

JD Edwards EnterpriseOne Tools

Developing and Customizing Mobile Enterprise Applications
Guide

Release 9.1.x

E56635-08

December 2015

Provides an overview of Oracle Mobile Application Framework (Oracle MAF) and describes the JD Edwards EnterpriseOne-specific tools and environment required to support the development of mobile enterprise applications for EnterpriseOne.

JD Edwards EnterpriseOne Tools Developing and Customizing Mobile Enterprise Applications Guide,
Release 9.1.x

E56635-08

Copyright © 2014, 2015, Oracle and/or its affiliates. All rights reserved.

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	ix
Audience.....	ix
Documentation Accessibility	ix
Related Documents	ix
Conventions	x
1 Understanding This Guide	
2 Introduction to Mobile Enterprise Application Development	
2.1 About the Runtime Architecture for EnterpriseOne Mobile Enterprise Applications	2-1
2.2 About Oracle Mobile Application Framework.....	2-2
2.3 Understanding Developing Custom Mobile Enterprise Applications for EnterpriseOne	2-2
2.3.1 JDE Mobile Helpers	2-3
2.3.2 The Data Model.....	2-4
2.3.3 Form Service Requests	2-4
2.3.4 EnterpriseOne Rest Services Interface	2-4
2.4 Sample Application	2-5
2.5 Mobile Application Archives	2-5
3 Getting Started	
3.1 Certifications (Formerly Known as Minimum Technical Requirements)	3-1
3.2 Prerequisites.....	3-1
3.3 Installing the AIS Client Class Generator Extension for JDeveloper	3-2
4 Setting Up the Login and Logout	
4.1 Before You Begin.....	4-1
4.2 Setting Up the Login Module.....	4-1
4.2.1 Pointing to the Login.jar	4-2
4.2.2 Making the Login Module the First Feature in Your Mobile Application	4-2
4.2.3 Verifying the LifecycleListenerImpl Activation.....	4-2
4.2.4 Setting the defaultFeature	4-2
4.2.5 Overriding the Login Values from Your Mobile Application.....	4-3
4.3 Configuring the Logout	4-3

5 Building the Data Model

5.1	Understanding the Data Model	5-1
5.2	Configuring the AIS Client Class Generator.....	5-1
5.3	Generating Data Classes Based on a Form.....	5-2

6 Performing AIS Form Service Calls

6.1	Understanding JD Edwards EnterpriseOne Mobile Framework APIs	6-1
6.2	Understanding AIS Server Capabilities.....	6-1
6.3	Understanding Form Service Requests.....	6-3
6.3.1	Overview	6-4
6.3.2	Events Used in a Form Service Request	6-4
6.3.3	Using the Form Service Request Event in FDA.....	6-5
6.3.4	Placing Events in the Proper Order	6-5
6.3.5	Considering Hidden Filters and Hidden QBE	6-5
6.3.6	Events on Power Forms	6-5
6.3.7	Control ID Notation for Return Control IDs	6-5
6.3.8	Example of JSON Code in a Form Service Request	6-6
6.3.9	Example of API Methods for a Form Service Request	6-10
6.3.10	Grid Action Events	6-11
6.3.10.1	Example of Grid Action Events	6-12
6.3.11	Query Events (Release 9.1 Update 5.2).....	6-31
6.3.11.1	Query Object Parameters.....	6-32
6.3.11.2	Query Object Examples	6-33
6.4	Understanding Batch Form Service.....	6-36
6.4.1	Batch Form Service - JSON Input and Output	6-36
6.4.2	Implementing the Batch Form Service.....	6-41
6.4.2.1	Batch Request Parent Class	6-41
6.4.2.2	Performing a Batch Form Request	6-41
6.5	Working with the EnterpriseOne REST Services Interface.....	6-43
6.5.1	Using a REST Services Client to Interact with AIS	6-44
6.5.1.1	Form Request Attributes	6-46
6.5.1.2	Calling FormService on Local EnterpriseOne HTML (JAS) Server through the AIS Server	6-48
6.6	Understanding Text Media Object Attachments.....	6-49
6.6.1	gettext Service.....	6-49
6.6.2	updatetext Service.....	6-50
6.6.3	JDEMobileFramework API Methods for Managing Text Media Objects.....	6-51
6.7	Understanding URL Media Object Attachments (Release 9.1 Update 5.2)	6-53
6.8	Understanding the Media Object API for Photo Media Object Attachments	6-55
6.8.1	List	6-55
6.8.2	Download.....	6-57
6.8.3	Upload.....	6-58
6.8.4	Delete	6-60
6.9	Understanding Processing Options	6-60
6.9.1	Using the AIS Service for Processing Options in Your Mobile Application.....	6-61
6.10	Understanding the Application Stack Service (Tools Release 9.1 Update 5)	6-63
6.10.1	Service Endpoint	6-63

6.10.2	Capability	6-63
6.10.3	Prerequisite	6-63
6.10.4	JSON Example of an Application Stack Request	6-63
6.10.4.1	Open Application: Request and Response.....	6-64
6.10.4.2	Execute Actions on Application: Request and Response	6-65
6.10.4.3	Adding a Phone Number	6-66
6.10.4.4	Execute Close Application: Request and Response.....	6-67
6.10.4.5	Mobile Application Example	6-68
6.10.5	ApplicationStack Methods	6-73
6.10.6	ApplicationStackResponse Methods	6-73

A Creating a Sample Mobile Application

A.1	Before You Begin.....	A-1
A.2	Creating the Sample Address Book Mobile Application	A-2
A.2.1	Creating a New Mobile MAF Application.....	A-2
A.2.2	Running the Mobile Application in the Simulator	A-5
A.3	Using the JDE Mobile Helpers	A-7
A.3.1	Including the JDEMobileFramework.jar	A-7
A.3.2	Including the Login.jar.....	A-8
A.3.3	Including the Javascript and CSS	A-10
A.3.4	Including the Resource Bundle.....	A-11
A.3.5	Including Logo Images	A-12
A.3.6	Enabling the Custom Springboard.....	A-14
A.3.7	Including the about.properties	A-15
A.3.8	Including an End User License Agreement (EULA)	A-16
A.4	Connecting to the EnterpriseOne Application Interface Services (AIS) Server.....	A-20
A.5	AIS Client Class Generator	A-21
A.6	Reading EnterpriseOne Data.....	A-21
A.7	Implementing Filter Fields	A-30
A.8	Page Navigation and Getting More Details	A-32
A.9	Updating Data in EnterpriseOne	A-36
A.9.1	Handling Errors	A-38
A.10	Device Integration.....	A-40

B Extending Mobile Application Archives

B.1	Before You Begin.....	B-1
B.2	Understanding Mobile Application Archives	B-2
B.3	Generating a New Application from the Deployment Profile	B-2
B.4	Customization Options	B-7
B.4.1	Customizing the Application Icons and Splash Screens.....	B-8
B.4.2	Customizing the Brand Images	B-10
B.4.3	Customizing the End User License Agreement (EULA).....	B-11
B.4.4	Customizing the About Page	B-11
B.4.5	Customizing the Pages.....	B-12
B.5	Extension Options.....	B-12
B.5.1	Displaying Additional Data	B-12

B.5.2	Removing Data from Pages.....	B-19
B.5.3	Adding New Pages.....	B-19

List of Examples

6-1	Requesting fields and grid columns on a traditional form.....	6-6
6-2	Requesting main form fields, subform fields, main form grid columns, and subform grid columns. 6-6	
6-3	Executing EnterpriseOne Actions - JSON	6-6
6-4	Executing EnterpriseOne Actions - JSON Response.....	6-7
6-5	API Methods for Setting Commands	6-10
6-6	Selecting Grid Rows - JSON	6-12
6-7	Selecting Grid Rows - JSON Response	6-13
6-8	Selecting Grid Rows - MAF Application Code.....	6-16
6-9	Insert Rows - JSON	6-17
6-10	Insert Rows - JSON Response	6-18
6-11	Insert Rows - MAF Application Code.....	6-22
6-12	Update Rows - JSON	6-24
6-13	Update Rows - JSON Response	6-26
6-14	Update Rows - MAF Application Code.....	6-30
6-15	Example of Control ID and Business View Information Displayed under Advanced Options in the EnterpriseOne Web Client Item Help 6-32	
6-16	Query - JSON	6-33
6-17	Query - Mobile Framework API	6-35
6-18	JSON Input in a Batch Form Service	6-36
6-19	JSON Output in a Batch Form Service	6-37
6-20	Batch Request Parent Class.....	6-41
6-21	Batch Form Request	6-42
6-22	Acceptable Input for the defaultconfig Service on JSON.....	6-44
6-23	Acceptable Input for the tokenrequest Service in JSON	6-45
6-24	Acceptable Input for the tokenrequest Service in JSON Response.....	6-46
6-25	Form Request.....	6-47
6-26	gettext Service Input	6-49
6-27	gettext Service Response	6-50
6-28	Update Text Service Input	6-50
6-29	Update Text Service Response	6-51
6-30	Get Example.....	6-51
6-31	Append Example	6-52
6-32	Update Example.....	6-52
6-33	URL Media Object - JSON Request and Response.....	6-54
6-34	URL Media Object - Mobile Framework API	6-54
6-35	Open Application - Request	6-64
6-36	Open Application - Response.....	6-64
6-37	Execute Actions on Application - Request	6-65
6-38	Execute Actions on Application - Response	6-65
6-39	Adding a Phone Number - Request	6-66
6-40	Execute Close Application - Request	6-67
6-41	Execute Close Application - Response.....	6-67
A-1	Using the Verbatim Tag to Include HTML	A-17
B-1	Adding Additional Data to the Supplier Search Mobile Application.....	B-12

Preface

Welcome to the *JD Edwards EnterpriseOne Tools Developing and Customizing Mobile Enterprise Applications Guide*.

Note: This guide has been updated for JD Edwards EnterpriseOne Tools Release 9.1 Update 5. For details on documentation updates, refer to the *JD Edwards EnterpriseOne Tools Net Change for Tools Documentation Library*.

Audience

This guide is intended for application developers who are responsible for creating or customizing EnterpriseOne mobile enterprise applications.

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 about Oracle MAF and EnterpriseOne mobile enterprise applications, see the following documents:

- *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework* documentation:
<http://docs.oracle.com/middleware/maf210/mobile/index.html>
- *JD Edwards EnterpriseOne Application Interface Services Server for Mobile Enterprise Applications Configuration Guide*
- *JD Edwards EnterpriseOne Applications Mobile Enterprise Applications Implementation Guide*

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.
<i>italic</i>	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.

Understanding This Guide

Use this guide as a companion guide to the *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework Guide*, which describes how to develop mobile applications using Oracle Mobile Application Framework (MAF). Oracle MAF is the framework used to create mobile enterprise applications for JD Edwards EnterpriseOne. You can access the Oracle MAF guide here:

<http://docs.oracle.com/middleware/maf210/mobile/develop-maf/toc.htm>

The *JD Edwards EnterpriseOne Tools Developing and Customizing Mobile Enterprise Applications Guide* includes:

- An overview of the environment required for developing EnterpriseOne mobile enterprise applications, which includes Oracle MAF and JD Edwards EnterpriseOne-specific extensions and tools, referred to as JDE Mobile Helpers.
See [Chapter 2, "Introduction to Mobile Enterprise Application Development"](#) for more information.
- How to use the JDE Mobile Helpers with Oracle MAF to create custom EnterpriseOne mobile enterprise applications.
See the following chapters for more information:
 - [Chapter 3, "Getting Started"](#)
 - [Chapter 4, "Setting Up the Login and Logout"](#)
 - [Chapter 5, "Building the Data Model"](#)
 - [Chapter 6, "Performing AIS Form Service Calls"](#)
- Step-by-step instructions on how to create a sample application, including how to add features with JDE Mobile Helpers.
See [Appendix A, "Creating a Sample Mobile Application."](#)
- How to use EnterpriseOne mobile application archives (MAAs) to generate mobile application binaries for internal distribution.
See [Appendix B, "Extending Mobile Application Archives."](#)

This guide also references JD Edwards EnterpriseOne Mobile Framework Java APIs for developing custom mobile enterprise applications. Descriptions of these APIs can be found in a Javadoc entitled *JD Edwards EnterpriseOne Mobile Framework Java API Reference*.



Introduction to Mobile Enterprise Application Development

This chapter provides an overview of Oracle Mobile Application Framework (Oracle MAF) and the JD Edwards EnterpriseOne-specific tools and environment required to support the development of mobile enterprise applications (also simply referred to as mobile applications) for EnterpriseOne.

This chapter contains the following topics:

- Section 2.1, "About the Runtime Architecture for EnterpriseOne Mobile Enterprise Applications"
- Section 2.2, "About Oracle Mobile Application Framework"
- Section 2.3, "Understanding Developing Custom Mobile Enterprise Applications for EnterpriseOne"
- Section 2.4, "Sample Application"
- Section 2.5, "Mobile Application Archives"

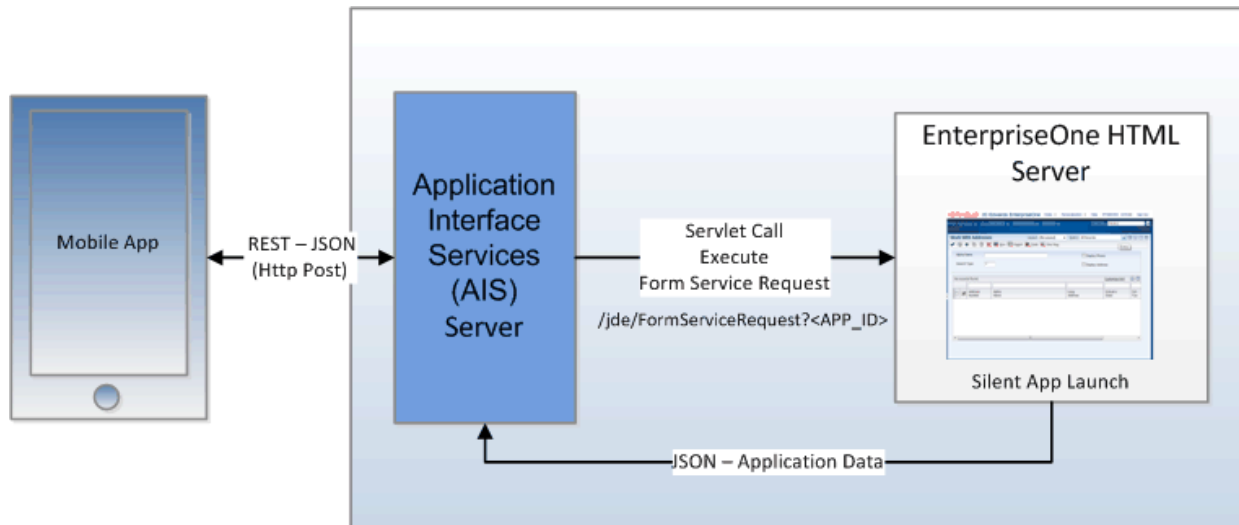
2.1 About the Runtime Architecture for EnterpriseOne Mobile Enterprise Applications

EnterpriseOne mobile enterprise applications require a light interface to manage EnterpriseOne data from mobile devices. The Application Interface Services (AIS) Server provides a JSON over REST interface to EnterpriseOne applications and forms through the EnterpriseOne HTML Server. The AIS Server exposes this interface to enable communication between mobile applications and EnterpriseOne.

The AIS Server includes support for JSON representation of Form Service Requests so mobile applications can easily format requests. The AIS Server submits these mobile application requests to the EnterpriseOne HTML Server.

The AIS Server maintains sessions for mobile applications. You can configure the session timeouts for the AIS Server through Server Manager.

The following illustration shows how the AIS Server functions as the interface between mobile applications and the EnterpriseOne HTML Server.

Figure 2–1 Runtime Architecture for Mobile Enterprise Applications

2.2 About Oracle Mobile Application Framework

Oracle Mobile Application Framework (MAF) is a solution that enables you to create mobile applications that run natively on both iOS and Android phones and tablets. EnterpriseOne mobile enterprise applications are built using Oracle MAF.

You should gain a thorough understanding of Oracle MAF before reading further. See the *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework Guide*, which you can access here:

<http://docs.oracle.com/middleware/maf210/mobile/develop-maf/toc.htm>

2.3 Understanding Developing Custom Mobile Enterprise Applications for EnterpriseOne

While Oracle provides many out-of-the-box EnterpriseOne mobile enterprise applications, the applications may not meet your specific business requirements. Therefore, Oracle provides a mobile application framework with tools that enable you to develop custom mobile applications for EnterpriseOne.

Also, if you have customized EnterpriseOne applications to meet your specific business requirements, you can extend the functionality of the mobile applications to interact with the customized EnterpriseOne applications. The mobile application framework enables you to extend data from custom EnterpriseOne applications to custom mobile applications by using existing business logic within your EnterpriseOne applications.

Before you develop custom mobile applications, you should plan all aspects of the process. The "About Developing Applications with Oracle Mobile Application Framework" section in the *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework Guide* provides a high-level description of activities that you should perform when building an MAF application. These activities include gathering requirements, designing, developing, deploying, testing and debugging, securing, enabling access to the server-side data, redeploying, retesting and debugging, and publishing. When developing custom EnterpriseOne mobile applications, pay special attention to the following activities:

- **Designing.** Determine which forms in EnterpriseOne the mobile application will access data from. If there are multiple forms, consider creating a batch request to gather data from multiple forms.
- **Developing.** Use the JD Edwards EnterpriseOne Mobile Framework APIs, Login Module, and JDE Mobile Helpers to help expedite the development of your mobile application. Refer to [Appendix A, "Creating a Sample Mobile Application"](#) in this guide for an example of how to develop a custom mobile application for EnterpriseOne using these utilities.
- **Securing.** The Login Module handles the authentication of EnterpriseOne mobile enterprise application users.
- **Publishing.** This is an important aspect of your deployment plan, as it typically involves publishing to an enterprise server, Apple App Store, or Google Play.

2.3.1 JDE Mobile Helpers

Oracle provides additional tools referred to as JDE Mobile Helpers that help simplify the development of mobile enterprise applications. JDE Mobile Helpers include:

- **JDEMobileFramework.jar**

This JAR file contains the JD Edwards EnterpriseOne Mobile Framework APIs, a set of classes and API methods that enable the mobile application to manage (create, read, update, delete) data in EnterpriseOne through REST services.
- **AIS Client Class Generator**

The AIS Client Class Generator is a JDeveloper extension that enables you to generate Application Controller foundational classes that are required by EnterpriseOne mobile applications.
- **Login.jar**

The Login.jar provides a configuration page, login page, and a springboard. The springboard contains links to Legal Terms or the End User License Agreement (EULA), About information, and the Logout.
- **about.properties**

This file is for configuring information displayed on the About page, including the application name, application version, and the application ID (which is part of enabling the application with EnterpriseOne application security). If you enable the springboard, you should provide these values so that they will appear on the about page.
- **Resource Bundle**

The Resource Bundle contains text resources for the pages provided in the Login.jar.
- **Javascript (JS) files and CSS files**

The Javascript and CSS files are dependencies of the JDEMobileFramework.jar and the Login.jar. The Javascript provides an animated icon to show that the mobile application is processing while service calls are made. The CSS provides an extension to the styling skin provided by Oracle MAF. It enables you to make adjustments to the style of the configuration, login, and springboard pages of your mobile application.

