirWatch EMM Solution
- Managing MAF Applications with the MobileIron EMM Solution
- Managing MAF Applications with the Blackberry EMM Solution
- Configuring Properties in MAF Applications for Use by EMM Solutions

31.1 Introduction to the AppConfig Community

The AppConfig Community provides tools and best practices to secure, configure, deploy, and manage mobile enterprise applications.

The AppConfig Community was formed, and is maintained, by Enterprise Mobile Management (EMM) organizations: VMware AirWatch, MobileIron, IBM MaaS360, and JAMF Software. The community works to streamline the development and deployment of mobile enterprise applications.

The tools and best practices of the community are defined by the following:

- Use of native frameworks that are made available through operating systems (OS)
- Absence of EMM - specific integrations

The AppConfig approach to developing enterprise mobile application provides a standard approach to application configuration and management because it builds upon the application security and configuration frameworks within the native OS functionality of the iOS and Android platforms. The AppConfig approach has been defined by The App Configuration for Enterprise (ACE). See About the MAF Approach to Enterprise Mobile Applications.

MAF applications support ACE capabilities such as app tunnelling, application configuration, and implementations of security polices and access control. MAF

supports application integration with third-party EMM solutions from AirWatch, MobileIron, and BlackBerry. For information about MAF integrations with EMM solutions, see Managing MAF Applications with the AirWatch EMM Solution, Managing MAF Applications with the MobileIron EMM Solution, and Managing MAF Applications with the Blackberry EMM Solution.

More information about the AppConfig Community is available at: http://appconfig.org/.

31.2 About the MAF Approach to Enterprise Mobile Applications

MAF adopts the AppConfig approach to enterprise mobile applications, and supports capabilities that have been defined by ACE.

MAF adopts the AppConfig approach to enterprise mobile applications as it neither requires proprietary Software Development Kit (SDK) nor application wrapping tools. The MAF integration with AirWatch, MobileIron, and Blackberry helps developers build EMM vendor neutral applications without containers and dual workspaces.

MAF supports application integration with third-party EMM solutions. The integration focuses on using the capabilities of the mobile operating systems to configure and secure MAF applications. MAF aims at providing the capabilities that have been defined by ACE. ACE is an initiative that defines standards for enterprise application management. ACE provides an application development framework that defines common standards for mobile application management so that an application could be managed by any vendor. More information about ACE is available at: http://www.appconfigforenterprise.org/.

MAF applications support the following ACE capabilities:

- App tunnel: An application may need to access services behind a firewall. Device level IPsec VPNs come with concerns pertaining to connectivity and security. To address these concerns, mobile operating systems provide a Per-App-VPN capability so that individual applications can tunnel their way into networks. An application tunnel is a Secure Sockets Layer (SSL) connection from an application through a gateway to backend resources. As the tunnel is provided on a Per-App basis, no rogue application can worm its way into the network.

- Application configuration: Users enter URL, port, email address, port numbers, tenant ids, skin configurations and other configurations when they set up applications. An EMM server can automatically and remotely set these configurations using the native APIs recommended by the AppConfig Community. Administrators use web consoles to enter configurations which are then pushed to applications. Developers define a set of configuration keys within their applications. EMM administrators set the same keys and values in the management console of the EMM provider, and they will be pushed to the application. See Configuring Properties in MAF Applications for Use by EMM Solutions.

- Single Sign-On: Users may need to sign-on to multiple systems, each of which may involve different user names and authentication techniques. A single sign-on (SSO) solution lets users authenticate themselves just once to access information on any of several systems. With SSO, the user is authenticated once and the authenticated identity is securely carried across the network to access resources. The application developer implements the Security Assertion Markup Language (SAML) standard to federate authentication to an Identity Provider (IDP). This SAML IDP is configured to use either Kerberos authentication or certificate authentication. The EMM solution will distribute the appropriate Kerberos

credentials and, or certificates based on the standard built in operating system API calls available to the EMM providers.

- Security policies and access control: Access control ensures that applications run only on approved devices. The capability enforces security policies at the application level. An organization requires security and data loss protection within enterprise applications to prevent sensitive data from moving outside company control.

  – Encryption: EMM vendors provide data protection for enterprise applications by enforcing a passcode policy on the device. Administrators can enable device level encryption by setting a passcode policy on the device.

  – Managed Open In: This is a mobile application management feature that restricts the flow of corporate data on iOS devices to only those applications that are under IT control.

  Enterprises may also want to disable application capabilities for security reasons.

  Common implementations of custom security polices include:

  – Disable copy and paste – the ability to disable the copy and paste capability from within the application

  – Default email settings – the ability to specify the default email application to be used to send email messages within the application

  – Disable use of camera - the ability to disable the use of camera

  – Disable capture of screenshots - the ability to disable the capture of screenshots

31.3 Access Control for MAF Applications with EMM Solutions

MAF uses the SSO certificate, application tunnel, and application configuration methods to enforce access control on MAF devices that are managed by AirWatch, MobileIron, and Blackberry EMM solutions.

Access control prevents users from logging into applications which are downloaded directly from iTunes and Google Play stores.

Access control may be enforced in three ways:

- Using SSO certificate: SSO authentication can use certificates. Access is controlled by provisioning a certificate for single sign on, which will only be made available to compliant applications on managed devices. A user who tries to download an application from the iTunes store or the Google Play store as a personal application on an unmanaged device will be unable to authenticate and log into the application.

- Using Application Tunnel: Access control can be enforced using the Application Tunnel capability. An enterprise can configure the authentication page of an application so that it only accepts connections from users who come through the secure Application Tunnel, based on IP address. The Application Tunnel capability is only available for compliant applications on managed devices. A user who tries to download the applications from the iTunes store or the Google Play store as a personal application on an unmanaged device will not be able to authenticate and log into the application.

- Using application configuration: Leverage application configurations defined within MAF applications to allow or deny access to an application. The application will use the value it received in the configuration key, and grant access to the application if it is set to true.

31.4 How to Manage MAF Application Configurations with EMM Solutions

MAF applications integrate with EMM solutions to set application-level configurations remotely on the EMM server.

Integration with EMM solutions such as AirWatch, MobileIron, and Blackberry provides MAF applications the capability to set application-level configurations remotely on the EMM server, which can then be accessed by the MAF applications.

Application configurations can simplify the setup process for users. The AirWatch, MobileIron, or Blackberry server sends a set of configuration keys which are then defined by developers. An organization administrator sets the keys and values in the EMM administrative web console from where they will be sent to MAF applications.

MAF applications implement backend service configurations such as URL, port, use SSL, group or tenant code, and user configurations such as user name, email, and domain.

Custom security policies can be enforced using application configurations. These custom security policies are commonly implemented:

- Disable Public Cloud Sync: the ability to disable the syncing of application data with public clouds such as Dropbox
- Disable Copy and Paste: the ability to disable the copy and paste capability from within the application

31.5 Managing MAF Applications with the AirWatch EMM Solution

Integration with the AirWatch EMM solution helps MAF applications implement data leak protection by means of security policies.

AirWatch provides an EMM solution to secure and manage applications. AirWatch has three development approaches to provide core application feature sets: the SDK, app wrapping, and the approach that follows the AppConfig Community. When integrated with AirWatch, MAF follows the AppConfig approach. MAF does not support SDK and App Wrapping from AirWatch. MAF only supports the AirWatch ability to leverage native standards to manage applications.

MAF applications can be managed by means of the AirWatch Administrative Console. The console allows the EMM administrators to create iOS configuration profiles, and Android for Work configuration profiles, and apply them to various managed devices which are enrolled into the AirWatch Administrative Console. When users enroll their devices into AirWatch Agent App, all the configuration profiles which are assigned to their device get downloaded, and get applied. The configuration profile contain the restrictions which allow EMM administrators to enable or disable a specific functionality such as camera or Managed Open In within the application. The configuration profiles also contain Per App level configuration information which allow secure tunneling between MAF applications and various backend services, which are hosted behind the firewall, and are used by MAF applications .

Information about the AirWatch EMM platform is available at: https://www.vmware.com/products/enterprise-mobility-management.html.

Core Application Feature Sets

MAF uses the EMM from AirWatch technologies to secure its applications. MAF uses the ACE to integrate MAF applications with AirWatch. Devices may be enrolled in AirWatch, applications may be installed from the AirWatch App Catalog, or internal or public applications may be uploaded to the AirWatch Administrative Console. Integration with AirWatch helps MAF implement data leak protection through the following security policies.

- Encryption: MAF applications on the iOS platform, Android 5.0, and higher platform versions provide the ability to enable encryption. When encryption is enabled, MAF uses the native OS encryption to encrypt the content of the entire device, including applications.
- Managed Open-In: The ability to open the documents stored in managed applications in other unmanaged applications like Dropbox or Box is available on the iOS platform, Android 5.0, and higher platform versions. On the iOS platform, when this restriction is enabled, it is applicable to all the applications on the device. When you set this restriction, you turn off the ability to share documents through email. In MAF, the Email Device Service is turned off. When you enable the Open In restriction on the Android platform, the restriction is applicable to each application and not to the whole device.
- Camera: The capability to enable or disable the camera on the device is available on the iOS platform, Android 5.0, and higher platform versions. On the iOS platform, the camera restriction, when enabled, is applicable to all the applications on the device. This restriction is not applied to each application. On the Android platform, the camera restriction is applied to each application and not to the device.
- Email: An iOS profile has no restriction which directly controls the email access at the device or application level. Setting the Open In restriction turns off the ability to share documents by means of email. In MAF, the Open In restriction turns off the Email Device Service.
- App Tunnelling with AirWatch Tunnel: MAF applications on the iOS platform, Android 5, and higher platform versions are provided the Per App VPN mode capability, an OS-level capability available for individual applications on a mobile device. AirWatch Tunnel is a server component that is installed and configured with the AirWatch Administrative Console. AirWatch Tunnel uses native operating system APIs to secure data-in-transit between MAF applications and the secure enterprise network. The secure tunnel isolates the application when it communicates with the network.
- Secure browser integration: Users who want to access web content from MAF applications are redirected from the application to the AirWatch Secure Browser. Tapping on a GoLink within a MAF application launches the AirWatch Secure Browser client. The EMM administrator sets policies in the AirWatch console. When the secure browser client is launched, the policies are applied to the application, and depending on compliance with the set policies, content is either blocked or displayed.
- Secure email integration: Users who want to compose new email or perform tasks such as attach a document are redirected from the MAF application to the AirWatch Secure Browser. AirWatch provides URL schemes to launch the secure email client. Tapping on a GoLink within a MAF application launches the AirWatch Secure Email client. The EMM administrator sets policies in the

AirWatch console. When the secure email client is launched, the policies are applied to the application, and compliance with the policies decides whether the user is allowed to attach files or blocked from doing so.

You can download the VMware AirWatch Mobile Device Management Guide at: http://mobile34.ca/wp-content/uploads/2016/02/Mobile-Device-Management-Guide-v8_3.pdf.

AirWatch documentation is available at: https://resources.air-watch.com/category/Documentation.

31.6 Managing MAF Applications with the MobileIron EMM Solution

Integration with the EMM solution from MobileIron helps MAF use the layered security approach of MobileIron to implement data loss prevention on mobile devices.

MAF integrates with the EMM from MobileIron technologies to secure applications. MobileIron provides a wrapping solution, an SDK, and also supports the ACE standard. MAF does not support the wrapping tool or SDK from MobileIron. MAF uses ACE to secure its applications. With ACE, developers can develop applications that work across EMM solutions.

MAF applications can be managed by means of the MobileIron Core administration console. The Core administration console allows IT administrators to create iOS configuration profiles and Android for Work configuration profiles and apply them to various managed devices which are enrolled into the MobileIron Core console.

When users enroll their devices into the MobileIron Agent application, all the configuration profiles, which are assigned to their device, get downloaded and applied. The configuration profile contains the restrictions that allow IT administrators enable or disable a specific functionality such as camera or Managed Open In within the application.

The configuration profiles also contain Per App level configuration information that allows secure tunneling between MAF applications and various backend services that are hosted behind the firewall. The backend services are used by MAF applications.

MobileIron Security Model

MobileIron uses a layered security approach to implement data loss prevention on mobile devices. The layered data loss prevention model includes basic controls, which directly address the requirements; supplemental controls, which provide additional controls; and compensating controls, which apply when no basic control is available. For example, for authentication, a device password provides the basic control while biometric authentication may supplement the basic control.