2.3.2 The Data Model

The data model for developing mobile enterprise applications includes foundation classes that hold the data retrieved from AIS services. These classes are specific to the EnterpriseOne applications accessed by a mobile enterprise application. In JDeveloper, you can include the classes in the Application Controller or the View Controller.

You use the AIS Client Class Generator to generate classes for the Application Controller and View Controller. To generate data classes for a form, in the AIS Client Class Generator, you define the service request information by identifying the form and the CRUD (create, read, update, delete) operation that you want the application form to perform.

2.3.3 Form Service Requests

AIS Server calls are used to retrieve data from forms in the EnterpriseOne web client. These calls are referred to as form service requests. Mobile applications use form service requests to interact with EnterpriseOne web client forms. Form service requests, formatted as REST service calls that use POST, contain form service events or commands that invoke actions on an EnterpriseOne form.

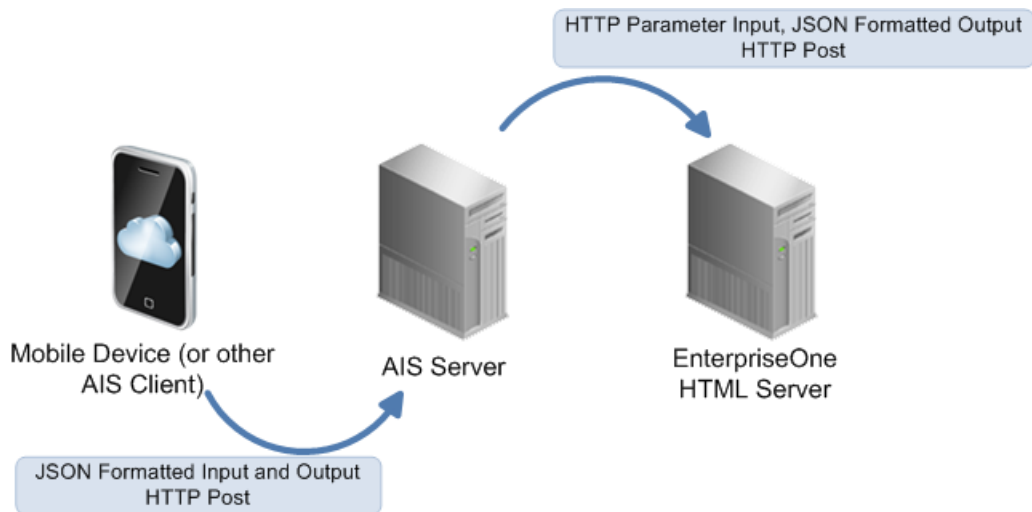
By sending an ordered list of commands, a form service request can replicate the actions taken by an EnterpriseOne web client user, including populating fields, pressing buttons, and other actions. The form service request enables you to perform various operations on a single form. The URL for the form service request is:

```
http://<aisserver>:<port>/jderest/formservice
```

2.3.4 EnterpriseOne Rest Services Interface

The following illustration shows JSON input and output over HTTP Post:

Figure 2–2 AIS Client and Server Communication



This illustration shows the communication from the client (which can be a mobile device or other AIS client) to the AIS Server and the AIS Server's communication with the EnterpriseOne HTML server, where the EnterpriseOne forms run and the data is gathered. All client communication uses JSON formatted strings.

You can make REST calls directly to the AIS Server without using the JDE Mobile Framework APIs. Use a REST service testing tool to call AIS services directly by

sending JSON text to the URL for the service. All services are accessed using URLs with this format: `http://<aisserver>:port/jderest/<uri>`, where `uri` is the path to the various types of services such as `formservice`, `file`, `defaultconfig`, and `poservice`. For a list of all services that are available on the AIS Server, see [Working with the EnterpriseOne REST Services Interface](#) in this guide.

2.4 Sample Application

This guide includes an appendix that provides step-by-step instructions on how to create a sample EnterpriseOne mobile application. As you read about the JDE Mobile Helpers and other features in this guide, you can refer to the steps in this appendix to see an example of how the features are used in the development of a mobile application.

2.5 Mobile Application Archives

Oracle provides mobile applications archives, which you can use to create your own iOS and Android artifacts. This enables you to integrate and control mobile applications internally within your company rather than connecting to a public application store.

You can create your own iOS or Android artifact by generating the mobile application from a mobile application archive (MAA), customizing it, and then publishing it as your own "new" mobile application.

Getting Started

This chapter refers to certifications (MTRs) and prerequisites for developing and running EnterpriseOne mobile enterprise applications.

This chapter contains the following topics:

- Section 3.1, "Certifications (Formerly Known as Minimum Technical Requirements)"
- Section 3.2, "Prerequisites"
- Section 3.3, "Installing the AIS Client Class Generator Extension for JDeveloper"

3.1 Certifications (Formerly Known as Minimum Technical Requirements)

Customers must conform to the supported platforms for the release, which can be found in the Certifications tab on My Oracle Support: <https://support.oracle.com>.

For more information about JD Edwards EnterpriseOne Minimum Technical Requirements, see the following document on My Oracle Support: JD Edwards EnterpriseOne Minimum Technical Requirements Reference (Doc ID 745831.1), which is available here:

<https://support.oracle.com/epmos/faces/DocumentDisplay?id=745831.1>

3.2 Prerequisites

To develop custom mobile applications, you must complete the following prerequisites:

- You must be running a minimum of EnterpriseOne Tools release 9.1.4.6 with an EnterpriseOne AIS Server configuration. The AIS Server enables communication between mobile enterprise applications and EnterpriseOne.

See the *JD Edwards EnterpriseOne Application Interface Services Server for Mobile Enterprise Applications Configuration Guide*.

- Download the `JDE_Mobile_Framework_2.1.0` package from the JD Edwards Update Center on My Oracle Support (<https://support.oracle.com/>). In the Update Center, enter "EnterpriseOne Mobile Enterprise Applications" in the Type field to locate the package.

This package contains the `JDE_MobileFramework_2.1.zip` file, which contains the following files:

- `JDEMobileFramework.jar` (JD Edwards EnterpriseOne Mobile Framework APIs)

- jdemfResourceBundle.xlf (resource bundle)
- Login.jar (Login Module)
- Javascript and CSS files.
- about.properties
- AISCGE 12c_v1.6.1.zip (AIS Client Class Generator extension for JDeveloper)

The package also contains the following files:

- maf-2.1100.20150325-1239-RELEASE.zip

This contains the Oracle MAF extension for JDeveloper, which is required to develop and customize mobile enterprise applications. In JDeveloper, you can install this file by selecting the "Install from Local File" option in the Check for Updates Wizard.

- JDEMobileSampleApplication.zip

This is an **optional** file that contains the components for running a sample mobile application. [Appendix A, "Creating a Sample Mobile Application"](#) in this guide describes how to create this sample mobile application. You can use this sample mobile application download for comparison purposes.

Important: The Update Center also contains the previous version of this package, *JDE_Mobile_Framework_2.0.1*. If you previously installed this version and plan on using the sample mobile application or mobile application archives, see the "Before You Begin" in the following appendixes for important component compatibility information:

[Appendix A, "Creating a Sample Mobile Application"](#)

[Appendix B, "Extending Mobile Application Archives"](#)

- Complete the prerequisites listed in the Oracle MAF guide, which include:
 - Oracle JDeveloper 12.1.3
 - Oracle JDeveloper extension for Oracle MAF. **Note:** This extension is included in the *JDE_Mobile_FrameWork* package as well.
 - Prerequisites specific to the iOS platform and Android platform

See "Setting Up Development Tools for iOS Platform" in the *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework Guide*.

3.3 Installing the AIS Client Class Generator Extension for JDeveloper

The AIS Client Class Generator extension for JDeveloper contains the AIS Client Class Generator, a tool that supports the creation of Application Controller foundational classes that are required by EnterpriseOne mobile applications.

For more information about the AIS Client Class Generator, see [Chapter 5, "Building the Data Model"](#) in this guide.

To install the extension:

1. In JDeveloper, select the **Help** menu, **Check for Updates**.
2. Click **Next**.

3. Select **Install From Local File**, and then enter the location of the maf-2.1100.20150325-1239-RELEASE.zip file. This file is part of the JDE_Mobile_Framework download package.
4. Click **Next**, and then click **Finish**.
JDeveloper closes automatically.

Setting Up the Login and Logout

This chapter describes how to use the Login Module to manually configure the user login and logout for a mobile application.

This chapter contains the following topics:

- [Section 4.1, "Before You Begin"](#)
- [Section 4.2, "Setting Up the Login Module"](#)
- [Section 4.3, "Configuring the Logout"](#)

4.1 Before You Begin

Before you can set up the Login Module for your mobile application, you have to:

- Create the mobile application in JDeveloper.
The instructions on how to create a sample application include the steps for creating a new mobile application. See [Section A.2.1, "Creating a New Mobile MAF Application."](#)
- Configure the JDEMobileFramework.jar for the mobile application.
The sample application instructions include the JDEMobileFramework.jar configuration steps as well. See [Section A.3.1, "Including the JDEMobileFramework.jar."](#)

4.2 Setting Up the Login Module

The Login Module is provided in the Login.jar file.

Important: An Oracle MAF license is required in order to use the Login.jar (Login Module) in production-ready mobile applications.

Setting up the Login Module includes the following tasks:

- [Pointing to the Login.jar](#)
- [Making the Login Module the First Feature in Your Mobile Application](#)
- [Verifying the LifecycleListenerImpl Activation](#)
- [Setting the defaultFeature](#)
- [Overriding the Login Values from Your Mobile Application](#)

4.2.1 Pointing to the Login.jar

To point to the Login.jar:

1. In JDeveloper, click the **Application** drop-down and select **Application Properties**.
2. Select **Libraries and Classpath** and then click the **Add JAR/Directory** button to locate the Login.jar file on your local file system.

The Login.jar is part of the JDE_Mobile_Framework_1.0.0 download.

3. After you locate the Login.jar, click **Open**.
4. Click **OK** to save the properties.

4.2.2 Making the Login Module the First Feature in Your Mobile Application

To make the Login Module the first feature in your mobile application:

1. In the Application Resources panel, double-click the **maf-application.xml** to open it.
2. Select the **Feature References** tab, and then click the green plus sign to add a feature reference.
3. In the Id drop-down menu, select **com.oracle.e1.jdemf.login** and click **OK**.
4. Use the blue arrow to the right to move the login feature to the top of the list.

4.2.3 Verifying the LifecycleListenerImpl Activation

You must verify that the LifecycleListenerImpl is activated before setting the defaultFeature or overriding your login values.

To do so:

1. In the Application Resources panel, under **Descriptors > ADF META-INF**, double click **maf-application.xml**.
2. Select the **Application** tab and then click the magnifying glass next to **Lifecycle Event Listener**.
3. Navigate to LifecycleListenerImpl in the hierarchy, which should be:
`application.LifecycleListenerImpl`
4. Click **OK**.

4.2.4 Setting the defaultFeature

You must set the default feature in your application so that the Login page can successfully navigate to your default page.

To do so:

1. In the Projects panel, expand **ApplicationController > Application Sources > application**, and double-click **LifeCycleListenrImpl.java**.
2. Locate a method that looks like the following method:

```
public void start()
{
    // Add code here...
}
```


3. Replace the code in the method with the following code, where the value in quotes is the Feature ID of your applications main feature:

```
public void start()
{
    LoginConfiguration.setDefaultFeature("myfeatureId");
}
```

4.2.5 Overriding the Login Values from Your Mobile Application

While developing your mobile application, you typically run the mobile application multiple times for testing. To make the testing process easier, you can override the login values.

To do so:

1. On the LifecycleListenerImpl.java page, locate a method that looks like the following method:

```
public void start()
{
    // Add code here...
}
```

2. Replace the code in the method with the following code, where the value in quotes is the Feature ID of your applications main feature:

```
public void start()
{
    LoginConfiguration.setCredentials("jde", "jde");
}
```

This code will override the user name and password on the login screen. Depending on the values that you need to override, use one of the following parameters to invoke this method:

- `LoginConfiguration.setCredentials(username, password);`
- `LoginConfiguration.setCredentials(username, password, environment, role);`
- `LoginConfiguration.setCredentials(username, password, environment, role, jasserver);`

4.3 Configuring the Logout

This section describes how to manually configure the logout if you are not using the springboard for the logout. The steps on how to use the springboard are located in the section that describes how to create a sample mobile application. See [Appendix A, "Creating a Sample Mobile Application"](#) for more information.

To configure the mobile application logout, you call the `JDEmfUtilities.logout()` method. To enable your mobile application to call it, place this call in a method of your DC class. This method returns the user to the login screen. When the default feature is invoked after logging in again, it will be in a new state.

```
public static void logout(){
    JDEmfUtilities.logout();
}
```

To place a Logout button on your screen, regenerate your data control from your DC class. You can now drag the item `logout()` onto your form.

Building the Data Model

This chapter contains the following topics:

- Section 5.1, "Understanding the Data Model"
- Section 5.2, "Configuring the AIS Client Class Generator"
- Section 5.3, "Generating Data Classes Based on a Form"

5.1 Understanding the Data Model

The data model for developing a mobile enterprise application includes foundation classes that hold the data retrieved from AIS services. These classes are specific to the EnterpriseOne applications accessed by a mobile enterprise application. You can include the classes in the Application Controller or the View Controller.

You use the AIS Client Class Generator to generate classes for the Application Controller and View Controller.

5.2 Configuring the AIS Client Class Generator

The AIS Client Class Generator is available as a JDeveloper extension. Before you configure it, you must install the extension. See [Chapter 3, "Getting Started"](#) in this guide for instructions on how to install the extension.

To configure the AIS Client Class Generator:

1. In JDeveloper, access Preferences:
 - On Microsoft Windows, select the **Tools** menu, **Preferences**.
 - On Mac, select the **JDeveloper** menu, **Preferences**.
2. Select **AIS Client Class Generator**.
3. On Preferences, complete the following fields to specify the AIS Server location and AIS Server information:
 - **AIS Server URL**. This is a fully qualified URL to the AIS Server. Make sure it ends with the port number of the AIS Server.
 - **JAS Server URL**. (Optional) Only enter a value if you want to override the value configured on the AIS Server.
 - **Username**. Enter a JD Edwards EnterpriseOne user name.
 - **Password**. Enter a JD Edwards EnterpriseOne user password.

- **Environment.** (Optional) Enter a value only if you want to override the value configured on the AIS Server.
 - **Role.** This is optional. Enter a value only if you want to override the value configured on the AIS Server.
 - **JSON Files Folder.**
 - **Default Java Classes Folder.** The folder for storing generated Java files.
The AIS Client Class Generator uses this folder only when it is run without a project open in JDeveloper. When a project is open in JDeveloper, the generator stores the Java files in the source directory for the project at the defined package path or the default package path which is `com.oracle.e1.formservicetypes`.
 - **Java Package.** The Java package name for the generated classes. The default is `com.oracle.e1.formservicetypes`.
4. Click OK.

5.3 Generating Data Classes Based on a Form

Use the AIS Client Class Generator to generate data classes for an EnterpriseOne form. In the AIS Client Class Generator, you supply the service request information.

Note: The AIS Client Class Generator supports form interconnects only; it does not support form events.

To use the AIS Client Class Generator to generate data classes:

1. In JDeveloper, in the mobile application that you created, select the ApplicationController project.
JDeveloper will save the classes generated by the AIS Client Class Generator in this location.
2. Select the **Tools** menu, **AIS Client Class Generator**.
3. On AIS Client Class Generator, complete the following fields to supply the service request information:
 - **Username.** This contains the default value entered in the preferences.
 - **Password.** This contains the default value entered in the preferences.
 - **Environment.** This contains the default value entered in the preferences.
 - **Role.** This contains the default value entered in the preferences.
 - **Application Name.** Enter the name of the EnterpriseOne application.
 - **Form Name.** Enter the name of the EnterpriseOne application form.
 - **Version.** (Optional) Enter the version name. If you leave it blank, the generator will use ZJDE0001 by default.
 - **MaxPageSize.** (Optional)
 - **ReturnControlIDs.** (Optional) Use this field to specify the exact fields on the form that you want generated. The return control IDs can specify hidden fields or a subset of fields.
 - **FormInputs.** (Optional)

- **FormServiceAction.** Enter the action to be performed. Valid values include: Create, Read, Update, Delete.
 - **FindOnEntry.** (Optional)
 - **DemoMode.** (Optional, but recommended) This ensures at least one grid row is present, so grid classes are generated even if there is no data in the database.
4. Make sure to select the **Preview JSON Data** and **Keep JSON Files** check boxes if you want to preview and keep the JSON files.
 5. Click the **Generate** button to generate the JSON, and then verify that it has the fields and records you need.
 6. Continue to generate the Java files. If successful, a confirmation message appears that shows the location of the JSON files.
 7. Highlight the Application Controller project and then click the "**refresh**" button to display the new files.

The AIS Client Class Generator displays a dialog box that shows where the classes are saved.

Performing AIS Form Service Calls

This chapter contains the following topics:

- Section 6.1, "Understanding JD Edwards EnterpriseOne Mobile Framework APIs"
- Section 6.2, "Understanding AIS Server Capabilities"
- Section 6.3, "Understanding Form Service Requests"
- Section 6.4, "Understanding Batch Form Service"
- Section 6.5, "Working with the EnterpriseOne REST Services Interface"
- Section 6.6, "Understanding Text Media Object Attachments"
- Section 6.7, "Understanding URL Media Object Attachments (Release 9.1 Update 5.2)"
- Section 6.8, "Understanding the Media Object API for Photo Media Object Attachments"
- Section 6.9, "Understanding Processing Options"
- Section 6.10, "Understanding the Application Stack Service (Tools Release 9.1 Update 5)"

6.1 Understanding JD Edwards EnterpriseOne Mobile Framework APIs

The JD Edwards EnterpriseOne Mobile Framework APIs provide Java classes and methods that enable users to call the REST services on the AIS Server. There are APIs that handle login authentication and authorization with the AIS Server, APIs for form service requests and media objects, as well as APIs for retrieving processing option values.