The layered security approach is implemented using the following controls.

Encryption

Encryption is available for iOS 9, Android 5.0 and higher version devices. MobileIron encryption for data-at-rest and data-in-motion has FIPS 140-2 validation and complements the embedded encryption capabilities of the operating systems. MobileIron provides the following types of encryption:

- Encryption of all enterprise data-at-rest on the device: The EMM solution enforces device passwords to enable device level encryption that is available to all applications.
- Encryption of all enterprise data-in-motion to and from the device: Enterprise mobile data, which includes email, applications, documents, and web pages,

flows through MobileIron Sentry, the gateway that protects against attacks and interception through the use of digital certificates and transport layer encryption.

- Encryption of all enterprise data in secure applications: In addition to the iOS and Android embedded encryption, MobileIron uses containerization to provide additional security controls, including encryption.

Data Sharing

The Open In function of mobile operating systems allows applications to share data. MobileIron allows IT to control which applications use this function to access application data. Setting this restriction on the iOS platform also turns off the ability to share documents through email. In MAF, the Email Device Service is turned off.

Copy and Paste functionality: The iOS platform does not provide a native restriction to control the ability to copy and paste content between managed applications. MAF developers can impose this restriction by implementing a custom Cordova plugin, which in turn calls the native iOS APIs to clear the buffer.

Capture screenshots functionality: MobileIron can also disable the screenshot capture function across the entire device.

Email: No restriction pertaining to email is available for the Android platform. For the iOS platform, although there is no restriction to manage the email service, by setting the Managed Open-In restriction, email within a managed application can be turned off.

Network Security

Application tunneling: MobileIron provides secure application-level tunneling for all enterprise data, including email, applications, documents, and web traffic. IT can separate the flow of enterprise data transmitted on a secure channel from non-enterprise data.

VPN: For standardized on device-wide VPN technology, MobileIron configures the VPN service to provide a secure channel for data. MobileIron provides secure access to business data through PerApp VPN.

Per-App-VPN with MobileIron Tunnel supports the following in MAF applications:

- Access to a basic authentication protected API hosted on non-Oracle cloud by tunneling the network request through MobileIron Sentry which acts as a proxy server
- Access to an unsecured API hosted in Oracle Mobile Cloud Service by tunneling the network request through MobileIron Sentry which acts as a proxy server
- Access to a basic authentication protected API hosted on an Oracle Mobile Cloud Service by tunneling the network request through MobileIron Sentry which acts as a proxy server

MobileIron Sentry is the gateway in the MobileIron platform that manages, encrypts, and secures traffic between the mobile device and back-end enterprise systems.

Secure Browser Integration

Users who want to access web content from a MAF application are redirected from the application to the MobileIron Secure Browser. Tapping on a GoLink within the MAF application launches the MobileIron Secure Browser client. MobileIron provides URL schemes to launch the secure browser client. The EMM administrator sets policies in the MobileIron console. When the secure browser client is launched, the policies are

applied to the application and depending on compliance, content is either blocked or displayed.

Secure Email Integration

Users who want to compose new email, or perform tasks such as attach a document are redirected from a MAF application to the MobileIron Secure Email client. MobileIron provides URL schemes to launch the secure email client. Tapping on an email link within a MAF application launches the MobileIron Secure Email client. The EMM administrator sets policies in the MobileIron console. When the secure email client is launched, the policies are applied to the application, and compliance with the policies decides whether the user is allowed to attach files or blocked from doing so.

MobileIron documentation is available at: https://community.mobileiron.com/.

31.7 Managing MAF Applications with the Blackberry EMM Solution

Integration with the Blackberry enterprise mobile management solution helps MAF secure applications by means of the BlackBerry security policies.

MAF applications can be managed by means of the BlackBerry Administrative Console. The Administrative Console allows EMM administrators to create configuration profiles, such as iOS configuration profiles, and apply them to various managed devices which are enrolled into the Blackberry Administrative Console. When users enroll their devices into the BlackBerry Agent application, all the configuration profiles that are assigned to their device get downloaded and applied. The configuration profile contain the restrictions that allow EMM administrators to enable or disable a specific functionality such as camera or Managed Open In within the application. The configuration profiles also contain Per-App level configuration information. This information allows secure tunneling between MAF applications and various backend services, which are hosted behind the firewall. The backend services are used by MAF applications.

The Blackberry EMM solution offers MAF applications the following capabilities:

- Encryption: On the iOS platform, BES provides the ability to encrypt the content stored on the device and applications using the native OS encryption. BES 12 uses a FIPS-validated cryptographic module to encrypt all the data that it stores directly, and writes indirectly to files. BES 12 supports both full device and work space encryption. IT policy rules can also be used to force some devices to encrypt data.

- Managed Open In is a security feature released in the Apple iOS 7 mobile operating system. The feature allows IT to configure which applications employees can use to access data. Supported on iOS 9, this feature manages the ability to open documents stored in managed applications, in other unmanaged applications such as Dropbox, and Box. BES supports Open-in management as a mobile application management (MAM) measure, and uses it as part of an enterprise data loss prevention (DLP) strategy for iOS devices.

- Copy and Paste feature: The restriction that pertains to Copy and Paste implements a method of data loss prevention by restricting the copying and pasting of text and, or images. This feature manages the ability to copy and paste content between different mobile applications on iOS devices. A custom Cordova plugin could be used to enable or disable copy and paste feature between managed and unmanaged applications.

- Device features: Device features to restrict the usage of camera during application use allow administrators to restrict camera usage in an enterprise setting. Tools

can also prevent screenshots from being taken. These restrictions are available for applications on iOS 9 and Android devices. Personal cloud applications such as email are not designed to convey or store enterprise data. Mobile device owners who use them for such purposes expose the enterprise to greater risk of data loss. The restriction that pertains to email determines from which application a user may send email messages.