The JD Edwards EnterpriseOne Mobile Framework APIs have a dependency on Oracle MAF. They do not function outside of an Oracle MAF environment.

See the [Prerequisites](#) section in this guide for information on how to obtain the APIs and the *JD Edwards EnterpriseOne Mobile Framework Java API Reference* Javadoc.

6.2 Understanding AIS Server Capabilities

The AIS Server exposes various capabilities that client applications may or may not depend on. If your mobile application requires a certain capability, you must include it in the list of required capabilities in the `about.properties` file.

If you indicated in the `about.properties` file that you have a required capability, the Login module verifies that capability is available when the mobile application

launches. If the capability is not available, the application returns an error message. If the capability is available, the application continues to the login screen.

You can access the AIS Server capabilities using the following URL:

`http://<AIS Server>:<Port>/jderest/defaultconfig`

The following code shows the available capabilities along with a description of each capability:

```

"capabilityList": [
  {
    "name": "grid",
    "shortDescription": "Grid Actions",
    "longDescription": "Ability to update, insert and delete grid
records.",
    "asOfRelease": "9.1.4.4"
  },
  {
    "name": "editable",
    "shortDescription": "Enabled/Disabled",
    "longDescription": "Ability to indicate if form field or grid cell is
editable (enabled) or not (disabled).",
    "asOfRelease": "9.1.4.4"
  },
  {
    "name": "log",
    "shortDescription": "Logging",
    "longDescription": "Endpoint exposed for logging to AIS server log
from client",
    "asOfRelease": "9.1.4.6"
  },
  {
    "name": "processingOption",
    "shortDescription": "Processing Options",
    "longDescription": "Processing Option Service exposed for fetching PO
values from E1",
    "asOfRelease": "9.1.4.6"
  },
  {
    "name": "ignoreFDAFindOnEntry",
    "shortDescription": "Ignore FDA Find On Entry",
    "longDescription": "Ability to use the IgnoreFDAFindOnEntry flag",
    "asOfRelease": "9.1.4.6"
  },
  {
    "name": "selectAllGridRows",
    "shortDescription": "Select or Unselect All Grid Rows",
    "longDescription": "Ability to use select and unselect all grid rows,
or unselect a single row in an action event.",
    "asOfRelease": "9.1.5"
  },
  {
    "name": "applicationStack",
    "shortDescription": "Operations on a Stack of E1 Applications",
    "longDescription": "Ability to maintain a stack of open E1
applications and operate forms that are called",
    "asOfRelease": "9.1.5"
  },
  {
    "name": "thumbnailSize",
    "shortDescription": "Specify desired thumbnail size for MO List",

```

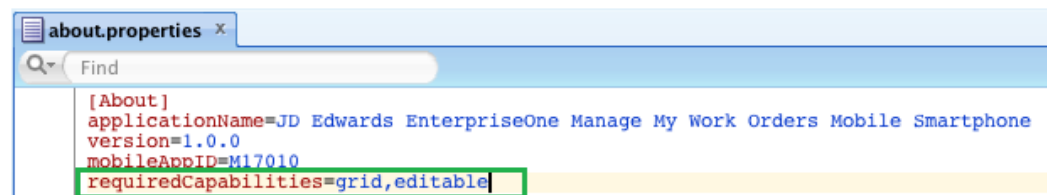


```

        "longDescription": "Ability to request a specific sized thumbnail
images in a Media Object List Request",
        "asOfRelease": "9.1.5"
    },
    {
        "name": "gridCellClick",
        "shortDescription": "Click Grid Cell Hyperlink",
        "longDescription": "Ability to use GridCellClick event, to execute
hyperlink in grid.",
        "asOfRelease": "9.1.5.2"
    },
    {
        "name": "query",
        "shortDescription": "Query",
        "longDescription": "Ability to use Query on forms that support it",
        "asOfRelease": "9.1.5.2"
    },
    {
        "name": "urlMediaObjects",
        "shortDescription": "URL Media Objects",
        "longDescription": "Ability to view, add or delete url type media
objects",
        "asOfRelease": "9.1.5.2"
    }
],

```

The following example shows the grid and editable capabilities listed in the `about.properties`:



6.3 Understanding Form Service Requests

This section contains the following topics:

- Overview
- Events Used in a Form Service Request
- Using the Form Service Request Event in FDA
- Placing Events in the Proper Order
- Considering Hidden Filters and Hidden QBE
- Events on Power Forms
- Example of JSON Code in a Form Service Request
- Example of API Methods for a Form Service Request
- Grid Action Events
- Query Events (Release 9.1 Update 5.2)

6.3.1 Overview

AIS Server calls that retrieve data from forms in the EnterpriseOne web client are referred to as form service requests. Mobile applications use form service requests to interact with EnterpriseOne web client forms. Form service requests, formatted as REST service calls that use POST, contain form service events or commands that invoke actions on an EnterpriseOne form.

A form service request enables you to perform various operations on a single form. By sending an ordered list of commands, a form service request can replicate the actions taken by an EnterpriseOne web client user, including populating fields, pressing buttons, and other actions.

To send a form service request to the AIS Server, send a POST to the following URL and send JSON in the body:

```
http://<AIS Server>:<Port>/formservice
```

If testing with a REST testing tool, you can send JSON directly. If using the JDE Mobile Helpers in a mobile application, you need to specify only the URI when calling the `jdeRestServiceCall`. The URI is "formservice" and you can use the static variable, for example:

```
JDERestServiceProvider.FORM_SERVICE_URI
```

6.3.2 Events Used in a Form Service Request

The following table lists the events that you can include in a form service request and describes the action that each event performs.

Event	Description	Parameters (and example values)
Set Control value	Sets the value of a control on a form, like filter fields or any other form control.	controlID ("25") value ("Bob" or "01/01/2015")
Set QBE Value	Sets the value of a QBE column.	controlID ("1[42]" or "1_2[25]") value ("Jill" or "55")
Set Checkbox Value	Sets the value of a check box.	controlID ("77") value ("on" or "off")
Set Radio Button	Sets the value of the radio button.	controlID ("87") value ("87")
Set Combo Value	Sets the value of a combo box entry.	controlID ("125") value (2) - Index of the entry.
Do Action	Presses a button or Hyper Item.	controlID ("156")
Select Row	Selects the specified row in a grid.	controlID ("1.30") - ID of the grid, dot (.), row index (zero based).
Select All Rows (Release 9.1 Update 5)*	Select all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - ID of the grid.
Un Select All Rows (Release 9.1 Update 5)*	Deselects all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - ID of the grid.
Un Select Row (Release 9.1 Update 5)*	Deselects the specified row in a grid.	controlID ("1.30") - ID of the grid, dot (.), row index (zero based).
Click Grid Cell** (Release 9.1 Update 5.2)	Clicks the hyperlink in a grid cell if the cell is enabled as a link.	controlID ("1.5.22") - ID of the grid, dot (.), row index, dot (.), grid column ID.

*These events are available only with the selectAllGridRows capability which is available starting with EnterpriseOne Tools 9.1 Update 5. The JDEMobileFramework API requires these events to be in a try block because they throw a Capability Exception.

**This event is available only with the gridCellClick capability, which is available starting with EnterpriseOne Tools 9.1 Update 5.2.

6.3.3 Using the Form Service Request Event in FDA

Starting with EnterpriseOne Tools release 9.1.4, the Form Service Request event is available within FDA for each form. This event occurs after the Post Dialog Initialized event, but before any of the form actions requested in the form service call execute. This event enables you to perform some operations, or business logic, when you know the form that is being called from a form service request.

This event also provides access to the requested form service actions, referred to as CRUD (Create, Read, Update, or Delete) actions, by using the "Get Form Service Request Action" system function. This enables you to create additional logic based on the value sent in the form service request.

Using the Form Service Request event in FDA should be secondary to using events (actions) provided in the Form Service Request from the mobile application. Oracle recommends that you only use the FDA event if you cannot accomplish a desired result with the form action events.

6.3.4 Placing Events in the Proper Order

Place the events in the form service request in the order you want them to execute, for example, populate a filter field value and then press the Find button. Remember that the FDA Form Service Request event occurs before the events you add to this list. Do not set the "Find On Entry" option when using the event model; the extra "find" is not necessary because it executes before the events you requested.

6.3.5 Considering Hidden Filters and Hidden QBE

By default, values are not written to hidden filter fields or hidden QBE columns. You must use the Form Service Request event to show the fields and columns first. Then values can be written to these fields and subsequently applied to the query.

6.3.6 Events on Power Forms

The Form Service Request event and other events that perform actions in a form are also available on power forms so that you can populate a subform value or press a button on a subform.

6.3.7 Control ID Notation for Return Control IDs

You can use the Property Browser in FDA to identify control IDs for fields on each EnterpriseOne form. You can also find control IDs using the Item Help option in the form in the EnterpriseOne web client. In the EnterpriseOne web client form, click the Help button (question mark in the upper right corner of a form) and then click the Item Help option to access field-level help. With the field level help activated, you can click in a field or column to access the control ID and business view information, which is displayed under the Advanced Options section.

For fields on the main form, the control ID will be a single value, such as 25.

Grids also have control IDs. For a traditional form, the grid ID is usually 1. For power forms, subforms, and reusable subforms the grid ID is typically a value other than 1.

The columns within a grid also have unique IDs and are often referenced in conjunction with the grid ID. For example column 28 and 29 in grid 1 would be 1[28,29].

Power forms have more complex IDs. The fields on the main power form are represented with single values. The fields on a subform are complex with an underscore separating them. So field 6 on subform 12 is 12_6. The ID of a re-usable subform is available when viewing the power form that the subform is used on. The IDs of individual fields, a grid, or columns on a re-usable subform is shown in FDA when viewing the subform directly; you cannot get these values when viewing the subform alias.

The returnControlIDs string is bar delimited, without a starting or ending bar.

Example 6–1 Requesting fields and grid columns on a traditional form.

```
formRequest.setReturnControlIDs("19|20|60|125|1[45,49,88]");
```

In this example, 19|20|60|125 represent field control IDs.

1[45,49,88] represents columns in the grid.

Example 6–2 Requesting main form fields, subform fields, main form grid columns, and subform grid columns.

```
formRequest.setReturnControlIDs("33|34|17[24,26,28]|50_45|50_53|50_9[35,39,41]");
```

In this example, 33|34 represent fields on the main form.

50_45|50_53 represent fields on the subform.

17[24,26,28] represent main form grid columns.

50_9[35,39,41] represent subform grid columns.

6.3.8 Example of JSON Code in a Form Service Request

The sample code in [Example 6–3](#) is an example of JSON code in a form service request that executes EnterpriseOne actions in the following order:

1. Open the Find/Browse form in P01012.
2. Enter a value in a QBE field.
3. Enter a value in a field control.
4. Select two check boxes.
5. Click the Find button.

Example 6–3 Executing EnterpriseOne Actions - JSON

```
{
  "token":
  "044B1LYkCUcjQGRxvR3r+LH271iC616psFHOTp9whmkPxE=MDE4MDA2MTYxMDIwOTQ1MDU2NTc0NDY0U2
  9hcFVJMTM4NDQ0NjU2NTUwNQ==",
  "version": "ZJDE0001",
  "formActions": [
    {
```

```

        "command": "SetQBValue",
        "value": "E",
        "controlID": "1[50]"
    },
    {
        "command": "SetControlValue",
        "value": "Al*",
        "controlID": "58"
    },
    {
        "command": "SetCheckboxValue",
        "value": "on",
        "controlID": "62"
    },
    {
        "command": "SetCheckboxValue",
        "value": "on",
        "controlID": "63"
    },
    {
        "command": "DoAction",
        "controlID": "15"
    }
],
"deviceName": "REST Service Testing Tool",
"formName": "P01012_W01012B"
}

```

Example 6-4 Executing EnterpriseOne Actions - JSON Response

```

{
  "fs_P01012_W01012B": {
    "title": "Work With Addresses",
    "data": {
      "lblSearchType_53": {
        "id": 53,
        "editable": false,
        "value": "Search Type",
        "title": "Search Type",
        "dataType": 2
      },
      "txtSearchType_54": {
        "id": 54,
        "editable": true,
        "value": "*",
        "internalValue": "null",
        "title": "Search Type",
        "assocDesc": "",
        "staticText": "Search Type",
        "dataType": 2
      },
      "lblAlphaName_57": {
        "id": 57,
        "editable": false,
        "value": "Alpha Name",
        "title": "Alpha Name",
        "dataType": 2
      },
      "txtAlphaName_58": {

```

```

        "id": 58,
        "editable": true,
        "value": "Allen*",
        "internalValue": "Allen*",
        "title": "Alpha Name",
        "staticText": "Alpha Name",
        "dataType": 2
    },
    "lblDL01_66": {
        "id": 66,
        "editable": false,
        "value": "",
        "title": "",
        "dataType": 2
    },
    "chkDisplayAddress_63": {
        "id": 63,
        "editable": true,
        "value": "on",
        "internalValue": "1",
        "title": "Display Address",
        "dataType": 1
    },
    "chkDisplayPhone_62": {
        "id": 62,
        "editable": true,
        "value": "on",
        "internalValue": "1",
        "title": "Display Phone",
        "dataType": 1
    },
    "gridData": {
        "titles": {
            "col_19": "Address Number",
            "col_20": "Alpha Name",
            "col_40": "Address Line 1",
            "col_44": "City",
            "col_81": "Prefix",
            "col_46": "Phone Number",
            "col_47": "Phone Type",
            "col_48": "Long Address",
            "col_49": "Industry Class",
            "col_50": "Sch Typ",
            "col_51": "Tax ID"
        },
        "rowset": [
            {
                "rowIndex": 0,
                "MOExist": false,
                "mnAddressNumber_19": {
                    "id": 19,
                    "editable": false,
                    "value": "576",
                    "internalValue": 576,
                    "title": "Address Number",
                    "dataType": 9
                },
                "sAlphaName_20": {
                    "id": 20,
                    "editable": false,

```

```
        "value": "Allen",
        "internalValue": "Allen",
        "title": "Alpha Name",
        "dataType": 2
    },
    "sAddressLine1_40": {
        "id": 40,
        "editable": false,
        "value": " ",
        "internalValue": " ",
        "title": "Address Line 1",
        "dataType": 2
    },
    "sCity_44": {
        "id": 44,
        "editable": false,
        "value": " ",
        "internalValue": " ",
        "title": "City",
        "dataType": 2
    },
    "sPrefix_81": {
        "id": 81,
        "editable": false,
        "value": "",
        "internalValue": "",
        "title": "Prefix",
        "dataType": 2
    },
    "sPhoneNumber_46": {
        "id": 46,
        "editable": false,
        "value": "",
        "internalValue": "",
        "title": "Phone Number",
        "dataType": 2
    },
    "sPhoneType_47": {
        "id": 47,
        "editable": false,
        "value": "",
        "internalValue": "",
        "title": "Phone Type",
        "dataType": 2
    },
    "sLongAddress_48": {
        "id": 48,
        "editable": false,
        "value": " ",
        "internalValue": " ",
        "title": "Long Address",
        "dataType": 2
    },
    "sIndustryClass_49": {
        "id": 49,
        "editable": false,
        "value": " ",
        "internalValue": " ",
        "title": "Industry Class",
        "dataType": 2
    }
```

```

    },
    "sSchTyp_50": {
      "id": 50,
      "editable": false,
      "value": "E",
      "internalValue": "E",
      "title": "Sch Typ",
      "dataType": 2
    },
    "sTaxID_51": {
      "id": 51,
      "editable": false,
      "value": " ",
      "internalValue": " ",
      "title": "Tax ID",
      "dataType": 2
    }
  }
],
"summary": {
  "records": 1,
  "moreRecords": false
}
},
"errors": [],
"warnings": []
},
"stackId": 2,
"stateId": 1,
"rid": "f199d7dd4210b9af",
"currentApp": "P01012_W01012B_ZJDE0001",
"sysErrors": []
}

```

6.3.9 Example of API Methods for a Form Service Request

Oracle provides API methods that you can use to set the commands when coding mobile applications.

Important: When setting a date field value, use the form field or QBE Date methods that use the `java.util.Date` for input. These methods format the date value into the proper format for data entry in EnterpriseOne.

Example 6–5 API Methods for Setting Commands

```

FormRequest formRequest = new FormRequest();

formRequest.setFormName("P01012_W01012B");
formRequest.setVersion("ZJDE0001");

//create event holder
FSREvent myFSREvent = new FSREvent();

//add actions in order
myFSREvent.setQBValue("1[50]", searchType);
myFSREvent.setFieldValue("58", name);

```



```

myFSREvent.checkBoxChecked ("62");
myFSREvent.checkBoxChecked ("63");
myFSREvent.doControlAction("15");

//add event holder to the form request
formRequest.addFSREvent(myFSREvent);

// Execute SEND and RECEIVE operation
try {

    JSONObject jsonObject =
(JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
    String postData = jsonObject.toString();

    String response = JDERestServiceProvider.jdeRestServiceCall(postData,
JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_URI);

    P01012_W01012B_FormParent newFormParent =
        ((P01012_W01012B_
FormParent)JSONBeanSerializationHelper.fromJSON(P01012_W01012B_FormParent.class,
response));
}

```

6.3.10 Grid Action Events

In addition to interacting with fields on the form, you can interact with grids using grid action events. If you use a grid action event, you must include "grid" as a required capability in the about.properties. See [Section 6.2, "Understanding AIS Server Capabilities"](#) for more information.

The types of grid action events include:

- Selecting grid rows

This action enables you to delete records in the grid by sending a row select event, followed by a delete button press event, and then finally an OK button press event. This is the exact sequence that a user would follow to delete a record in an EnterpriseOne application.

- Inserting grid rows

This action enables you to insert one or more rows into a grid, setting the column value for each row. This includes text entry columns, drop-down columns, or check box columns. You must include an OK button pressed event to commit the inserts.

- Updating grid rows

This action enables you to update one or more existing grid rows by setting the column values for each row. This includes text entry columns, drop-down columns, or check box columns. You must include an OK button pressed event to commit the updates.

The following table describes the commands that you can use in grid column events to set values for a cell in a grid insert or update event:

Grid Column Event	Description	Parameters
Set Grid Cell Value	Sets the value of a cell in a grid.	"value": "720", "command": "SetGridCellValue", "columnID": "28"
Set Grid Combo Value	Sets the value of a drop-down column in a grid. The value that you send is in the "Code" for the UDC associated with the column.	"value": "ABC" "command": "SetGridComboValue", "columnID": "43"

6.3.10.1 Example of Grid Action Events

This section provides examples of grid action events in both JSON and Oracle MAF code.