- Per-App VPN: Virtual private network (VPN) on demand and direct application tunnels to automatically route all communication from a specific application through a secure channel back to the enterprise network is available for applications on both iOS and Android platforms with MobileIron Tunnel. A VPN provides an encrypted tunnel between a device and the enterprise network. A VPN solution consists of a VPN client on a device and a VPN concentrator. The device uses the VPN client to authenticate with the VPN concentrator, which acts as the gateway to the enterprise network. Use Per-App VPN to specify which work applications and secured applications on devices use a VPN for the transit of data. Per-App VPN helps decrease the VPN load on an organization by enabling only discretionary traffic use the VPN. For example, the VPN may be used to access application servers or web pages behind the corporate firewall.

Documentation about Blackberry security is available at: http://help.blackberry.com/en/bes12-security/current/.

31.8 Configuring Properties in MAF Applications for Use by EMM Solutions

Configure properties in the `maf-application.xml` file of a MAF application. Administrators can use EMM software to configure values for the properties when the application is deployed to users.

You can configure the properties in the `maf-application.xml` file application of the application using the `<adfmf:emmAppConfig>` element. The following sample `maf-application.xml` file shows a number of properties that are defined.

```
<adfmf:emmAppConfig>
<adfmf:property name="serverURL" type="String" description="URL to
connect the backend service"/>
<adfmf:property name="port" type="Integer" description="Port number of the backend
service"/>
<adfmf:property name="enableEncryption" type="Boolean" description="Turn on app
level encryption"/>
<adfmf:property name="refreshDate" type="Date" description="Date on which
application will be refreshed"/>
</adfmf:emmAppConfig>
```

An EMM administrator configures values for these properties in an EMM console. The EMM software then pushes the values to the devices on which your MAF application is installed. This feature is only supported for MAF applications that are deployed to the Android and iOS platforms. Make sure that the EMM software supports the data types that you specify in the `<adfmf:emmAppConfig>` element. In the example above, the specified properties have the following data types: `String`, `Integer`, `Boolean`, and `Date`.

See the documentation of the EMM vendor for information about how to configure the corresponding property values in the EMM console and the data types that the EMM software supports.

You can read the property values in the application lifecycle of your MAF application using the `#{EMMConfigProperties}`EL expression. For example, write an EL

expression as follows to read the value of the serverURL property:
`#{EMMConfigProperties.serverURL}`

You can also register your property change listener to listen to property changes by invoking the following:

```
EMMAppConfigScope.getInstance().addPropertyChangeListener(this);
```

A

Troubleshooting MAF Applications

This appendix describes problems with various aspects of MAF applications, as well as how to diagnose and resolve them.

This appendix includes the following sections:

- Problems with Input Components on iOS Simulators
- Code Signing Issues Prevent Deployment
- The credentials Attribute Causes Deployment to Fail

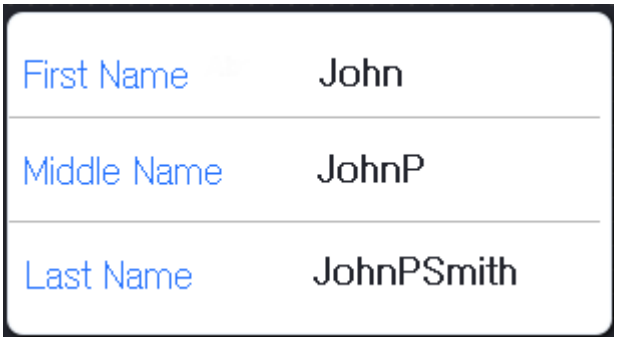
A.1 Problems with Input Components on iOS Simulators

Text in one field spills in to the next field when navigating input text fields with a mouse. Using the keyboard on the iOS simulator to navigate the fields resolves the issue.

Issue:

On MAF 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

Adding code signing data to the Mach object file resolves the issue of deployment failure caused by code signing errors on the iOS platform.

Issue:

In some iOS development environments, MAF application deployment fails because of code signing errors.

Solution:

To ensure that the MAF 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/`).

A.3 The credentials Attribute Causes Deployment to Fail

The authentication mode, if defined using the `authenticationMode` attribute in the `connections.xml` file, resolves the issue of the credentials attribute causing JDeveloper to cancel deployment.

Issue:

The presence of the `credentials` attribute defined for the `adfmf:feature` element in the `maf-feature.xml` file causes JDeveloper to cancel deployment and write an error similar to the following to the deployment log:

```
XML validation failed for file
/Users/jsmith/jdeveloper/mywork/MobileApplication/ViewController/src/META-INF/maf-
feature.xml.
[12:26:44 PM] The file contains the following errors:
Error (Line 3, Column 44): Attribute credentials not defined on element adfmf:feature
Error (Line 10, Column 49): Attribute credentials not defined on element
adfmf:feature
Error (Line 19, Column 51): Attribute credentials not defined on element
adfmf:feature
Error (Line 35, Column 69): Attribute credentials not defined on element
adfmf:feature
Error (Line 50, Column 65): Attribute credentials not defined on element
adfmf:feature
[12:26:50 PM] Deployment canceled.
[12:26:50 PM] ----  Deployment incomplete  ----.
[12:26:50 PM] XML validation failed.
```

Solution:

When you migrate an application created by ADF Mobile, you must verify that the authentication mode once defined in `maf-feature.xml` (such as `<adfmf:feature id="feature1" name="feature1" credentials="remote">`) is now defined using the `authenticationMode` attribute in the `connections.xml` file.

JDeveloper's audit rules can detect the presence of the `credentials` attribute and assist you in removing it from the `maf-feature.xml` file.

Because only the `local` and `remote` values are valid for the `autneticationMode` attribute, do not migrate the value of `none` (`<adfmf:feature id="feature1" name="feature1" credentials="none">`) to the `authenticateionMode` attribute, as doing so will cause the deployment will fail. See Overview of the Authentication Process for MAF Applications.

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:

- Using MAF APIs to Create a Custom HTML Springboard Application Feature
- The MAF Container Utilities API
- Accessing Files Using the getDirectoryPathRoot Method

B.1 Using MAF APIs to Create a Custom HTML Springboard Application Feature

Call JavaScript API extensions to add navigation functions to a custom springboard page authored in HTML.

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 What You May Need to Know About Custom Springboard Application Features with HTML Content, you can enable callbacks and use Apache Cordova by including methods in the JavaScript `<script>` tag. The following example 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). See Enabling Remote Applications to Access Container Services.

Tip:

To access (and determine the location of) the `www/js` directory, you must first deploy a MAF application and navigate 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. For a Windows deployment in the Release mode, the directory location is `appLocationOnMachine\deploy \deploymentprofilename\release\MafTemplate\www\js`. See also What You May Need to Know About Custom Springboard Application Features with HTML Content.

Note:

Because the path does not exist during design time, JDeveloper 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 API of the mobile device 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 the MAF 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 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

Custom HTML pages can invoke their own code after MAF has loaded by listening to the `showpagecomplete` event on the `handlePageShown` callback function.

The following example 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 functionality such as navigation and display of features.

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 following example 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

JavaScript being asynchronous has two callback functions added for every function: a `success` callback that returns results, and a `failed` callback that returns exceptions.

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 following example 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 following example 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 the following example 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>) {
   ...
}
```

For information on how to invoke MAF JavaScript APIs from pages defined as local HTML or remote URL, see Enabling Remote Applications to Access Container Services.

B.2.2 Using the Container Utilities API

Call the methods in the list to use the Container Utilities API.

The Container Utilities API provides the following methods:

- getApplicationInformation—Retrieves the metadata for the MAF application.
- gotoDefaultFeature—Activates the default application feature.
- gotoFeature—Activates 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—Activates 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. See Application Icon Badging.

B.2.3 getApplicationInformation

The `getApplicationInformation` method returns an `ApplicationInformation` object that contains metadata, such as the application ID and application name, about the 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 `ApplicatiaonInformation` 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 following 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

The `gotoDefaultFeature` method requests that MAF display the default application feature, the feature that appears when the application starts.

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.

The following 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

The `gotoFeature` method requests that MAF display the application feature that is 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 following example, 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 following 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

The `getFeatures` method returns an array of `FeatureInformation` objects representing the available application features that a custom springboard implementation can use.

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.

> **Note:** 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 following 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

The `getFeatureByName` 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 following example, 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 following 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

The `getFeatureById` method retrieves an application feature using the application ID.

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.FeatureInformation getFeatureById(String featureId)
                                throws oracle.adfmf.framework.exception.AdfException
```

This method's parameter, as shown in the following 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 following example 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

The `resetFeature` method resets the state of the application feature, and refreshes the user interface.

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 MAF 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 following example, 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 following example 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

The `resetApplication` method, to be used only if resetting individual application features is insufficient, resets a running application.

This method resets the running application and it should be used only when resetting individual application features is not sufficient. 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

The `gotoSpringboard` method requests MAF to 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 Configuring Application Navigation.

Within Java, this method is called as follows:

```
public static void gotoSpringboard()
```

The following example 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.

The following 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

The `showSpringboard` method requests MAF to show the springboard.

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

The `hideSpringboard` method requests MAF to 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

The `showNavigationBar` method requests MAF to show the navigation bar.

This method requests that MAF display the navigation bar.

Within Java, this method is called as follows:

```
public static void showNavigationBar()
```

The following 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 following example 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

The `hideNavigationBar` method requests MAF to hide the navigation bar.

Within Java, this method is called as follows:

```
public static void hideNavigationBar()
```

The following 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.

The following 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

The `showPreferences` method requests MAF to display the Preferences page.

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
   }
```

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 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

With the examples and the listed parameters, call the `invokeMethod`, unavailable in Java, to invoke a Java method from any class in a classpath.

This method is not available in Java. The following example 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 invokeMethod

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

Use the listed parameters and call `invokeContainerMethod` to invoke a native method on a specified class with given arguments.

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.

Table B-2 (Cont.) Parameters Passed to invokeContainerMethod

Parameter	Description
`methodName`	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 an `Object`.

```
public static java.lang.Object invokeContainerMethod(java.lang.String className,
                                                      java.lang.String methodName)
                                                      java.lang.Object[] args)
```

B.2.19 invokeContainerJavaScriptFunction

Use the `invokeContainerJavaScriptFunction` method to invoke JavaScript methods.

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 following 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 next example.

```
(function()
    {
       if (!window.application) window.application = {};

       application.testFunction = function()
       {
          var args = arguments;

          alert("APP ALERT " + args.length + " ");
          return "application.testFunction - passed";
       };
    })();
```

Because the application includes a command button that is configured with an action listener that calls this function, an end user sees the following alerts after clicking this button:

- APP ALERT 0
- APP ALERT 1
- APP ALERT 2

The pseudocode in the following example illustrates how the `invokeApplicationJavaScriptFunction` method calls the JavaScript method (`application.testFunction`) that is described in the preceding 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"} );
    }
```

See *Java API Reference for Oracle Mobile Application Framework* and the APIDemo sample application. This sample application is in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.maf/Samples`

B.2.20 sendEmail

For information, see How to Use the sendEmail Method to Enable Email.

B.2.21 sendSMS

For information, see How to Use the SendSMS Method to Enable Text Messaging.

B.2.22 Application Icon Badging

On the iOS platform, use the listed methods of the `AdfmfContainerUtilities` class either to place a badge number on a MAF application icon, or to retrieve it.

The `AdfmfContainerUtilities` class includes methods to place or retrieve a badge number on a MAF 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 MAF application icon. Returns zero (0) if the application icon is not badged.	None
`setApplicationIconBadgeNumber`	Sets the badge number on a MAF application icon.	The value of the badge (`int badge`).

Note:

Application icon badging is not supported either on the Android or the Windows platforms.

B.3 Accessing Files Using the getDirectoryPathRoot Method

Use the `getDirectoryPathRoot` method of the `AdfmfJavaUtilties` API from the Java layer to enable access to directories on the iOS, Android, and Windows platforms.