The sample code in [Example 6–6](#) is an example of JSON that deletes a phone number in the third row of a grid. It is important to note:

- The row index is zero based.
- You must get the row index based on a previous fetch (since rows may be hidden and the index may not be consecutive).
- By sending 3 form actions, first select row 3, then selecting the Delete button, and then selecting the OK button.
- The form inputs will get the correct set of phone records for address number 6001, who's who line 0.
- The formServiceAction code is a U for update, so the form is in update mode.

Example 6–6 *Selecting Grid Rows - JSON*

```
{
  "token":
  "0443HC90ZH4pq9CScdvJ+nkecf1SJI9q+YGbc71XrGZ7So=MDE5MDA2ODQ4MjcyMDk2MTUwMjg0NDkyOF
  NvYXBVSTeZOTIwNzE5NzY4NzE=",
  "formActions": [
    {
      "command": "SelectRow",
      "controlID": "1.3"
    },
    {
      "command": "DoAction",
      "controlID": "59"
    },
    {
      "command": "DoAction",
      "controlID": "4"
    }
  ],
  "formInputs": [
    {
      "value": "6001",
      "id": "4"
    },
    {
      "value": "0",
```

```

        "id": "5"
      }
    ],
    "formServiceAction": "U",
    "deviceName": "RESTclient",
    "formName": "P0115_W0115A"
  }
}

```

Example 6–7 Selecting Grid Rows - JSON Response

```

{
  "fs_P0115_W0115A": {
    "title": "Phone Numbers",
    "data": {
      "lblDL01_71": {
        "id": 71,
        "editable": false,
        "value": "Ray Allen",
        "title": "Ray Allen",
        "dataType": 2
      },
      "lblWhosWhoLine_52": {
        "id": 52,
        "editable": false,
        "value": "Who's Who Line",
        "title": "Who's Who Line",
        "dataType": 2
      },
      "txtAddressNumber_7": {
        "id": 7,
        "editable": false,
        "value": "6001",
        "internalValue": 6001,
        "title": "Address Number",
        "assocDesc": "Allen, Ray",
        "staticText": "Address Number",
        "dataType": 9
      },
      "lblDL01_54": {
        "id": 54,
        "editable": false,
        "value": "Allen, Ray",
        "title": "Allen, Ray",
        "dataType": 2
      },
      "gridData": {
        "titles": {
          "col_28": "Prefix",
          "col_29": "Phone Number",
          "col_27": "Phone Type",
          "col_66": "Phone Type Description",
          "col_26": "Line Number"
        },
        "rowset": [
          {
            "rowIndex": 0,
            "MOExist": false,
            "sPrefix_28": {

```

```

        "id": 28,
        "editable": true,
        "value": "303",
        "internalValue": "303",
        "title": "Prefix",
        "dataType": 2
    },
    "sPhoneNumber_29": {
        "id": 29,
        "editable": true,
        "value": "334-4000",
        "internalValue": "334-4000",
        "title": "Phone Number",
        "dataType": 2
    },
    "sPhoneType_27": {
        "id": 27,
        "editable": true,
        "value": "HOM",
        "internalValue": "HOM",
        "title": "Phone Type",
        "dataType": 2
    },
    "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Home",
        "internalValue": "Home",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "1",
        "internalValue": 1,
        "title": "Line Number",
        "dataType": 9
    }
},
{
    "rowIndex": 1,
    "MOExist": false,
    "sPrefix_28": {
        "id": 28,
        "editable": true,
        "value": "303",
        "internalValue": "303",
        "title": "Prefix",
        "dataType": 2
    },
    "sPhoneNumber_29": {
        "id": 29,
        "editable": true,
        "value": "555-1212",
        "internalValue": "555-1212",
        "title": "Phone Number",
        "dataType": 2
    },
    "sPhoneType_27": {

```

```

        "id": 27,
        "editable": true,
        "value": "CAR",
        "internalValue": "CAR",
        "title": "Phone Type",
        "dataType": 2
    },
    "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Car or Mobile",
        "internalValue": "Car or Mobile",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "2",
        "internalValue": 2,
        "title": "Line Number",
        "dataType": 9
    }
}
],
"summary": {
    "records": 2,
    "moreRecords": false
}
},
"lblAddressNumber_6": {
    "id": 6,
    "editable": false,
    "value": "Address Number",
    "title": "Address Number",
    "dataType": 9
},
"txtWhosWhoLine_32": {
    "id": 32,
    "editable": false,
    "value": "0",
    "internalValue": 0,
    "title": "Who's Who Line",
    "assocDesc": "Ray Allen",
    "dataType": 9a
}
},
"errors": [],
"warnings": []
},
"stackId": 5,
"stateId": 1,
"rid": "c2a9c1c93a6874f0",
"currentApp": "P0115_W0115A",
"sysErrors": []
}

```

Example 6-8 Selecting Grid Rows - MAF Application Code

This sample code performs the same delete operation as the JSON request example in the preceding section; it deletes a single phone number in a grid of phone numbers.

```
public void deletePhone(int key) {

    AdfmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "false");
    FormRequest formRequest = new FormRequest();

    formRequest.setFormName("P0115_W0115A");
    formRequest.setFormServiceAction("U");

    formRequest.addToFISet("4", addressNumber);
    formRequest.addToFISet("5", "0");

    FSREvent fsrEvent = new FSREvent();

    //get currently selected row
    P0115_W0115A_GridRow selectedRow = getSelectedPhonebyKey(key);

    fsrEvent.selectRow("1", selectedRow.getRowIndex());

    //press Delete button
    fsrEvent.doControlAction("59");

    //press OK button
    fsrEvent.doControlAction("4");

    //add the FSR event to the request
    formRequest.addFSREvent(fsrEvent);

    // Execute SEND and RECEIVE operation
    try {
        // For POST request, set data payload is header delimited with | and
        service input class

        JSONObject jsonObject =
        (JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
        String postData = jsonObject.toString();

        String response = JDERestServiceProvider.jdeRestServiceCall(postData,
        JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_URI);
        P0115_W0115A_FormParent tempFormParent =
        ((P0115_W0115A_
        FormParent)JSONBeanSerializationHelper.fromJSON(P0115_W0115A_FormParent.class,
        response));

        if(tempFormParent.getFs_P0115_W0115A().getErrors() != null &&
        tempFormParent.getFs_P0115_W0115A().getErrors().length>0){

            AdfmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "true");
            }else{

                p0115_W0115A_FormParent.getFs_P0115_
                W0115A().getData().getGridData().setRowsetList(tempFormParent.getFs_P0115_
                W0115A().getData().getGridData().retrieveRowsetList());
            }
        }
    }
}
```

```

        p0115_W0115A_FormParent.getFs_P0115_
W0115A().getData().getGridData().setSummary(tempFormParent.getFs_P0115_
W0115A().getData().getGridData().getSummary());

    }

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException(e.getMessage(),
        AdfException.ERROR);
        throw adfe;
    }

}

```

Example 6-9 Insert Rows - JSON

This sample code is an example of adding two phone numbers with two form actions: a grid action that adds two rows followed by an OK button selection. The form inputs will get the correct set of phone records for address number 6001, who's who line 0. Also, it is important to note that the formServiceAction code is a U for update, which indicates that the form is in update mode.

```

{
  "token":
"044b02/1CT8FViW5A0Sm5GqcqU3P8BB2kwUv0bZzG84YYU=MDE4MDA2NTIyNTk1NjQyNTc1MjQ2MzZU2
9hcFVJMTM5MjE0OTA4MjYyMA==",
  "formInputs": [
    {
      "value": "6001",
      "id": "4"
    },
    {
      "value": "0",
      "id": "5"
    }
  ],
  "formServiceAction": "U",

  "formActions": [
    {
      "gridAction":
      {
        "gridID": "1",
        "gridRowInsertEvents": [
          {
            "gridColumnEvents": [
              {
                "value": "720",
                "command": "SetGridCellValue",
                "columnID": "28"
              },
              {
                "value": "331-4014",
                "command": "SetGridCellValue",
                "columnID": "29"
              }
            ]
          }
        ]
      }
    }
  ]
}

```

```

        {
            "value": "CAR",
            "command": "SetGridCellValue",
            "columnID": "27"
        }
    ]
},
{
    "gridColumnEvents": [
        {
            "value": "303",
            "command": "SetGridCellValue",
            "columnID": "28"
        },
        {
            "value": "421-1010",
            "command": "SetGridCellValue",
            "columnID": "29"
        },
        {
            "value": "HOM",
            "command": "SetGridCellValue",
            "columnID": "27"
        }
    ]
}
    ]
}
},
{
    "command": "DoAction",
    "controlID": "4"
}
],
"deviceName": "RESTclient",
"formName": "P0115_W0115A"
}

```

Example 6–10 Insert Rows - JSON Response

```

{
    "fs_P0115_W0115A": {
        "title": "Phone Numbers",
        "data": {
            "lblDL01_71": {
                "id": 71,
                "editable": false,
                "value": "Ray Allen",
                "title": "Ray Allen",
                "dataType": 2
            },
            "lblWhosWhoLine_52": {
                "id": 52,
                "editable": false,
                "value": "Who's Who Line",
                "title": "Who's Who Line",
            }
        }
    }
}

```



```

    "dataType": 2
  },
  "txtAddressNumber_7": {
    "id": 7,
    "editable": false,
    "value": "6001",
    "internalValue": 6001,
    "title": "Address Number",
    "assocDesc": "Allen, Ray",
    "staticText": "Address Number",
    "dataType": 9
  },
  "lblDL01_54": {
    "id": 54,
    "editable": false,
    "value": "Allen, Ray",
    "title": "Allen, Ray",
    "dataType": 2
  },
  "gridData": {
    "titles": {
      "col_28": "Prefix",
      "col_29": "Phone Number",
      "col_27": "Phone Type",
      "col_66": "Phone Type Description",
      "col_26": "Line Number"
    }
  },
  "rowset": [
    {
      "rowIndex": 0,
      "MOExist": false,
      "sPrefix_28": {
        "id": 28,
        "editable": true,
        "value": "303",
        "internalValue": "303",
        "title": "Prefix",
        "dataType": 2
      },
      "sPhoneNumber_29": {
        "id": 29,
        "editable": true,
        "value": "334-4000",
        "internalValue": "334-4000",
        "title": "Phone Number",
        "dataType": 2
      },
      "sPhoneType_27": {
        "id": 27,
        "editable": true,
        "value": "HOM",
        "internalValue": "HOM",
        "title": "Phone Type",
        "dataType": 2
      },
      "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Home",
        "internalValue": "Home",

```

```

        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "1",
        "internalValue": 1,
        "title": "Line Number",
        "dataType": 9
    }
},
    {
        "rowIndex": 1,
        "MOExist": false,
        "sPrefix_28": {
            "id": 28,
            "editable": true,
            "value": "303",
            "internalValue": "303",
            "title": "Prefix",
            "dataType": 2
        },
        "sPhoneNumber_29": {
            "id": 29,
            "editable": true,
            "value": "555-1212",
            "internalValue": "555-1212",
            "title": "Phone Number",
            "dataType": 2
        },
        "sPhoneType_27": {
            "id": 27,
            "editable": true,
            "value": "CAR",
            "internalValue": "CAR",
            "title": "Phone Type",
            "dataType": 2
        },
        "sPhoneTypeDescription_66": {
            "id": 66,
            "editable": false,
            "value": "Car or Mobile",
            "internalValue": "Car or Mobile",
            "title": "Phone Type Description",
            "dataType": 2
        },
        "mnLineNumber_26": {
            "id": 26,
            "editable": false,
            "value": "2",
            "internalValue": 2,
            "title": "Line Number",
            "dataType": 9
        }
    }
},
    {
        "rowIndex": 2,
        "MOExist": false,
        "sPrefix_28": {

```

```

        "id": 28,
        "editable": true,
        "value": "720",
        "internalValue": "720",
        "title": "Prefix",
        "dataType": 2
    },
    "sPhoneNumber_29": {
        "id": 29,
        "editable": true,
        "value": "331-4014",
        "internalValue": "331-4014",
        "title": "Phone Number",
        "dataType": 2
    },
    "sPhoneType_27": {
        "id": 27,
        "editable": true,
        "value": "CAR",
        "internalValue": "CAR",
        "title": "Phone Type",
        "dataType": 2
    },
    "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Car or Mobile",
        "internalValue": "Car or Mobile",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "3",
        "internalValue": 3,
        "title": "Line Number",
        "dataType": 9
    }
},
    {
        "rowIndex": 3,
        "MOExist": false,
        "sPrefix_28": {
            "id": 28,
            "editable": true,
            "value": "303",
            "internalValue": "303",
            "title": "Prefix",
            "dataType": 2
        },
        "sPhoneNumber_29": {
            "id": 29,
            "editable": true,
            "value": "421-1010",
            "internalValue": "421-1010",
            "title": "Phone Number",
            "dataType": 2
        },
        "sPhoneType_27": {

```

```

        "id": 27,
        "editable": true,
        "value": "HOM",
        "internalValue": "HOM",
        "title": "Phone Type",
        "dataType": 2
    },
    "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Home",
        "internalValue": "Home",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "4",
        "internalValue": 4,
        "title": "Line Number",
        "dataType": 9
    }
}
},
"summary": {
    "records": 4,
    "moreRecords": false
}
},
"lblAddressNumber_6": {
    "id": 6,
    "editable": false,
    "value": "Address Number",
    "title": "Address Number",
    "dataType": 9
},
"txtWhosWhoLine_32": {
    "id": 32,
    "editable": false,
    "value": "0",
    "internalValue": 0,
    "title": "Who's Who Line",
    "assocDesc": "Ray Allen",
    "dataType": 9
}
},
"errors": [],
"warnings": []
},
"stackId": 1,
"stateId": 1,
"rid": "b7ebc0f0832cfbb",
"currentApp": "P0115_W0115A",
"sysErrors": []
}

```

Example 6–11 Insert Rows - MAF Application Code

This sample code is an example of adding one new phone number using grid actions.

Important: Your mobile application should allow adding records only when the record count is below the maximum. You can determine this by checking the `moreRecords` field in the grid summary when you fetch existing records. You will not receive an error message if you attempt to add a record beyond the maximum allowed. The record will simply not be added.

```
public void addPhone() {

    AdmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "false");

    FormRequest formRequest = new FormRequest();

    formRequest.setFormName("P0115_W0115A");
    formRequest.setFormServiceAction("U");

    formRequest.addToFISet("4", addressNumber);
    formRequest.addToFISet("5", "0");

    FSREvent fsrEvent = new FSREvent();
    GridAction gridAction = new GridAction();

    GridRowInsertEvent gri = new GridRowInsertEvent();

    //set the column values
    gri.setGridColumnValue("27", addPhoneType);
    gri.setGridColumnValue("28", addPhonePrefix);
    gri.setGridColumnValue("29", addPhoneNumber);

    //add the row
    gridAction.insertGridRow("1", gri);

    //add the grid action to the events
    fsrEvent.addGridAction(gridAction);

    //press OK button
    fsrEvent.doControlAction("4");

    //add the FSR event to the request
    formRequest.addFSREvent(fsrEvent);

    // Execute SEND and RECEIVE operation
    try {
        // For POST request, set data payload is header delimited with | and
        // service input class

        JSONObject jsonObject =
            (JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
        String postData = jsonObject.toString();

        String response = JDERestServiceProvider.jdeRestServiceCall(postData,
            JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_URI);
        P0115_W0115A_FormParent tempFormParent =
            ((P0115_W0115A_
            FormParent)JSONBeanSerializationHelper.fromJSON(P0115_W0115A_FormParent.class,
            response));
    }
}
```

```

        if(tempFormParent.getFs_P0115_W0115A().getErrors() != null &&
tempFormParent.getFs_P0115_W0115A().getErrors().length>0){

            AdmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "true");
        }else{

            p0115_W0115A_FormParent.getFs_P0115_
W0115A().getData().getGridData().setRowsetList(tempFormParent.getFs_P0115_
W0115A().getData().getGridData().retrieveRowsetList());
            p0115_W0115A_FormParent.getFs_P0115_
W0115A().getData().getGridData().setSummary(tempFormParent.getFs_P0115_
W0115A().getData().getGridData().getSummary());

            AdmfJavaUtilities.setELValue("#{pageFlowScope.addready}", "true");

        } catch (JDERestServiceException e) {
            JDERestServiceProvider.handleServiceException(e);
        } catch (Exception e) {
            AdfException adfe = new AdfException(e.getMessage(),
AdfException.ERROR);
            throw adfe;
        }

    }
}

```

Example 6–12 Update Rows - JSON

This sample code is an example of updating two phone numbers. You must specify the index of the row you want to update. The row index is included in the information returned when you query the grid. Therefore, you must perform a query before you update a row. In this example, the JSON code updates rows 2 and 4 and sets values in each of the three columns for these rows.