The `AdfmfJavaUtilties` API includes the `getDirectoryPathRoot` method. This method, which can only be called from the Java layer, enables access to directories on the iOS, Android, and Windows systems. As shown in the following example, this method enables access to the location of the temporary, application (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 the *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
   String downloadDir =
      AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DownloadDirectory);
}
```

B.3.1 Accessing Platform-Independent Download Locations

Use the `getDirectoryPathRoot` method to get the paths to external storage directories and the default attachments directory.

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, applications 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 `android.os.Environment.getExternalStorageDirectory` method (which retrieves the external Android storage directory, such as an SD card). For MAF 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 JDeveloper generates when you create a MAF application using the Mobile Application Framework Application template.

This appendix includes the following section:

- Introduction to MAF Application and Project Files
- About the Application Controller Project-Level Resources
- About the View Controller Project Resources
- About the MAF Application Configuration File
- About the MAF Application Feature Configuration File

C.1 Introduction to MAF Application and Project Files

An understanding of what files are generated in the ApplicationController and ViewController projects of the application is needed if you want to configure an application, application features, or files that a MAF application needs for deployment.

By default, JDeveloper creates a MAF application with two projects (ApplicationController and ViewController). The ApplicationController project contains application-wide resources such as a login page if you configure security for your MAF application. The ViewController project contains application feature resources such as HTML, AMX, or task flow files that render the content of an application feature.

Use the ViewController project to create or store artifacts that you may want to use in more than one MAF application. Consider, for example, a MAF skin that determines the look and feel of a MAF application. You may intend to reuse this artifact in multiple MAF applications. Similarly, you may want to share one or more data controls in multiple MAF applications. For this reason, create an additional ViewController project in your MAF application to store artifacts such as these that you may intend to share among multiple MAF applications. This additional ViewController project can be reused by packaging it into a FAR, as described in Reusing MAF Application Content .

See About the Application Controller Project-Level Resources and About the View Controller Project Resources.

JDeveloper 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. Example C-1 shows the files that JDeveloper generates for a newly-created MAF application.

Two of the files that you use most frequently as you develop a MAF application are the `maf-application.xml` file (application configuration) and the `maf-features.xml` file (feature configuration). See About the MAF Application Configuration File and About the MAF Application Feature Configuration File.

Example C-1 Files in a Newly-Created MAF Application

```
Folder PATH listing
Volume serial number is 52FC-1121
C:.
|   Application1.jws
|
+---.adf
|   \---META-INF
|           adf-config.xml
|           maf-application.xml
|           maf-config.xml
|           maf-plugins.xml
|           sync-config.xml
|           wsm-assembly.xml
|
+---.data
|   +---00000000
|   |       00000000.jdb
|   |       je.lck
|   |
|   \---ApplicationController
|       \---00000000
|               00000000.jdb
|               je.lck
|
+---ApplicationController
|   |   ApplicationController.jpr
|   |
|   +---adfmsrc
|   |   +---application
|   |   |       DataControls.dcx
|   |   |
|   |   \---META-INF
|   |           adfm.xml
|   |
|   +---public_html
|   \---src
|       +---application
|       |       LifeCycleListenerImpl.java
|       |
|       \---META-INF
|               maf-skins.xml
|
+---resources
|   +---android
|   |       display-hdpi-icon.png
|   |       display-land-hdpi-splashscreen.9.png
|   |       // Additional image files omitted for brevity
|   |       display-xxxhdpi-icon.png
|   |
|   +---default
|   |       MissingIcon_144x144.png
|   |
|   +---ios
|   |       Default-1104h@2x.png
```

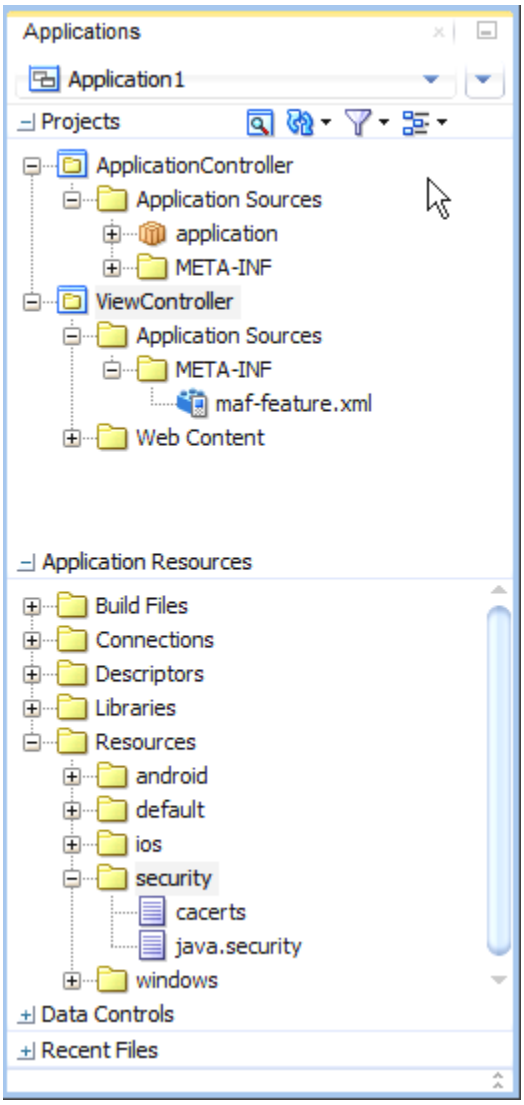
```
|   |       Default-568h@2x.png
|   |       Default-667h@2x.png
|   |       Default-Landscape-621@2x.png
|   |       Default-LandscapeRetina.png
|   |       Default-LandscapeRetina@2x.png
|   |       Default-PortraitRetina.png
|   |       Default-PortraitRetina@2x.png
|   |       Default@2x.png
|   |       Icon-120.png
|   |       // Additional image files omitted for brevity
|   |       iTunesArtwork.png
|   |
|   +---security
|   |       cacerts
|   |       java.security
|   |
|   \---windows
|           BadgeLogo.scale-100.png
|           // Additional image files omitted for brevity
|           splashscreen.scale-100.png
|
|           Square150x150Logo.scale-100.png
|
|           Wide310x150Logo.scale-100.png
|           // Additional image files omitted for brevity
|
+---src
|   \---META-INF
|           logging.properties
|           maf.properties
|
\---ViewController
    |   ViewController.jpr
    |
    +---public_html
    \---src
        \---META-INF
                maf-feature.xml
```

C.2 About the Application Controller Project-Level Resources

Files in the application controller project contain configuration information that describes the metadata of the MAF application, and can be accessed from the Application Resources pane of the Applications window.

JDeveloper generates the files for the MAF application in the application controller project. These files, described in Table C-1, contain configuration information describing the metadata of the MAF application. You access these files from the Application Resources pane of the Applications window, shown in Figure C-1.

Figure C-1 MAF Application Artifacts Accessed from the Application Resources Pane



The application controller project, 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. See About the View Controller Project Resources.

Table C-1 MAF Application-Level Artifacts Accessed Through Application Resources

Artifact(s)	File Location	Description
`maf-application.xml`	*`application workspace directory`*`\.adf\Meta-INF` For example: `JDevloper\mywork\application name\.adf\META-INF`	An XML file that defines application-level information. You can define the content for an application, its navigation behavior, and its user authentication requirements.

Table C-1 (Cont.) MAF Application-Level Artifacts Accessed Through Application Resources

Artifact(s)	File Location	Description
`maf-config.xml`	*`application workspace directory`*`\.adf\Meta-INF` For example: `JDeveloper\mywork\application name\.adf\META-INF`	Use to configure the default skin used for MAF applications. See Skinning MAF Applications.
Application images	*`application workspace directory`*`\Application Resources\resources\`	A set of images required for the deployment of applications. For Android and Windows, these include images for application icons and splash screens. For iOS, these include images for application icons. MAF applications that you deploy to iOS devices use a HTML page as the launch screen. The filename for each image indicates the purpose it serves. For example, use the `display-port-ldpi-splashscreen.9.png` image that appears under Android resources as a splash screen in portrait mode on Android devices. For information about how to override the application icons and splash screens for MAF applications you deploy to Android, see How to Add a Custom Image to an Android Application. For information about how to override the application icons for MAF applications you deploy to iOS, see Adding a Custom Image to an iOS Application. To change the launch screen for iOS devices, see Changing the Launch Screen for Your MAF Application on iOS.
`cacerts`	*`application workspace directory`*`\Application Resources\resources\Security\cacerts` For example: `JDeveloper\mywork\application name\resources\Security\cacerts`	The `cacerts` certificate file, a system-wide keystore that identifies the CA certificates to the Java virtual machine (JVM). You can update this file using the Java keytool utility. You can create a custom certificate file using keytool as described in Supporting SSL. Any certificate file must reside within the `Security` directory.
`logging.properties`	*`application workspace directory`*`\src\.META-INF\logging.properties` For example: `JDeveloper\mywork\application name\src\META-INF\logging.properties`	Enables you to set the application error logging, such as the logging level and logging console. See Using and Configuring Logging.
`maf.properties`	*`application workspace directory`*`\src\.META-INF\maf.properties` For example: `JDeveloper\mywork\application name\src\META-INF\maf.properties`	The configuration file for the JVM. Use this file to configure the application startup and heap space allotment, as well as Java and JavaScript debugging options. See How to Enable Debugging of Java Code and JavaScript.

Table C-1 (Cont.) MAF Application-Level Artifacts Accessed Through Application Resources

Artifact(s)	File Location	Description
`adf-config.xml`	*`application workspace directory`*`\.adf\META-INF` For example: `JDeveloper\mywork\application\.adf\META-INF`	Used to configure application-level settings, including the Configuration Service parameters. See also Configuring End Points Used in MAF Applications .
`connections.xml`	*`application workspace directory`*`\.adf\META-INF` For example: `JDeveloper\mywork\application name\.adf\META-INF`	The repository for all of the connections defined in the MAF application.
`wsm-assembly.xml`	*`application workspace directory`*`\.adf\META-INF` For example: `JDeveloper\mywork\application name\.adf\META-INF`	Stores the web service policy definitions used for secured web services.

Tip:

Place code that supports application-wide functionality, such as an application-level lifecycle listener, in the application controller project.

Within the application controller project itself, JDeveloper creates the artifacts listed in Table C-2.

Table C-2 Application Controller Artifacts

Artifact(s)	File Location	Description
`LifeCycleListenerImpl.java`	*`application workspace directory`*`\ApplicationController\src\application` For example: `JDeveloper\mywork\application name\ApplicationController\src\application`	The default application lifecycle listener (ALCL) for the MAF application. See Using Lifecycle Listeners in MAF Applications .
`maf-skins.xml`	*`application workspace directory`*`\ApplicationController\src\META-INF` For example: `JDeveloper\mywork\application name\ApplicationController\src\META-INF`	Defines the available skins and also enables you to define new skins. See Skinning MAF Applications.
`adfm.xml`	*`application workspace directory`*`\ApplicationController\adfmsrc\META-INF` For example: `JDeveloper\mywork\application name\ApplicationController\adfmsrc\META-INF`	Maintains the paths (and relative paths) for the `.cpx`, `.dcx`, `.jpx`, and `.xcfg` files (registries of metadata).

Table C-2 (Cont.) Application Controller Artifacts

Artifact(s)	File Location	Description
`DataControls.dcx`	*`application workspace directory`* `\ApplicationController\adfmsrc\` For example: `JDeveloper\mywork\application name \ApplicationController\adfmsrc\`	The data controls registry. For information on using the DeviceFeatures data control, which leverages the services of the device, see 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 What You May Need to Know About Custom Springboard Application Features with MAF AMX Content.

C.3 About the View Controller Project Resources

The metadata files of the view controller project describe the resources at the application feature-level, and can be reused in other applications.

The view controller project (which is generated with the default name, `ViewController`) contains the resources for application features. Unlike the application controller project, the view controller 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. Furthermore, 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 MAF applications as described in Reusing MAF Application Content . 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. See What You May Need to Know About Feature Reference IDs and Feature IDs.

As shown in Table C-3, these resources include the configuration file for application features called `maf-feature.xml`.

Table C-3 View Controller Artifacts

Artifact(s)	File Location	Description
`maf-feature.xml`	*`application workspace directory`*`\src\META_INF\maf-feature.xml` For example: `JDeveloper\mywork\application name\ViewController\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 *Installing Oracle Mobile Application Framework*, you can deploy this application using the default deployment profile settings. See Deploying MAF Applications .
Application-Specific Content	*`application workspace directory`*`\ViewController\public_html` For example: `JDeveloper\mywork\application name\ViewController\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. See What You May Need to Know About Selecting External Resources .

C.4 About the MAF Application Configuration File

The `maf-application.xml` file not only specifies the basic configuration of a MAF application but also helps users create user preferences pages for the MAF application.

The `maf-application.xml` file specifies the basic configuration of the MAF application by designating its display name, a unique application ID (to prevent naming collisions) and selecting the application features that display on the springboard of the MAF application at runtime. Furthermore, the `maf-application.xml` file enables you to create the user preferences pages for the MAF application.

This file, which is generated by JDeveloper after you complete the application creation wizard as described in Creating a MAF Application, contains the elements listed in Table C-4.

Table C-4 Elements of the Application Descriptor File

Element	Description
`<adfmf:application>`	The root element of `maf-application.xml`.
`<adfmf:description>`	A description of the application.

Table C-4 (Cont.) Elements of the Application Descriptor File

Element	Description
`<adfmf:featureReference>`	A feature reference denotes which of the application features packaged in the FAR (Feature archive file) or defined in the `maf-feature.xml` file is relevant to the content of the MAF application. You define the character and content of MAF applications by selecting feature references. See Reusing MAF Application Content .
`<adfmf:preferences>`	Enables you to set the user preference options and behavior at the application level. You can also set how user preferences display and behave for the application features in the `maf-feature.xml` file. See Enabling User Preferences.
`<adfmf:login>`	Enables you to set the login page for an application feature. See Securing MAF Applications .
`<adfmf:navigation>`	Enables you to define the behavior of the navigation bar and the springboard. A springboard is a home page in which all of the application icons and labels for the embedded application features are organized in a List View. A springboard provides a top-level view of all of the applications available to a user, who can page through and select applications. See Configuring the Application Navigation.

C.5 About the MAF Application Feature Configuration File

The `maf-feature.xml` file contains configurations of the features in a MAF application.

The `maf-feature.xml` file configures the application features that the `<adfmf:featureReference>` elements in the MAF application's `maf-application.xml` file references. Example C-2 shows the People and Organization application features of the WorkBetter sample application in that application's `maf-feature.xml` file.

The `<adfmf:features>` root element in the `maf-feature.xml` file accepts one or more `<adfmf:feature>` elements where you define the properties of the application feature(s) in your MAF application. In Example C-2, values are provided for the properties that define the name and identify of the People and Organization application features in addition to the icons that render in the springboard and navigation bars for these application features in the WorkBetter sample application. Furthermore, the `<adfmf:feature>` elements reference the content that the application features render. The People and Organization application features reference task flows, `.CSS` and `.JS` files.

MAF applications implement security at the application feature level. One step in securing an application feature is to require that end users be authenticated before they can access the application feature. You do this by configuring the Enable Security

property (`securityEnabled`) for the application feature in the `maf-feature.xml` file. See Configuring Security for MAF Applications.

Table C-5 Child Elements of <Feature> Element

Element	Description
`<adfmf:content>`	Describes the format that the application feature uses for a particular device or user. The content (generally, the user interface) of an application feature can be written as MAF AMX pages, HTML5 pages, or be delivered from web pages hosted on a remote web server. For more information on designating content as a web application, see Implementing Application Feature Content Using Remote URLs .
`<adfmf:constraint>`	Determines whether a given application feature can be displayed in the application at runtime. Constraints can be used to allow or prevent the use of an application feature based on such criteria as user roles or device properties. See Setting Constraints on Application Features .

Example C-2 WorkBetter Sample Application's maf-feature.xml File