This sample code also includes syntax that shows a column that contains a drop-down selection. The value should be the 'code' value, not the description.

```

"gridColumnEvents": [
  {
    "value": "30",
    "command": "SetGridComboValue",
    "columnID": "43"
  }
]

```

This sample code shows the phones update request:

```

{
  "token":
"044t+mF8cq2gxqvAlH1SziE9EnBfs5QYN1j3ZQpmFAW9tc=MDE5MDA2ODQ0NjQ4MjI5OTYxODQ5MjQxN1
NvYXBVSTeZOTIwNzI3MjAzMTM=",
  "formInputs": [
    {
      "value": "6001",
      "id": "4"
    },
    {
      "value": "0",
      "id": "5"
    }
  ]
},

```

```

"formActions": [
  {
    "gridAction":
      {
        "gridID": "1",
        "gridRowUpdateEvents": [
          {
            "rowNumber": 2,
            "gridColumnEvents": [
              {
                "value": "720",
                "command": "SetGridCellValue",
                "columnID": "28"
              },
              {
                "value": "111-1111",
                "command": "SetGridCellValue",
                "columnID": "29"
              },
              {
                "value": "CAR",
                "command": "SetGridCellValue",
                "columnID": "27"
              }
            ]
          },
          {
            "rowNumber": 4,
            "gridColumnEvents": [
              {
                "value": "303",
                "command": "SetGridCellValue",
                "columnID": "28"
              },
              {
                "value": "22233333",
                "command": "SetGridCellValue",
                "columnID": "29"
              },
              {
                "value": "CAR",
                "command": "SetGridCellValue",
                "columnID": "27"
              }
            ]
          }
        ]
      }
  },
  {
    "command": "DoAction",
    "controlID": "4"
  }
],
"deviceName": "RESTclient",
"formName": "P0115_W0115A"
}

```

Example 6-13 Update Rows - JSON Response

```

{
  "fs_P0115_W0115A": {
    "title": "Phone Numbers",
    "data": {
      "lblDL01_71": {
        "id": 71,
        "editable": false,
        "value": "Ray Allen",
        "title": "Ray Allen",
        "dataType": 2
      },
      "lblWhosWhoLine_52": {
        "id": 52,
        "editable": false,
        "value": "Who's Who Line",
        "title": "Who's Who Line",
        "dataType": 2
      },
      "txtAddressNumber_7": {
        "id": 7,
        "editable": false,
        "value": "6001",
        "internalValue": 6001,
        "title": "Address Number",
        "assocDesc": "Allen, Ray",
        "staticText": "Address Number",
        "dataType": 9
      },
      "lblDL01_54": {
        "id": 54,
        "editable": false,
        "value": "Allen, Ray",
        "title": "Allen, Ray",
        "dataType": 2
      },
      "gridData": {
        "titles": {
          "col_28": "Prefix",
          "col_29": "Phone Number",
          "col_27": "Phone Type",
          "col_66": "Phone Type Description",
          "col_26": "Line Number"
        },
        "rowset": [
          {
            "rowIndex": 0,
            "MOExist": false,
            "sPrefix_28": {
              "id": 28,
              "editable": true,
              "value": "303",
              "internalValue": "303",
              "title": "Prefix",
              "dataType": 2
            },
            "sPhoneNumber_29": {
              "id": 29,
              "editable": true,
              "value": "334-4000",

```



```

        "internalValue": "334-4000",
        "title": "Phone Number",
        "dataType": 2
    },
    "sPhoneType_27": {
        "id": 27,
        "editable": true,
        "value": "HOM",
        "internalValue": "HOM",
        "title": "Phone Type",
        "dataType": 2
    },
    "sPhoneTypeDescription_66": {
        "id": 66,
        "editable": false,
        "value": "Home",
        "internalValue": "Home",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "1",
        "internalValue": 1,
        "title": "Line Number",
        "dataType": 9
    }
},
    {
        "rowIndex": 1,
        "MOExist": false,
        "sPrefix_28": {
            "id": 28,
            "editable": true,
            "value": "303",
            "internalValue": "303",
            "title": "Prefix",
            "dataType": 2
        },
        "sPhoneNumber_29": {
            "id": 29,
            "editable": true,
            "value": "555-1212",
            "internalValue": "555-1212",
            "title": "Phone Number",
            "dataType": 2
        },
        "sPhoneType_27": {
            "id": 27,
            "editable": true,
            "value": "CAR",
            "internalValue": "CAR",
            "title": "Phone Type",
            "dataType": 2
        },
        "sPhoneTypeDescription_66": {
            "id": 66,
            "editable": false,
            "value": "Car or Mobile",

```

```

        "internalValue": "Car or Mobile",
        "title": "Phone Type Description",
        "dataType": 2
    },
    "mnLineNumber_26": {
        "id": 26,
        "editable": false,
        "value": "2",
        "internalValue": 2,
        "title": "Line Number",
        "dataType": 9
    }
},
    {
        "rowIndex": 2,
        "MOExist": false,
        "sPrefix_28": {
            "id": 28,
            "editable": true,
            "value": "720",
            "internalValue": "720",
            "title": "Prefix",
            "dataType": 2
        },
        "sPhoneNumber_29": {
            "id": 29,
            "editable": true,
            "value": "111-1111",
            "internalValue": "111-1111",
            "title": "Phone Number",
            "dataType": 2
        },
        "sPhoneType_27": {
            "id": 27,
            "editable": true,
            "value": "CAR",
            "internalValue": "CAR",
            "title": "Phone Type",
            "dataType": 2
        },
        "sPhoneTypeDescription_66": {
            "id": 66,
            "editable": false,
            "value": "Car or Mobile",
            "internalValue": "Car or Mobile",
            "title": "Phone Type Description",
            "dataType": 2
        },
        "mnLineNumber_26": {
            "id": 26,
            "editable": false,
            "value": "3",
            "internalValue": 3,
            "title": "Line Number",
            "dataType": 9
        }
    }
},
    {
        "rowIndex": 3,
        "MOExist": false,

```

```

        "sPrefix_28": {
            "id": 28,
            "editable": true,
            "value": "303",
            "internalValue": "303",
            "title": "Prefix",
            "dataType": 2
        },
        "sPhoneNumber_29": {
            "id": 29,
            "editable": true,
            "value": "22233333",
            "internalValue": "22233333",
            "title": "Phone Number",
            "dataType": 2
        },
        "sPhoneType_27": {
            "id": 27,
            "editable": true,
            "value": "CAR",
            "internalValue": "CAR",
            "title": "Phone Type",
            "dataType": 2
        },
        "sPhoneTypeDescription_66": {
            "id": 66,
            "editable": false,
            "value": "Car or Mobile",
            "internalValue": "Car or Mobile",
            "title": "Phone Type Description",
            "dataType": 2
        },
        "mnLineNumber_26": {
            "id": 26,
            "editable": false,
            "value": "4",
            "internalValue": 4,
            "title": "Line Number",
            "dataType": 9
        }
    },
    "summary": {
        "records": 4,
        "moreRecords": false
    }
},
"lblAddressNumber_6": {
    "id": 6,
    "editable": false,
    "value": "Address Number",
    "title": "Address Number",
    "dataType": 9
},
"txtWhosWhoLine_32": {
    "id": 32,
    "editable": false,
    "value": "0",
    "internalValue": 0,
    "title": "Who's Who Line",

```

```

        "assocDesc": "Ray Allen",
        "dataType": 9
    }
},
"errors": [],
"warnings": []
},
"stackId": 2,
"stateId": 1,
"rid": "b7ebc0f0832cfbb",
"currentApp": "P0115_W0115A",
"sysErrors": []
}

```

Example 6-14 Update Rows - MAF Application Code

This sample code is an example of updating a single phone row from an MAF application.

```

public void updatePhone(int key) {

    AdfmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "false");

    FormRequest formRequest = new FormRequest();

    formRequest.setFormName("P0115_W0115A");
    formRequest.setFormServiceAction("U");

    formRequest.addToFISet("4", addressNumber);
    formRequest.addToFISet("5", "0");

    FSREvent fsrEvent = new FSREvent();
    GridAction gridAction = new GridAction();

    GridRowUpdateEvent gru = new GridRowUpdateEvent();

    //get currently selected row
    P0115_W0115A_GridRow selectedRow = getSelectedPhonebyKey(key);

    //set the column values
    gru.setGridColumnValue("27", selectedRow.getSPhoneType_27().getValue());
    gru.setGridColumnValue("28", selectedRow.getSPrefix_28().getValue());
    gru.setGridColumnValue("29", selectedRow.getSPhoneNumber_29().getValue());

    //update the row based on it's index - zero based
    gridAction.updateGridRow("1",
        selectedRow.getRowIndex(),
        gru);

    //add the grid action to the events
    fsrEvent.addGridAction(gridAction);

    //press OK button
    fsrEvent.doControlAction("4");

    //add the FSR event to the request
    formRequest.addFSREvent(fsrEvent);
}

```

```

        // Execute SEND and RECEIVE operation
        try {
            // For POST request, set data payload is header delimited with | and
            service input class

            JSONObject jsonObject =
            (JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
            String postData = jsonObject.toString();

            String response = JDERestServiceProvider.jdeRestServiceCall(postData,
            JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_URI);
            P0115_W0115A_FormParent tempFormParent =
            ((P0115_W0115A_
            FormParent)JSONBeanSerializationHelper.fromJSON(P0115_W0115A_FormParent.class,
            response));

            if(tempFormParent.getFs_P0115_W0115A().getErrors() != null &&
            tempFormParent.getFs_P0115_W0115A().getErrors().length>0){

                AdfmfJavaUtilities.setELValue("#{pageFlowScope.errors}", "true");

            }else{

                p0115_W0115A_FormParent.getFs_P0115_
                W0115A().getData().getGridData().setRowsetList(tempFormParent.getFs_P0115_
                W0115A().getData().getGridData().retrieveRowsetList());
                p0115_W0115A_FormParent.getFs_P0115_
                W0115A().getData().getGridData().setSummary(tempFormParent.getFs_P0115_
                W0115A().getData().getGridData().getSummary());

            }

        } catch (JDERestServiceException e) {
            JDERestServiceProvider.handleServiceException(e);
        } catch (Exception e) {
            AdfException adfe = new AdfException(e.getMessage(),
            AdfException.ERROR);
            throw adfe;
        }
    }
}

```

6.3.11 Query Events (Release 9.1 Update 5.2)

You can configure a form service request to send ad hoc queries to EnterpriseOne web client application forms that support the query control.

To add a query, you include a single query object in the form service request. A query object includes parameters that contain the same query criteria that you would use to set up a query in EnterpriseOne. The parameters determine:

- How the query runs.** You can configure query option parameters to load grid records in the form or clear all other fields in the form before the query runs. You can also specify whether the results of the query should match all (AND) or any (OR) of the conditions specified in the query.

- **The conditions of the query.** The query object includes condition parameters that specify the control ID of the columns or fields that you want to query and an operator for filtering results that are equal to, greater than, or less than a particular value.

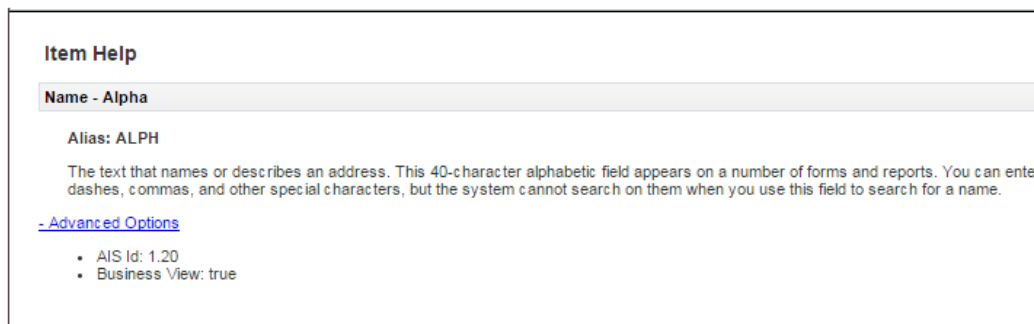
Important: Queries will work only if the field or columns identified in the query are part of the business view.

- **The value used for the search criteria in the query.** The query object includes value parameters that specify the value or range of values that you want displayed in the query results.

Before you add a query object to a form service request, access the form in the EnterpriseOne web client and use the query control to gather the criteria for the query object parameters. For more information about setting up a query, see "Understanding the Query Control" in the *JD Edwards EnterpriseOne Tools Foundation Guide*.

Also, in the EnterpriseOne form, you need to identify the control ID of the field or column that you want to query, and verify that the field or column is part of the business view. To do so, click the Help button (question mark in the upper right corner of a form) and then click the Item Help option to access field-level help. With the field level help activated, you can click in a field or column to access the control ID and business view information, which is displayed under the Advanced Options section as shown in [Example 6–15](#).

Example 6–15 Example of Control ID and Business View Information Displayed under Advanced Options in the EnterpriseOne Web Client Item Help



In the Item Help, the syntax of the control ID is 1.20 with 1 representing the grid ID and 20 representing the column ID, which are separated by a dot (.). In the parameter for the query request, the same control ID must be presented with the following syntax: 1[20]. See the [Table 6–2, "Query Condition Parameters"](#) for more information.

6.3.11.1 Query Object Parameters

The following tables describe the parameters for a query object.

Table 6–1 Query Option Parameters

Parameter	Description	Values
autoFind	Directs the query to automatically press Find on the form to populate the grid records. You do not need to include events to press the Find button if you use autoFind.	true, false

Table 6–1 (Cont.) Query Option Parameters

Parameter	Description	Values
matchType	Determines if you want the query to search for records that match all (AND) or any (OR) of the specified conditions.	MATCH_ALL, MATCH_ANY
autoClear	Do you want to clear all other fields on the form (for example default filter fields).	true, false

Table 6–2 Query Condition Parameters

Parameter	Description	Value
controlId	The control ID that the condition applies to. This is the field that you add to the query from the form when using the web client to create a Query. It is either a filter field or a grid column that is associated with the business view.	Example of control IDs: "28", "1[34]"
operator	The comparison operation to use with the query.	For all types, valid values are: BETWEEN, LIST, EQUAL, NOT_EQUAL, LESS, LESS_ EQUAL, GREATER, GREATER_EQUAL For strings, valid values are: STR_START_WITH, STR_ END_WITH, STR_CONTAIN, STR_BLANK, STR_NOT_ BLANK

Table 6–3 Query Value Parameters

Parameter	Description	Value
content	This is either a literal value to be used in the comparison operation, or it relates to a special value ID.	Examples of values are: "23", "Joe", "2"
specialValueId	This is a special value, mostly for dates that may be today, or calculated dates from today. For calculated dates, the content field is used in the calculation.	Valid values are: LITERAL, TODAY, TODAY_ PLUS_DAY, TODAY_ MINUS_DAY, TODAY_ PLUS_MONTH, TODAY_ MINUS_MONTH, TODAY_ PLUS_YEAR, TODAY_ MINUS_YEAR

6.3.11.2 Query Object Examples

This section contains JSON and API code examples of the query object.

Example 6–16 Query - JSON

This is an example of JSON code that executes a query on the W42101C form in EnterpriseOne.

```
{
  "token":
  "044mK8eBlup02jaE4BAvXzaRXsMAM7edfHybw26zSZga1w=MDE5MDA2MjUyMTQ1MDE0Njg1NzMwODE2MF
```

```

NvYXBVSTE0MTIzNTc1MDkwMTC=",
    "maxPageSize": "20",
    "formServiceAction": "R",
    "returnControlIDs": "350|360|41[129,130,116,125,132,123]",
    "query": {
      "condition": [
        {
          "value": [
            {
              "content": "2",
              "specialValueId": "LITERAL"
            }
          ],
          "controlId": "41[129]",
          "operator": "EQUAL"
        },
        {
          "value": [
            {
              "content": "2",
              "specialValueId": "TODAY_MINUS_YEAR"
            }
          ],
          "controlId": "41[116]",
          "operator": "GREATER"
        },
        {
          "value": [
            {
              "content": "7000",
              "specialValueId": "LITERAL"
            },
            {
              "content": "8000",
              "specialValueId": "LITERAL"
            }
          ],
          "controlId": "41[125]",
          "operator": "BETWEEN"
        },
        {
          "value": [
            {
              "content": "00070",
              "specialValueId": "LITERAL"
            },
            {
              "content": "00001",
              "specialValueId": "LITERAL"
            },
            {
              "content": "00077",
              "specialValueId": "LITERAL"
            }
          ],
          "controlId": "360",
          "operator": "LIST"
        }
      ]
    }
  ],

```



```

    "autoFind": true,
    "matchType": "MATCH_ALL",
    "autoClear": false
  },

  "deviceName": "RESTclient",
  "formName": "P42101_W42101C"
}

```

Example 6–17 Query - Mobile Framework API

This is an example of API code that executes a query on the W42101C form in EnterpriseOne.

```

public void executeQuery() {

    FormRequest formRequest = new FormRequest();

    formRequest.setReturnControlIDs("350|360|41[129,130,116,125,132,123]");
    formRequest.setFormName("P42101_W42101C");
    formRequest.setVersion("ZJDE0001");
    formRequest.setFormServiceAction("R");

    // Execute SEND and RECEIVE operation
    try {

        Query query = new Query();
        query.setAutoFind(true); //automatically presses find button (don't
need to use form action to find)
        query.setMatchType(Query.MATCH_ALL); //this is an AND operation on
all of the conditions
        query.setAutoClear(false); //clears any existing filter values on the
form

        //find all line numbers matching
        NumberCondition numCon = query.addNumberCondition("41[129]",
NumericOperator.EQUAL());
        numCon.setValue(linNo);

        //and requested date within num years from today
        DateCondition dateCon = query.addDateCondition("41[116]",
DateOperator.GREATER());
        dateCon.setSpecialDateValue(DateSpecialValue.TODAY_MINUS_YEAR(),
years);

        //and sold to between
        BetweenCondition betCon = query.addBetweenCondition("41[125]");
        betCon.setValues(soldTo1, soltTo2);

        //and company in list
        ListCondition listCon = query.addListCondition("360");
        addValuesToList(listCon);

        //add the query object to the request
        formRequest.setQuery(query);

        formRequest.setMaxPageSize("20");
        // For POST request, set data payload is header delimited with | and
service input class

```

```

        JSONObject jsonObject = (JSONObject)
JSONBeanSerializationHelper.toJSON(formRequest);
        String postData = jsonObject.toString();

        //Call to JDERestServiceProvider with parameters json string, method
(POST), URI (formservice)
        String response = JDERestServiceProvider.jdeRestServiceCall(postData,
JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_URI);

        p41010_W42101C_FormParent=
            (P42101_W42101C_FormParent)
JSONBeanSerializationHelper.fromJSON(P42101_W42101C_FormParent.class,
response);

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException(e.getMessage(),
ADFException.ERROR);
        throw adfe;
    }
}

```

6.4 Understanding Batch Form Service

If you make several sequential calls to forms without any data dependencies between them, consider using the Batch Form Service. The Batch Form Service will improve your mobile application's performance.

6.4.1 Batch Form Service - JSON Input and Output

The Batch Form Service requires JSON input and output.

The request URL is: `POST http://aissserver:port/jderest/batchformservice`

You can use this URL in a REST testing tool. In the preceding URL, `batchformservice` represents the URI, which you define when you perform the `jdeRestServiceCall` from the API.

The following examples show JSON input and JSON output in a Batch Form Service.

Example 6–18 JSON Input in a Batch Form Service

The JSON consists of an array of form requests along with other single form request fields such as token, environment, role, and so forth. You may send as many requests as desired to the same or different form. Each request is executed individually by the EnterpriseOne HTML (JAS) Server and all responses are compiled into a single response.

This example shows a request for two form executions: W54GS220A and W54HS230A. Each have different form interconnect sets, control sets, and so forth. They are completely independent requests.

```

{
  "formRequests": [
    {

```

```

    "formInputs": [
      {
        "value": "09/17/2013",
        "id": "3"
      },
      {
        "value": "1",
        "id": "4"
      }
    ],
    "returnControlIDs": "1[19,20,21,27,28,92,174,177,178]|324",
    "findOnEntry": "TRUE",
    "formName": "P54HS220_W54HS220A"
  },
  {
    "formInputs": [
      {
        "value": "09/17/2013",
        "id": "2"
      },
      {
        "value": "1",
        "id": "3"
      }
    ],
    "returnControlIDs": "1[35,36,37,44,102,105,200,201,277]|368",
    "findOnEntry": "TRUE",
    "formName": "P54HS230_W54HS230A"
  }
],
"token": "jkfdjfdk"
}

```

Example 6–19 JSON Output in a Batch Form Service

The JSON output contains one form element for each form that is called. The forms are numbered in the order they were requested. This is not an array of elements; they are individually defined.

The following sample code is an example of JSON output. For `fs_0_P54HS220_W54HS220A`, the 0 indicates it was the first form requested. For `fs_1_P54HS230_W54HS230A`, the 1 indicates it was the second form requested.

```

{
  "fs_0_P54HS220_W54HS220A": {
    "title": "One View Incident Summary Inquiry",
    "data": {
      "txtAlertDescription": {
        "id": 324,
        "value": "Recent Incidents",
        "internalValue": "Recent Incidents",
        "title": "Alert Description",
        "dataType": 2
      },
      "gridData": {
        "titles": {
          "col_19": "Incident Number",
          "col_20": "Incident Description",
          "col_21": "Incident Date",
          "col_27": "Incident Status",

```

```

        "col_28": "Incident Severity",
        "col_92": "Incident Handler Address",
        "col_174": "Incident Handler Name",
        "col_177": "Incident Status Description",
        "col_178": "Incident Severity Description"
    },
    "rowset": [
        {
            "mnIncidentNumber": {
                "id": 19,
                "value": "113",
                "internalValue": 113,
                "title": "Incident Number",
                "dataType": 9
            },
            "sIncidentDescription": {
                "id": 20,
                "value": "Gallon drums exposed to sea water rusted and leaked
motor oil and coolant",
                "internalValue": "Gallon drums exposed to sea water rusted
and leaked motor oil and coolant",
                "title": "Incident Description",
                "dataType": 2
            },
            "dtIncidentDate": {
                "id": 21,
                "value": "09/17/2013",
                "internalValue": 1379397600000,
                "title": "Incident Date",
                "dataType": 11
            },
            "sIncidentStatus": {
                "id": 27,
                "value": "01",
                "internalValue": "01",
                "title": "Incident Status",
                "dataType": 2
            },
            "sIncidentSeverity": {
                "id": 28,
                "value": "03",
                "internalValue": "03",
                "title": "Incident Severity",
                "dataType": 2
            },
            "mnIncidentHandlerAddress": {
                "id": 92,
                "value": "6001",
                "internalValue": 6001,
                "title": "Incident Handler Address",
                "dataType": 9
            },
            "sIncidentHandlerName": {
                "id": 174,
                "value": "Allen, Ray",
                "internalValue": "Allen, Ray",
                "title": "Incident Handler Name",
                "dataType": 2
            },
            "sIncidentStatusDescription": {

```

```

        "id": 177,
        "value": "Active",
        "internalValue": "Active",
        "title": "Incident Status Description",
        "dataType": 2
    },
    "sIncidentSeverityDescription": {
        "id": 178,
        "value": "Medium",
        "internalValue": "Medium",
        "title": "Incident Severity Description",
        "dataType": 2
    }
},
    {
        "mnIncidentNumber": {
            "id": 19,
            "value": "117",
            "internalValue": 117,
            "title": "Incident Number",
            "dataType": 9
        },
        "sIncidentDescription": {
            "id": 20,
            "value": " ",
            "internalValue": " ",
            "title": "Incident Description",
            "dataType": 2
        },
        "dtIncidentDate": {
            "id": 21,
            "value": "10/01/2013",
            "internalValue": 1380607200000,
            "title": "Incident Date",
            "dataType": 11
        },
        "sIncidentStatus": {
            "id": 27,
            "value": " ",
            "internalValue": " ",
            "title": "Incident Status",
            "dataType": 2
        },
        "sIncidentSeverity": {
            "id": 28,
            "value": " ",
            "internalValue": " ",
            "title": "Incident Severity",
            "dataType": 2
        },
        "mnIncidentHandlerAddress": {
            "id": 92,
            "value": "0",
            "internalValue": 0,
            "title": "Incident Handler Address",
            "dataType": 9
        },
        "sIncidentHandlerName": {
            "id": 174,
            "value": "",

```

```

        "internalValue": "null",
        "title": "Incident Handler Name",
        "dataType": 2
    },
    "sIncidentStatusDescription": {
        "id": 177,
        "value": "Default",
        "internalValue": "Default",
        "title": "Incident Status Description",
        "dataType": 2
    },
    "sIncidentSeverityDescription": {
        "id": 178,
        "value": "Default",
        "internalValue": "Default",
        "title": "Incident Severity Description",
        "dataType": 2
    }
}
],
"summary": {
    "records": 2,
    "moreRecords": false
}
},
"errors": [],
"warnings": []
},
"fs_1_P54HS230_W54HS230A": {
    "title": "One View Incident People Inquiry",
    "data": {
        "txtAlertDescription": {
            "id": 368,
            "value": "Recent Reportable Incidents",
            "internalValue": "Recent Reportable Incidents",
            "title": "Alert Description",
            "dataType": 2
        },
        "gridData": {
            "titles": {
                "col_35": "Incident Number",
                "col_36": "Incident Description",
                "col_37": "Incident Date",
                "col_44": "Incident Status",
                "col_102": "Incident Handler Address",
                "col_105": "Incident Severity",
                "col_200": "Incident Status Description",
                "col_201": "Incident Severity Description"
            },
            "rowset": [],
            "summary": {
                "records": 0,
                "moreRecords": false
            }
        }
    }
},
"errors": [],
"warnings": []
}

```

```
}

```

6.4.2 Implementing the Batch Form Service

This section describes how to implement the Batch Form Service in your mobile application.

6.4.2.1 Batch Request Parent Class

To consume the JSON response, the mobile application must have a class that matches the response. You must manually code this class because you have to provide the forms in the order in which they are being called.

Example 6–20 Batch Request Parent Class

The following sample code is an example of what the new class looks like with one member defined, with the form level type for each of the forms called. Notice that the names of the class members match the names returned by the JSON response. This is the most important aspect of the class, to enable the deserialization from JSON to this object.

The form objects, which are P54HS220_W54HS220A and P54HS230_W54HS230A in the following sample code, are the same ones generated by the AIS Client Class Generator and are used for a single form request.

```
package com.oracle.e1.formservicetypes

public class BatchRequestParent {

    private P54HS220_W54HS220A fs_0_P54HS220_W54HS220A;
    private P54HS230_W54HS230A fs_1_P54HS230_W54HS230A;

    public BatchRequestParent() {
        super();
    }

    //getters and setters for everything

}
```

6.4.2.2 Performing a Batch Form Request

In the data control class, create a method for performing the batch form request. An example of a method named `batchFormRequest()` follows the steps below.

To create this method:

1. Create a `BatchFormRequest` object.
2. Add each form request to the list of requests. Each form request is an object called `SingleFormRequest`.
3. Populate these requests in the same manner that you would use for a single form request.
4. Call the service with `uri JDERestServiceProvider.BATCH_FORM_SERVICE_URI` and marshal the JSON response to an instance of `BatchRequestParent` class.

Example 6–21 Batch Form Request

You can use each member of the `BatchRequestParent` class the same way you would use the response from a single form request. In the following sample code, each form in the batch is saved to a member variable of the main data control class.

```
public void batchFormRequest() {

    BatchFormRequest batchFormRequest = new BatchFormRequest();
    //Date used for all
    // Format Incident Date to send to E1
    SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");
    String sDate = dateFormat.format(getSearchDate());

    //recentIncidents
    SingleFormRequest formRequest = new SingleFormRequest();
    formRequest.setFindOnEntry("TRUE");
    formRequest.setReturnControlIDs("1[19,20,21,27,28,92,174,177,178]|324");
    formRequest.setFormName("P54HS220_W54HS220A");
    formRequest.addToFISet("3", sDate);
    formRequest.addToFISet("4", "1");
    batchFormRequest.getFormRequests().add(formRequest);

    // recentReportableIncidents
    formRequest = new SingleFormRequest();
    formRequest.setFindOnEntry("TRUE");
    formRequest.setReturnControlIDs("1[35,36,37,44,102,105,200,201,277]|368");
    formRequest.setFormName("P54HS230_W54HS230A");
    formRequest.addToFISet("2", sDate);
    formRequest.addToFISet("3", "1");
    batchFormRequest.getFormRequests().add(formRequest);
    try {
        // For POST request, set data payload is header delimited with | and
service input class

        JSONObject jsonObject =
(JSONObject)JSONBeanSerializationHelper.toJSON(batchFormRequest);

        String postData = jsonObject.toString();

        String response = JDERestServiceProvider.jdeRestServiceCall(postData,
JDERestServiceProvider.POST, JDERestServiceProvider.BATCH_FORM_SERVICE_URI);

        BatchRequestParent batchParent =

(BatchRequestParent)JSONBeanSerializationHelper.fromJSON(BatchRequestParent.class,
response);

        if(batchParent != null){

            //recentIncidents
            this.p54hs220_formIncidents = new P54HS220_W54HS220A_FormParent();
            this.p54hs220_formIncidents.setFs_P54HS220_
W54HS220A(batchParent.getFs_0_P54HS220_W54HS220A());

            IncidentCount ic = new IncidentCount();
            ic.setCount(p54hs220_formIncidents.getFs_P54HS220_
W54HS220A().getData().getGridData().getSummary().getRecords());
```



```

        ic.setTitle("Recent");
        ic.setScoreboardNumber("1");
        incidentCounts.add(ic);

        // recentReportableIncidents
        this.p54hs230_formParent = new P54HS230_W54HS230A_FormParent();
        this.p54hs230_formParent.setFs_P54HS230_
W54HS230A(batchParent.getFs_3_P54HS230_W54HS230A());
        ic = new IncidentCount();
        ic.setCount(p54hs230_formParent.getFs_P54HS230_
W54HS230A().getData().getGridData().getSummary().getRecords());
        ic.setTitle("Reportable");
        ic.setScoreboardNumber("4");
        incidentCounts.add(ic);

    }

    } catch (Exception e) {
        AdfException adfe = new AdfException("Rest Call Failed" +
e.getMessage(), AdfException.ERROR);
        throw adfe;
    }

}

```

6.5 Working with the EnterpriseOne REST Services Interface

You can make REST calls directly to the AIS Server without using APIs from the JDEMobileFramework.jar. To do so, use a REST service testing tool to send JSON text to AIS Server endpoints. Each endpoint provides a particular service that the mobile application can use to interact with EnterpriseOne applications. You can access AIS Server endpoints using this URL format:

```
http://<server>:<port>/jderest/<URI>
```

See [Section 2.3.4, "EnterpriseOne Rest Services Interface"](#) for an illustration of JSON input and output communication between a mobile device and the AIS Server and between the AIS Server and the EnterpriseOne HTML Server.

[Table 6–4](#) provides a description of the available endpoints on the AIS Server along with the HTTP method used to access them. Unless otherwise stated, the request and response are in the form of JSON strings.

Table 6–4 AIS Server Endpoints

Endpoint URI	HTTP Method	Description of Service
/defaultconfig	GET	Response includes information about the AIS Server including the release level, JAS configuration and capabilities list. Different versions of AIS have different capabilities, even if the client API you are using is up to date with the latest capabilities, the AIS Server may not be.
/tokenrequest	POST	Based on the input, the response will contain login information including a login token and user details.
/tokenrequest/logout	POST	Based on the input (AIS token), the response will be a code of 200 if successful and 500 if the logout fails.
/formservice	POST	Based on the input, the response will contain a JSON representation of the form requested.
/batchformservice	POST	Based on the input, the response will contain a JSON representation of all of the forms requested.
/file/gettext	POST	Base on the input, the response will contain the text for the first text media object.
/file/updatesetext	POST	Base on the input, the response will contain the status of the text update.
/file/list	POST	Base on the input, the response will contain the list of media objects for the structure and key requested.
/file/upload	POST (Multi-part Form)	The response will contain the details of the uploaded file, including the media object sequence number
/file/download	POST	This response is a Multi-Part form including the data for the attachment.
/file/delete	POST	The response indicates the success or failure to delete the media object for the sequence number passed in.
/poservice	POST	Based on the input, the response will contain the processing option values for the requested application and version.
/log	POST	Base on the input, the AIS Server will write a log entry with the information passed to the log service.
/appstack	POST	This service is available starting with the Tools 9.1 Update 5 release. Based on the input, the response will contain the current form open on the stack and any stack related information.

6.5.1 Using a REST Services Client to Interact with AIS

In the text area in the bottom left of the form, enter the JSON string that is the input to the tokenrequest service. Entering an environment and role is optional. Include them only if you want to override the default values.

Example 6–22 Acceptable Input for the defaultconfig Service on JSON

The defaultconfig service is the only service that uses an HTTP GET operation. It accepts one parameter for the required capabilities in the client application. If any required capabilities listed in that parameter are missing, the response will indicate the missing capabilities in the "requiredCapabilityMissing" field.

```
{
  "jasHost": "jasserver.domain.com",
  "jasPort": "8412",
  "jasProtocol": "http",
  "defaultEnvironment": "DV910",
```

```

"defaultRole": "*ALL",
"displayEnvironment": true,
"displayRole": true,
"displayJasServer": false,
"defaultJasServer": "http://jasserver.domain.com:port",
"ssoAllowed": true,
"allowSelfSignedSsl": false,
"sessionTimeout": "30",
"timeToLive": "720",
"aisVersion": "EnterpriseOne 9.1.4.6",
"capabilityList": [
  {
    "name": "grid",
    "shortDescription": "Grid Actions",
    "longDescription": "Ability to update, insert and delete grid records.",
    "asOfRelease": "9.1.4.4"
  },
  {
    "name": "editable",
    "shortDescription": "Enabled/Disabled",
    "longDescription": "Ability to indicate if form field or grid cell is
editable (enabled) or not (disabled).",
    "asOfRelease": "9.1.4.4"
  },
  {
    "name": "log",
    "shortDescription": "Logging",
    "longDescription": "Endpoint exposed for logging to AIS server log from
client",
    "asOfRelease": "9.1.4.6"
  },
  {
    "name": "processingOption",
    "shortDescription": "Processing Options",
    "longDescription": "Processing Option Service exposed for fetching PO
values from E1",
    "asOfRelease": "9.1.4.6"
  },
  {
    "name": "ignoreFDAFindOnEntry",
    "shortDescription": "Ignore FDA Find On Entry",
    "longDescription": "Ability to use the IgnoreFDAFindOnEntry flag",
    "asOfRelease": "9.1.4.6"
  }
],
"requiredCapabilityMissing": false,
"keepJasSessionOpen": true,
"jasSessionCookieName": "JSESSIONID"
}

```

Example 6–23 Acceptable Input for the tokenrequest Service in JSON

```

{
"username": "jde",
"password": "jde",
"deviceName": "RESTclient",
"environment": "JDV910",
"role": "*ALL"
}

```

Example 6–24 Acceptable Input for the tokenrequest Service in JSON Response

```

{
  "username": "jde",
  "environment": "JDV910",
  "role": "*ALL",
  "jasserver": "http://jasserver.domain.com:port",
  "userInfo": {
    "token":
"044kbwMICIsL8P4jttMj/VtpnEhrSK17i7fa2q8V+hVD1c=MDE4MDEwNjA1NTMwODkwMzQ1ODY0MTkyUk
VTVGNsaWVudE0MTI2MDg3MDgzODY=",
    "langPref": " ",
    "locale": "en",
    "dateFormat": "MDE",
    "dateSeperator": "/",
    "simpleDateFormat": "MM/dd/yyyy",
    "decimalFormat": ".",
    "addressNumber": 2111,
    "alphaName": "Ingram, Paul"
  },
  "userAuthorized": false,
  "version": null,
  "poStringJSON": null,
  "altPoStringJSON": null,
  "aisSessionCookie": "W3XmCk8k6vnb3H-dwxSdBpCGZWAb7kh5D4gzemFoqoFqcoWgwqF_
!-1217528743!1412608708388"
}

```

6.5.1.1 Form Request Attributes

The following list contains a description of the form request attributes. For an example of a form request, see [Example 6–25](#).

- **maxPageSize.** (optional) Maximum number of rows to return to the grid. The default is 100 if not passed, the maximum is 500.

The value in the "Watchlists: Upper Limit on Max Records to Return" setting in the Web Object Settings section of the EnterpriseOne HTML Server configuration file (jas.ini) supersedes the maxPageSize setting. The "Watchlists: Upper Limit on Max Records to Return" setting is typically set to a higher value (500 is the default), but you might have to modify it if your results are not as expected. See "Configuration Groups" in the *JD Edwards EnterpriseOne Tools Server Manager Guide* for more information on how to modify server configuration settings.

Important: Starting with EnterpriseOne Tools release 9.1 Update 5.3, the EnterpriseOne HTML Server uses a different setting—DB Fetch Limit in the Web Runtime Interactivity section—to determine maximum numbers of rows returned to the grid. The default value for DB Fetch Limit is 2000.

- **returnControlIDs.** (optional) Indicates which form and grid fields the service passes back. Grid is always 1 and has an array of fields. Form fields are bar delimited after grid, for example 1[19,20,50,48] | 54 | 55. For power forms, indicate the control ID of the subform and then underscore and the control within the subform, for example 1_20 | 1_22[23,34].
- **formInputs.** (optional) Collection of ID value pairs that represent the form interconnect input to the form. Associate a string for the FI ID with the value to be passed into that FI.

- **version.** The version of the application, for example ZJDE0001.
- **formName.** (required) Application and form name, for example P01012_W01012B.
- **formServiceAction.** (optional) The CRUD operation the form will perform.
- **token.** (required) The EnterpriseOne token that was returned from the tokenrequest service.
- **environment.** (optional) EnterpriseOne environment that was used to request the token.
- **role.** (optional) The EnterpriseOne role used to request the token.
- **findOnEntry.** (optional) Valid values are TRUE or FALSE. Performs a find automatically when the EnterpriseOne application launches. In the EnterpriseOne application this autofind event occurs after post dialog initialized.
- **ignoreFDAFindOnEntry.** (optional) Valid values are TRUE or FALSE. Applies to applications that have the box checked for "Automatically Find on Entry" in the grid in FDA. Allows the form service caller to control if that flag is used or not.
- **formActions.** (optional) A set of actions to be performed on the form, in the order listed. The actions can include entering a value into a form field, QBE field, selecting a radio button, pressing a button, and selecting a check box value.