```
<?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="People" name="People" icon="images/people.png" image="images/people.png">
    <adfmf:content id="People.1">
      <adfmf:amx file="People/taskflow.xml#taskflow">
        <adfmf:includes>
          <adfmf:include type="StyleSheet" file="css/WorkBetter.css" id="i1"/>
          <adfmf:include type="JavaScript" file="js/customsearch.js" id="i2"/>
        </adfmf:includes>
      </adfmf:amx>
    </adfmf:content>
  </adfmf:feature>
  <adfmf:feature id="Organizations" name="Organizations" icon="images/departments.png"
                                                         image="images/departments.png">
      <adfmf:content id="Organizations.1">
        <adfmf:amx file="Organizations/taskflow.xml#taskflow">
          <adfmf:includes>
            <adfmf:include type="StyleSheet" file="css/WorkBetter.css" id="i3"/>
            <adfmf:include type="JavaScript" file="js/customsearch.js" id="i4"/>
          </adfmf:includes>
        </adfmf:amx>
      </adfmf:content>
  </adfmf:feature>
  ...
  </adfmf:features>
```

D

Converting Preferences for Deployment

This chapter describes how MAF converts user preferences during deployment.

This document includes the following sections:

- Naming Patterns for Preferences
- Converting Preferences for Android
- Converting Preferences for iOS
- 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`

Table D-1 (Cont.) MAF Naming Patterns for Preferences

Expression	Description	Syntax

`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.

The three documents are:

- maf_preferences.xml
- maf_arrays.xml
- maf_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 maf_preferences.xml file contains references to string resources contained in both the maf_strings.xml and maf_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 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 Naming Patterns for Preferences.
- Attributes with an asterisk (*) are custom template attributes defined in a MAF namespace and must appear in the `maf_preferences.xml` file in the form `adfmf:<attributeName>`. Otherwise, the attributes are part of the Android namespace and must appear in the maf_preferences.xml file 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 `maf_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 overview editors for the `maf-application.xml` and `maf-feature.xml` files 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 `maf_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 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 following example shows MAF preferences contained in the `maf-feature.xml` file.

```
<?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 maf_arrays.xml

The `maf_arrays.xml` file consists of string-array elements that enumerate the names and values of list preferences that are referenced from the `maf_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. The following example shows a MAF `preferenceList` definition.

```
<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: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 following example illustrates the pair of string array elements in the `maf_arrays.xml` file that are transformed from a `<preferenceList>` element. The

MAF `preferenceList` definition in the preceding example results in `<string-array name="feature.oracle.hello.prefGroup.MyList___entry_values">` and `<string-array name="feature.oracle.hello.prefGroup.MyList___entries">` in the `maf_arrays.xml` file shown in the following example.

```
<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 following example shows the `<string-arrays>` referenced in maf_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 maf_strings.xml

The `maf_strings`.xml file, shown in the following example, consists of string elements that are referenced by the `maf_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/jdev/adfmf">
   <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

MAF can convert the MAF Preference elements to the respective iOS counterparts.

The MAF deployment transforms the MAF preferences listed in Table D-4 to the preference list (`.plist`) 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/`).

The following example shows XML of MAF preferences 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.