Example 6–25 Form Request

```
{
  "token":
  "044RnByWbW3FbLzpxWjSg55QzZWguAGnYqUNMlyB30IgyU=MDE5MDA2ODg0MDE5NjYwNzNm10DcyNDA5N1
  NvYXBVSTeZODc0ODc4OTEzNTc=",
  "maxPageSize": "10",
  "returnControlIDs": "1[19,20]58|62",
  "version": "ZJDE0001",
  "formInputs": [
    {
      "value": "E",
      "id": "2"
    }
  ],
  "formActions": [
    {
      "value": "E",
      "command": "SetQBEValue",
      "controlID": "1[50]"
    },
    {
      "value": "A1*",
      "command": "SetControlValue",
      "controlID": "58"
    },
    {
      "value": "on",
      "command": "SetCheckboxValue",
      "controlID": "62"
    },
    {
      "value": "on",
      "command": "SetCheckboxValue",
      "controlID": "63"
    }
  ]
}
```

```

        "command": "DoAction",
        "controlID": "15"
    }
],
    "role": "*ALL",
    "environment": "JDV910",
    "formsServiceAction": "R",
    "deviceName": "RESTclient",
    "formName": "P01012_W01012B"
}

```

6.5.1.2 Calling FormService on Local EnterpriseOne HTML (JAS) Server through the AIS Server

You have the option to use this technique if you have FDA changes locally and want to test the JSON output while still accessing the AIS Server. This is the closest approximation of how the mobile application will call the REST services.

Configure your local EnterpriseOne HTML Server to accept requests from the AIS Server. To do so:

1. Locate the jas.ini file on the local EnterpriseOne Windows client (development client) machine:

```

C:\E910_
1\system\OC4J\j2ee\home\applications\webclient.ear\webclient\WEB-INF\classes\jas.ini

```

2. Add or modify the form service section like this:

```

#specify hosts allowed to call the form service and media object service
[FORMSERVICE]
allowedhosts=127.0.0.1|10.123.456.7

```

The two preceding entries are the local host IP address and the IP address of the AIS Server. This enables you to make calls through the AIS Server to your local EnterpriseOne HTML Server. For the AIS Server, you can also use `allowedhosts=*` if you do not know the AIS Server IP address or want to allow access for any AIS Server.

3. Restart the EnterpriseOne Windows client.
4. Test the configuration by repeating the preceding steps, but include the following change to the JSON requests:

For tokenrequest: Indicate the JAS server, which is your local EnterpriseOne HTML Server. Also, Oracle recommends that you always include the environment and role because the defaults stored on the AIS Server may not work with a local instance (such as JDV910 vs DV910).

```

{"username": "JDE",
 "password": "jde",
 "deviceName": "RESTclient",
 "jasserver": "http://dnnlaurentvml:8888",
 "environment": "DV910",
 "role": "*ALL"
}

```

6.6 Understanding Text Media Object Attachments

The AIS Server provides a method to get and update the first text attachment stored for a particular media object data structure key in EnterpriseOne.

The JDEMobileFramework API provides the following two services for managing text media object attachments in mobile applications:

- [gettext Service](#)
- [updatetext Service](#)

See Also:

- "User-Generated Data Structures" in the *JD Edwards EnterpriseOne Tools Data Structure Design Guide* for more information about the function of media object data structures in EnterpriseOne

6.6.1 gettext Service

The URL for getting text media objects is:

```
http://<ais-server>:<port>/jderest/file/gettext
```

When Only the first text media object is returned; it does not handle multiple text attachments. Only plain text is supported. While the original text media object is stored in HTML or RTF format (based on the settings on the EnterpriseOne HTML Server), the tags will be removed and the response will be in plain text with line breaks in Unicode format.

The following table describes the expected JSON input for the gettext service:

Field	Value Description	Example
token (String)	The token you received when you executed a token request.	"044TqjlQVSNHKvokvhWFKa8MYc kPgG..."
moStructure (String)	The Media Object Structure	"ABGT" or "GT0801A"
moKey (Array of Strings)	The key to that media object structure.	["7"] or ["2015", "283", "2"]
formName (String)	The form where this media object is used.	"P01012_W01012B"
version (String)	The version of the app where this media object is used.	"ZJDE0001"
deviceName (String)	The device making the call to the service.	"FUSE" or "iPhoneSimulator"

Example 6–26 gettext Service Input

```
{
  "token":
  "044TqjlQVSNHKvokvhWFKa8MYc kPgGJdrd3CEDkzz2YjLQ=MDE5MDA2Nzg3NjEzOTA2MTY1MzIwMTkyMF
  NvYXBVSTeZOTk1ODA5MzQ0OTg=",
  "moStructure": "ABGT",
  "moKey": [
    "7"
  ],
  "formName": "P01012_W01012B",
  "version": "ZJDE0001",
  "deviceName": "RESTclient"
}
```

Example 6–27 *gettext Service Response*

```
{
  "text": "9.1.4.6 \\u000aFirst Line \\u000aSecond Line \\u000aThird Line
\\u000aFourth Line \\u000aFifth line",
  "isRTF": false
}
```

6.6.2 updatetext Service

The URL for updating text media objects is:

`http://<ais-server>:<port>/jderest/file/updatetext`

Only plain text is supported.

The following table describes the expected JSON input for the updatetext service:

Field	Value Description	Example
token (String)	The token you received when you executed a token request.	"044TqjlQVSNHKvokvhWFKa8MYc kPgG..."
moStructure (String)	The media object structure.	"ABGT" or "GT0801A"
moKey (Array of Strings)	The key to the media object structure.	["7"] or ["2015", "283", "2"]
formName (String)	The form where this media object is used.	"P01012_W01012B"
version (String)	The version of the application where the media object is used.	"ZJDE0001"
deviceName (String)	The device making the call to the service.	"RESTClient" or "iPhoneSimulator"
inputText (String)	The text you are updating for the first text media object.	"Update Text"
appendText (Boolean)	A flag to indicate if the text should append to (true) or replace (false) the existing first text attachment.	true or false

Example 6–28 *Update Text Service Input*

```
{
  "token":
  "044X9NgLP5VHQIW0zJcWseFjMtjrnZ35TUFmVV3NW9a9g4=MDE5MDA2NDY5MzQ1NjkzNDM1MjE3NjEyOF
NvYXBVSTE0MDE3MTC4MDkzNTk=",
  "moStructure": "ABGT",
  "moKey": [
    "7"
  ],
  "formName": "P09E2011_W09E2011A",
  "version": "ZJDE0001",
  "inputText": "\nFirst Line \nSecond Line \nThird Line \nFourth Line \nFifth
line",
  "appendText": false,
  "deviceName": "RESTclient"
}
```


Example 6–29 Update Text Service Response

```
{
  "updateTextStatus": "Success"
}
```

6.6.3 JDEMobileFramework API Methods for Managing Text Media Objects

The `MediaObjectOperations` class in the `JDEMobileFramework` API provide methods for managing text media objects from mobile applications. The same two methods described in the preceding sections, `getText` and `updateText`, are exposed in this API.

MediaObjectOperations Class

The following table describes the methods in the `MediaObjectOperations` class:

Return	Method	Description
<code>MediaObjectGetTextResponse</code>	<code>getTextMediaObject(MediaObjectGetTextRequest mediaObgetTextRequest)</code> throws <code>Exception</code>	Takes a <code>MediaObjectGetTextRequest</code> , calls the get text service, replaces unicode returns and line feeds with system line separator and returns a <code>MediaObjectGetTextResponse</code> with the resulting text attachment value.
<code>MediaObjectUpdateTextResponse</code>	<code>updateTextMediaObject(MediaObjectUpdateTextRequest mediaObgetUpdateTextRequest)</code> throws <code>Exception</code>	Takes a <code>MediaObjectUpdateTextRequest</code> , calls the update text service and returns result in <code>MediaObjectUpdateTextResponse</code> .

Example 6–30 Get Example

```
public void getText() {
    MediaObjectGetTextRequest moGetText = new MediaObjectGetTextRequest();

    moGetText.setFormName(formName);
    moGetText.setVersion(version);
    moGetText.setMoStructure(moStructure);

    //set mo key
    moGetText.addMoKeyValue(mokey);

    try {

        MediaObjectGetTextResponse response =
MediaObjectOperations.getTextMediaObject(moGetText);
        this.setFirstText(response.getText());

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException(e, AdfException.ERROR);
        throw adfe;
    }

}
```

Example 6–31 Append Example

```
public void appendText() {
    MediaObjectUpdateTextRequest moUpdateText = new
MediaObjectUpdateTextRequest();

    moUpdateText.setFormName(formName);
    moUpdateText.setVersion(version);
    moUpdateText.setMoStructure(moStructure);

    //set mo key
    moUpdateText.addMoKeyValue(mokey);
    moUpdateText.setAppendText(true);
    moUpdateText.setInputText(this.getFirstText());

    try {

        MediaObjectUpdateTextResponse response =
MediaObjectOperations.updateTextMediaObject(moUpdateText);

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException(e, AdfException.ERROR);
        throw adfe;
    }

}
```

Example 6–32 Update Example

```
public void updateText() {
    MediaObjectUpdateTextRequest moUpdateText = new
MediaObjectUpdateTextRequest();
    moUpdateText.setFormName(formName);
    moUpdateText.setVersion(version);
    moUpdateText.setMoStructure(moStructure);

    //set mo key
    moUpdateText.addMoKeyValue(mokey);
    moUpdateText.setAppendText(false);
    moUpdateText.setInputText(this.getFirstText());

    try {

        MediaObjectUpdateTextResponse response =
MediaObjectOperations.updateTextMediaObject(moUpdateText);

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException(e, AdfException.ERROR);
        throw adfe;
    }

}
```

}

6.7 Understanding URL Media Object Attachments (Release 9.1 Update 5.2)

You can use the `addURLMediaObject` method to include URL media object attachments in a mobile application. A URL media object attachment provides a link to a web page or a file located on a server. In EnterpriseOne, a URL media object displays the target of a URL in the Media Object Viewer.

The `addURLMediaObject` method uses a media object data structure key to identify the record to which the attachment belongs.

The following is a description of the method and the fields that make up the request and the response of the method:

Method - `addURLMediaObject`

Description - Adds a media object of the type URL.

Returns - `MediaObjectAddUrlResponse`

Parameters - `MediaObjectAddUrlRequest`

MediaObjectAddUrlRequest

Field	Type	Description
<code>moStructure</code>	String	The media object structure ID, for example GT0801, GT48100.
<code>moKey</code>	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the <code>JDEmfUtilities.convertMillisecondsToYMDString()</code> method to pass dates.
<code>formName</code>	String	The application and form usually associated with this media object structure, for example P01012_W01012B. This is for security purposes. Security for media objects is based on the form being used.
<code>version</code>	String	The version of the application noted in the <code>formName</code> . This is for security purposes. Security for media objects is based on the form being used.
<code>urlText</code>	String	The text of the URL, for example <code>www.oracle.com</code> or <code>http://www.oracle.com</code>

MediaObjectAddUrlResponse

Field	Type	Description
<code>urlText</code>	String	The text of the url, for example <code>www.oracle.com</code> or <code>http://www.oracle.com</code>
<code>saveUrl</code>	String	The url as it was saved in the media object table F00165.

Field	Type	Description
sequence	int	The sequence number for the newly added url type media object.

Example 6–33 URL Media Object - JSON Request and Response

The following code is an example of a JSON request:

```
{
  "moStructure": "ABGT",
  "moKey": [
    "4242"
  ],
  "formName": "P01012_W01012B",
  "version": "ZJDE0001",
  "token": "044BPeneep/jkkldafjdkla",
  "urlText": "http://www.oracle.com",
  "deviceName": "RESTclient"
}
```

The following code is an example of a JSON response:

```
{
  "saveURL": "http://www.oracle.com",
  "urlText": "http://www.oracle.com",
  "sequence": 5
}
```

Example 6–34 URL Media Object - Mobile Framework API

The following is an example of API code for a URL media object.

```
public void addURL() throws Exception {

    //set request info include URLs so they don't have to be fetched later
    MediaObjectAddUrlRequest mediaObjectAddUrlRequest = new
    MediaObjectAddUrlRequest();

    mediaObjectAddUrlRequest.setFormName("P01012_W01012B");
    mediaObjectAddUrlRequest.setVersion("ZJDE00001");
    mediaObjectAddUrlRequest.setMoStructure("ABGT");
    mediaObjectAddUrlRequest.addMoKeyValue("4242");

    mediaObjectAddUrlRequest.setUrlText("http://www.oracle.com");

    MediaObjectAddUrlResponse mediaObjectAddUrlResponse = new
    MediaObjectAddUrlResponse();
    try {
        //get the list of available files for this media object

        mediaObjectAddUrlResponse =
    MediaObjectOperations.addUrlMediaObject(mediaObjectAddUrlRequest);

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        AdfException adfe = new AdfException("Media Object addURL Failed " + "
    " + e, AdfException.ERROR);
    }
}
```

```

        throw adfe;
    }
}

```

6.8 Understanding the Media Object API for Photo Media Object Attachments

You can incorporate photo attachments into native Oracle MAF mobile applications using the JDEMobileFramework.jar, which is a JDE Mobile Helper. The JDEMobileFramework.jar contains a Media Object API that provides classes and methods that enable access to media objects from EnterpriseOne. It works in conjunction with the AIS Server and the MediaObjectRequest capability of the EnterpriseOne HTML Server.

MediaObjectOperations is an abstract class that contains all of the static methods needed to manage media objects. This section describes the following four main operations:

- List
- Download
- Upload
- Delete

6.8.1 List

Method - getMediaObjectList

Description - Returns a list of Media Object details based on the information provided in the request.

Returns - MediaObjectListResponse

Parameters - MediaObjectListRequest

MediaObjectListRequest

Field	Type	Description
moStructure	String	The Media Object structure ID (such as GT0801, GT48100)
moKey	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the JDEmfUtilities.convertMillisecondsToYMDString() method to pass dates.
formName	String	The application and form usually associated with this media object structure (e.g. P01012_W01012B). This is for security purposes. Security for media objects is based on the form being used.
version	String	The version of the application noted in the formName. This is for security purposes. Security for media objects is based on the form being used.
includeURLs	boolean	When true, the response will include the JAS URL for each file type media object.

Field	Type	Description
includeData	boolean	When true, the response will include the Base64 encoded data for a 40 X 40 size of the image in the data field of the response.
moTypes	ArrayList <String>	Valid types are 1 and 5, which are the two file type media objects. Type 1 file attachments are from a Media Object Queue. Type 5 attachments are files uploaded individually to the EnterpriseOne HTML Server.
extensions	ArrayList	Use this field to further reduce the data set by indicating the extensions of the files you want to be included in the response (such as "jpg", "gif", "png", "pdf").
thumbnailSize	<String> int	This field is available with the Tools 9.1 Update 5 release. Use this in conjunction with the includeData flag. When data is included in the response you can pass an integer value in this field to control the size of the thumbnail returned in the data field. A value of 0 will return the default size of 40x40. The maximum size is 200, any value over 200 will return images 200x200.

MediaObjectListResponse

Field	Type	Description
mediaObjects	MediaObjectListItem []	An array of media object list items.

MediaObjectListItem

Field	Type	Description
downloadUrl	String	When true, the response will include the JAS URL for each file type media object.
file	String	When true, the response will include the Base64 encoded data for any image file in the data field of the response. For text type media objects the text will be include in the data field of the response.
itemName	String	A user given name for the media object item. When viewing media objects on JAS this is the name shown directly under each media object item in the left panel of the MO control.
link	String	
moType	int	Type 1 file attachments are from a Media Object Queue. Type 5 attachments are files uploaded individually to the JAS server.
queue	String	For type 1 file attachments from a Media Object Queue the name of the queue.
sequence	int	The sequence the number of that individual attachment in the set of attachments.
updateDate	String	The date that the media object was last updated.

Field	Type	Description
updateHourOfDay	int	The hour that the media object was last updated.
updateMinuteOfHour	int	The minute that the media object was last updated.
updateSecondOfMinute	int	The second that the media object was last updated.
updateUserID	String	The user that last updated that media object.
hasValidTimestamp	boolean	Indicates if time stamp was valid.
isDefaultImage	boolean	Indicates if image is default image.
isMisc	boolean	<unknown>
isOLE	boolean	Is an OLE attachment.
isShortCut	boolean	<unknown>
isText	boolean	Is a text type attachment.
isUpdated	boolean	<unknown>
isURL	boolean	Is a URL type attachment.
data	String	For an image file this is the Base64 encoded string of the image data for a 40x40 size of the image.
thumbFileLocation	String	The location of the thumbnail (40x40) saved on the device. A file is written for each image that has Base64 data included. Files are only written if data is requested.

6.8.2 Download

Method - downloadMediaObject

Description - Downloads a full sized media object to a file on the device, based on the key information provided in the request.

Returns - MediaObjectDownloadResponse

Parameters - MediaObjectDownloadRequest

MediaObjectDownloadRequest

Field	Type	Description
moStructure	String	The Media Object structure ID, for example GT0801, GT48100.
moKey	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the JDEmfUtilities.convertMillisecondsToYMDString() method to pass dates.
formName	String	The application and form usually associated with this media object structure, for example P01012_W01012B. This is for security purposes. Security for media objects is based on the form being used.

Field	Type	Description
version	String	The version of the application noted in the formName. This is for security purposes. Security for media objects is based on the form being used.
downloadURL	String	The downloadURL provided in the list response. When this value is already known, provide it so the service will not do the extra work of getting the URL.
fileName	String	This will be the name of the file saved on the device file system. It is easiest (and uniqueness is guaranteed) to pass the file provided in the list response.
sequence	int	The sequence number of this individual file in the set of attachments.
height	int	When a value is passed in this field, the image will be scaled to this size before it is returned. If you provide only height, the width will be scaled to maintain the aspect ratio.
width	int	When a value is passed in this field, the image will be scaled to his size before it is returned. If you provide only width, the height will be scaled to maintain the aspect ratio.

FileAttachment

Field	Type	Description
fileName	String	The name of the file as stored in the database.
fileLocation	String	The location of the file saved to the device file system. This includes the "file://" prefix so it can be directly used in image tags in an amx page.
itemName	String	A string value the user may provide as the name of the attachment (if not supplied the file name will be used.)
sequence	int	The sequence number of this individual attachment.
thumbFileLocation	String	The file location of the thumbnail image (this value is not updated when the download method is called).
downloadUrl	String	The JAS URL of the file attachment (this value is not populated when the download method is called).

6.8.3 Upload

Method - uploadMediaObject

Description - Uploads a single file to the media object database based on key values provided in the request.

Returns - MediaObjectUploadResponse

Parameters - MediaObjectUploadRequest

MediaObjectUploadRequest

Field	Type	Description
moStructure	String	The Media Object structure ID (such as GT0801, GT48100).
moKey	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the <code>JDEmfUtilities.convertMillisecondsToYMDString()</code> method to pass dates.
formName	String	The application and form usually associated with this media object data structure (such as P01012_W01012B). This is for security purposes. Security for media objects is based on the form being used.
version	String	The version of the application noted in the formName. This is for security purposes. Security for media objects is based on the form being used.
file	FileAttachment	The resulting downloaded file with the name, fileLocation, sequence and itemName fields updated.

FileAttachment

Field	Type	Description
fileName	String	The name of the file as stored in the database. (This value is not required for upload.)
fileLocation	String	The location of the file saved to the device file system. This includes the "file://" prefix so it can be directly used in image tags in an amx page.
itemName	String	A descriptive name. If not passed, the filename will be used. The extension will be added automatically.
sequence	int	The sequence number of this individual attachment (this value is not required for upload).
thumbFileLocation	String	The file location of the thumbnail image (this value is not required for upload).
downloadUrl	String	The JAS URL of the file attachment (this value is not required for upload).

MediaObjectUploadResponse

Field	Type	Description
itemName	String	A user given name for the media object item. When viewing media objects on JAS, this is the name shown directly under each media object item in the left panel of the MO control. The extension will be added automatically. If not passed, the file name is used.

Field	Type	Description
sequence	int	The sequence number of this individual attachment (assigned during the upload).

6.8.4 Delete

Method - deleteMediaObject

Description - Deletes the file in the media object database based on key values provided in the request. Deletes the file on the device file system also.

Returns - MediaObjectDeleteResponse

Parameters - MediaObjectDeleteRequest

MediaObjectDeleteRequest

Field	Type	Description
moStructure	String	The Media Object structure ID (e.g. GT0801, GT48100).
moKey	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the <code>JDEmfUtilities.convertMillisecondsToYMDString()</code> method to pass dates.
formName	String	The application and form usually associated with this media object structure (e.g. P01012_W01012B). This is for security purposes. Security for media objects is based on the form being used.
version	String	The version of the application noted in the formName. This is for security purposes. Security for media objects is based on the form being used.
sequence	int	The sequence number of this individual file in the set of attachments.
fileLocation	String	The location of the file on the device file system.

MediaObjectDeleteResponse

Field	Type	Description
deleteStatus	String	The value 'Success' will be returned for successful deletes. Any other value is a failure.
error	String	In the case of failure details of the error will be included here.

6.9 Understanding Processing Options

Starting with EnterpriseOne Tools release 9.1.4.4, an AIS service is available for handling processing options.

The URI for the service is:

`http://<AIS Server>:<port>/jderest/poservice`

The input JSON is:

```

{
  "token":
  "044uZUG2Uk1Vd6hzmAhPfILitA2pVLDVKLOYdh4HR71D7s=MDE5MDA2ODM3NTQ3NzU4MDMwNTg2MzY4MF
  NvYXBVSTeZOTcwNTYxMTIxMTE=",

  "applicationName": "P01012",
  "version": "ZJDE0001",
  "deviceName": "RESTclient"
}

```

The response JSON is like this:

```

{
  "application": "P01012",
  "version": "ZJDE0001",
  "processingOptions": {
    "GoToSupplierMaster_5": {
      "type": 1,
      "value": " "
    },
    "GoToCustomerMaster_6": {
      "type": 1,
      "value": " "
    },
    "GoToCSMS_8": {
      "type": 1,
      "value": " "
    },
    "HideTax_4": {
      "type": 1,
      "value": " "
    },
    "SearchTypeDefault_7": {
      "type": 2,
      "value": " "
    },
    "cTypeCode_11": {
      "type": 1,
      "value": "A"
    }
  },
  "errors": ""
}

```

6.9.1 Using the AIS Service for Processing Options in Your Mobile Application

You can call the service passing an application and version. You can obtain the version from the mobile application processing options, which specify the version of the EnterpriseOne application the mobile application uses. For more information about the processing options for mobile applications, see the *JD Edwards EnterpriseOne Applications Functionality for Mobile Devices Implementation Guide*.

If you use this service, you must add "processingOption" to the list of required capabilities in the about.properties. If you do not do this, you will receive an error at runtime. See [Section 6.2, "Understanding AIS Server Capabilities"](#) for more information.

```

ProcessingOptionRequestpoRequest = new ProcessingOptionRequest();
poRequest.setEnvironment(ApplicationGlobals.getInstance.getLoginResponse().getEnvironment());
poRequest.setRole(ApplicationGlobals.getInstance.getLoginResponse().getRole());
poRequest.setJasserver(ApplicationGlobals.getInstance.getLoginResponse().getJasserver());
poRequest.setToken(ApplicationGlobals.getInstance.getLoginResponse().getUserInfo().getToken());
poRequest.setApplicationName("P5648203");
poRequest.setVersion("GNJ001");

try {

    JSONObject jsonObject = (JSONObject) JSONBeanSerializationHelper.toJSON(poRequest);
    String postData = jsonObject.toString();

    //send in po Request json, POST and poservice
    String response =
        JDERestServiceProvider.jdeRestServiceCall(postData,
poRequest.POST, poRequest.PO_SERVICE);

    //response can be serialized to ProcessingOptionSet class
    ProcessingOptionsSet poSet =
        (ProcessingOptionsSet) JSONBeanSerializationHelper.fromJSON(ProcessingOptionsSet.class, response);

    if (poSet != null) {
        //get the individual option and cast it to a variable of matching type
        String attachmentAllowed = (String)poSet.getOptionValue ("szAttachmentAllowed_1");
    }
}

```

Remember to inspect the "errors" element of the response in case an error was encountered when trying to fetch the processing options you requested.

There are six supported data types. These are based on the data item used in the Processing Option Design Aid for each option.

You can get the type of the option before attempting to cast it, which is the recommended method. Or you can just cast it to the type you expect, because it is unlikely to change. The default is String, so you will always be able to get to a string version of the option value.

Type Code	Type Constant	Java Type	JDE DD Type
1	STRING_TYPE	String	String
2	CHAR_TYPE	String	Character
9	BIG_DECIMAL_TYPE	BIG Decimal	Math Numeric
11	DATE_TYPE	Date	Date
15	INTEGER_TYPE	Integer	Integer
55	CALENDAR_TYPE	Calendar	Utime

6.10 Understanding the Application Stack Service (Tools Release 9.1 Update 5)

The application stack is a service that enables interactive communication with applications running on the EnterpriseOne web client. By using the application stack service, you can perform form interconnects to receive data from the resulting form. You can perform more complex interactions with applications that have cross-form transaction boundaries, for example where you do not want the header saved until the details are added and so forth. Also, with the application stack, you can implement a record reservation in mobile applications that corresponds to a record reservation in the web application.

The purpose of application stack processing is that it enables you to establish a session for a specific application and maintain that session across calls. So you initiate a session or 'open' it with one service call, then you can have any number of service calls to 'execute' actions against the currently running application. Finally, you 'close' the session with a service call.

To accomplish this stateful model, some additional data is managed for each service call, including stack ID, state ID, and rid. Also, each request to execute actions against the running application must be sent with the formOID to ensure the form you are expecting is indeed the currently running form.

Note: You can execute actions on only one form during a stack request. So if your request performs a form interconnect, you will have to submit a subsequent request to operate on the second form running after the interconnect.

6.10.1 Service Endpoint

The application stack service is exposed by the AIS Server at the endpoint:

```
http://<aishost>:<port>/jderest/appstack
```

6.10.2 Capability

The application stack capability is exposed in the default configuration as "applicationStack." You must add this to your used or required capability lists within your mobile application to use this capability.

6.10.3 Prerequisite

The AIS Server must be configured to "Keep JAS Sessions Open" so that the session can be maintained across service calls.

6.10.4 JSON Example of an Application Stack Request

This section shows an example of JSON code that performs the following actions:

- Accessing the Address Book application to find a record.
See [Open Application: Request and Response](#).
- Taking the row exit to the Phones form.
See [Execute Actions on Application: Request and Response](#).
- Adding a phone number.

See [Adding a Phone Number](#).

- Saving and closing the application stack.

See [Execute Close Application: Request and Response](#).

6.10.4.1 Open Application: Request and Response

Example 6–35 shows an example of the application stack request. It contains all of the other environment and credential information that is included in a form service request. After this information, it contains a single instance of a form service request called `formRequest`. You can use this to include any action you want to take on this original form, just like a normal form service request. Most important is the `action`, which is "open" in this case. This tells the EnterpriseOne web client to keep the form open so that you can interact with it on future requests.

So for this request with the Address Book (P01012), the search type is set to "E" to perform a "find" and retrieve the first five records.

Important: The `maxPageSize` indicated in this originating request will be the max page size for all requests in this stack. If you do not specify a `maxPageSize`, it uses the form service request default of 100.

Example 6–35 Open Application - Request

```
{
  "token":
  "044J57pHCCB3AzH7/W6Vetm0+yVG6pnKOP1823587SfY6o=MDE5MDA2MTYxOTE0NTE0NjE2OTEyNTg4OF
  NvYXBVSTE0MDEyMTA5MDE1MDE=",
  "action": "open",
  "formRequest": {
    "returnControlIDs": "54|1[19,20]",
    "maxPageSize": "5",
    "formName": "P01012_W01012B",
    "version": "ZJDE0001",
    "findOnEntry": "TRUE",
    "formInputs": [
      {
        "value": "E",
        "id": "2"
      }
    ]
  },
  "deviceName": "RESTclient"
}
```

Example 6–36 Open Application - Response

```
{
  "fs_P01012_W01012B": {
    "title": "Work With Addresses",
    "data": {
... condensed...
    },
    "errors": [],
    "warnings": []
  },
  "stackId": 1,
  "stateId": 1,
  "rid": "e51b593df7a884ea",
}
```

```

    "currentApp": "P01012_W01012B_ZJDE0001",
    "sysErrors": []
  }

```

The response contains the data (including the 5 rows, which are not shown), along with additional information used for all subsequent interactions with the open application.

6.10.4.2 Execute Actions on Application: Request and Response

Next, the JSON code should contain commands to select one of the records returned and access the row exit to Phones (through a form interconnect). With this service call, make sure to pass in the values for stack ID, state ID and rid from the last call, along with the token. Also, indicate an action of "execute," so it knows you are interacting with an open application.

This execute type request contains an actionRequest which can include returnControlIDs for the form you expect to end on and an array of formActions to execute on the form indicated by formOID.

Example 6-37 Execute Actions on Application - Request

```

{
  "token":
  "044DCcituJDHbxBhPcgOuhCb0Av/xnNiUFxqP'TVD6hDfnU=MDE5MDA2MTc1NjYwMzYwMzk0MTE0OTY5NlNvYXBVSTE0MDEyMTkzOTIxMzA=",
  "stackId": 1,
  "stateId": 1,
  "rid": "e51b593df7a884ea",
  "action": "execute",
  "actionRequest": {
    "returnControlIDs": "32|7|1[28,29,66]",
    "maxPageSize": "4",
    "formOID": "W01012B",
    "formActions": [
      {
        ".type": "com.oracle.e1.jdemf.FormAction",
        "command": "SelectRow",
        "controlID": "1.0"
      },
      {
        ".type": "com.oracle.e1.jdemf.FormAction",
        "command": "DoAction",
        "controlID": "65"
      }
    ]
  },
  "deviceName": "RESTclient"
}

```

Example 6-38 Execute Actions on Application - Response

```

{
  "fs_P0115_W0115A": {
    "title": "Phone Numbers",
    "data": {
      ...condensed..
      "summary": {

```

```

        "records": 3,
        "moreRecords": false
    }
},
"errors": [],
"warnings": []
},
"stackId": 1,
"stateId": 2,
"rid": "e51b593df7a884ea",
"currentApp": "P0115_W0115A_ZJDE0001",
"sysErrors": []
}

```

Notice that the state ID has increased after each service call. Also notice that the "currentApp" has changed, and the JSON is representing the Phones application.

6.10.4.3 Adding a Phone Number

[Example 6–39](#) shows how to execute an additional action on this stack that adds a phone number. You can do this as many times as you want, incrementing the state ID each time. The action does not include saving the information; it only populates the grid.

Example 6–39 Adding a Phone Number - Request

```

{
  "token":
  "044DCcituJDHbxBhPcgOuhCb0Av/xnNiUFxqPTVD6hDfnU=MDE5MDA2MTc1NjYwMzYwMzk0MTE0OTY5N1
  NvYXBVSTE0MDEyMTkzOTIxMzA=",
  "stackId": 1,
  "stateId": 2,
  "rid": "e51b593df7a884ea",
  "action": "execute",
  "actionRequest": {
    "formOID": "W0115A",
    "formActions": [
      {
        "gridAction": {
          "gridID": "1",
          "gridRowInsertEvents": [
            {
              "gridColumnEvents": [
                {
                  "value": "720",
                  "command": "SetGridCellValue",
                  "columnID": "28"
                },
                {
                  "value": "12345",
                  "command": "SetGridCellValue",
                  "columnID": "29"
                },
                {
                  "value": "CAR",
                  "command": "SetGridCellValue",
                  "columnID": "27"
                }
              ]
            }
          ]
        }
      }
    ]
  }
}

```



```

    }
  ]
},
"deviceName": "RESTclient",
"ssoEnabled": true
}

```

6.10.4.4 Execute Close Application: Request and Response

Lastly, the JSON code should contain commands to press the Save button and close the application stack. This saves all phone records that were added to the grid.

Example 6-40 Execute Close Application - Request

```

{
  "token":
  "044DCcituJDHbxBhPcgOuhCb0Av/xnNiUFxqPTVD6hDfnU=MDE5MDA2MTc1NjYwMzYwMzk0MTE0OTY5Nl
  NvYXBVSTE0MDEyMTkzOTIxMzA=",
  "stackId": 1,
  "stateId": 3,
  "rid": "e51b593df7a884ea",
  "action": "close",
  "actionRequest": {
    "formOID": "W0115A",
    "formActions": [
      {
        "command": "DoAction",
        "controlID": "4"
      }
    ]
  },
  "deviceName": "RESTclient",
  "ssoEnabled": true
}

```

Example 6-41 Execute Close Application - Response

```

{
  "fs_P0115_W0115A": {
    "title": "Phone Numbers",
    "data": {
      ...condensed...
    },
    "summary": {
      "records": 4,
      "moreRecords": false
    }
  },
  "errors": [],
  "warnings": []
},
"stackId": 0,
"stateId": 0,
"rid": "",
"currentApp": "P0115_W0115A_ZJDE0001",

```

```

    "sysErrors": []
}

```

When the stack is closed, the stack ID, state ID, and rid are cleared, indicating you can no longer interact with that stack.

6.10.4.5 Mobile Application Example

The application manages the alternate addresses for the 0 contact of an address number. It lists the first 10 "E" type records in the Address Book, and allows the user to select a record where they can add, update, or delete alternate addresses. Notice that the framework handles all of the stack maintenance for you within the ApplicationStack object. You do not need to manually manage the rid, stack ID, or state ID in your mobile application. The API does that for you.

An ApplicationStack variable is defined in the main DC class. You should define one of these for each stack you want to maintain. For example:

```
ApplicationStack appStackAddress = new ApplicationStack();
```

When the mobile application opens, it retrieves the Address Book records and saves the response in a data control variable for the P01012_W01102B form.

```

public void getABRecords() {

    // Retrieve a list of customers from the P03013_W03013A form.
    LoginResponse lr = getLoginResponse();

    FormRequest formRequest = new FormRequest();
    formRequest.setReturnControlIDs("1");
    formRequest.setFormName("P01012_W01012B");
    formRequest.setReturnControlIDs("54|1[19,20]");
    formRequest.setFormServiceAction("R");
    formRequest.setMaxPageSize("10");
    FSREvent findFSREvent = new FSREvent();
    String name = lr.getUsername();
    findFSREvent.setFieldValue("54", "E");
    findFSREvent.doControlAction("15"); // Find button
    formRequest.addFSREvent(findFSREvent);

    try {
        //open P01012_W01012B
        String response = appStackAddress.open(formRequest);

        p01012_W01012B_FormParent =
            (P01012_W01012B_
            FormParent)JSONBeanSerializationHelper.fromJSON(P01012_W01012B_FormParent.class,
            response);

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        throw new AdfException(e.getMessage(), AdfException.ERROR);
    }

}

```

When an Address Book record is selected in the mobile application, the mobile application performs a service call to perform the row exit on the open application. It checks to make sure the interconnect happened, and then performs a second interconnect to Work with Alternate Addresses form. Finally, it checks that it is on the addresses form and then deserializes the response to the data control variable for the addresses form. The Address Book application has a multi-select grid, and taking a row exit when multiple records are selected causes issues; therefore the action is to deselect all, and then select the correct one.

```
public String goToAltAddress() {

    String nav = "";
    ActionRequest getAddressAction = new ActionRequest();
    FSREvent selectEvent = new FSREvent();
    try {
        getAddressAction.setFormOID("W01012B");
        Integer rowIndex =
(Integer)AdfmfJavaUtilities.evaluateELEExpression("#{pageFlowScope.selectedABIndex}
");
        if (rowIndex != null) {
            selectEvent.unselectAllGridRows("1");
            //select the row they chose
            selectEvent.selectRow("1", rowIndex.intValue());
            //deselect the previous row (W01012B has multi select grid and we are only viewing
            one at a time
            selectEvent.doControlAction("67"); //whos who exit
            getAddressAction.addFSREvent(selectEvent);
            String response =
appStackAddress.executeActions(getAddressAction);
            if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A")) {
                selectWhosWho();
                if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")) {
                    nav = "to_Addresses";
                }
            }
        }
    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    } catch (Exception e) {
        throw new AdfException(e.getMessage(), AdfException.ERROR);
    }

}

public void selectWhosWho() {

    if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A")) {

        try {
            //send another request to go to alt address
            ActionRequest selectAction = new ActionRequest();
            FSREvent findEvent = new FSREvent();
            //findAction.setReturnControlIDs("32|7|1[28,29,66]");
            selectAction.setFormOID("W0111A");
            findEvent.selectRow("1", 0); //first whos who row
            findEvent.doControlAction("148");
            selectAction.addFSREvent(findEvent);
            String response = appStackAddress.executeActions(selectAction);
```

```

        if
        (appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")) {

            p01111_W01111E_FormParent =
                (P01111_W01111E_
                FormParent)JSONBeanSerializationHelper.fromJSON(P01111_W01111E_FormParent.class,
                response);

        }

        } catch (JDERestServiceException e) {
            JDERestServiceProvider.handleServiceException(e);
        } catch (Exception e) {
            throw new AdfException(e.getMessage(), AdfException.ERROR);
        }
    }

    } new AdfException(e.getMessage(), AdfException.ERROR);
    }

    return nav;
}

```

The user can then navigate to an individual address to change or delete it. They can also add a new one.

This method handles saving an existing address or adding a new one. It also handles any errors that were returned from the service.

```

public String saveAddress() {
    String nav = "";
    String formMode =
        (String)AdfmfJavaUtilities.evaluateELEExpression("#{pageFlowScope.formMode}");
    ActionRequest addAction = new ActionRequest();
    FSREvent saveEvent = new FSREvent();
    Integer rowIndex = null;

    if (formMode.equals("A") && this.getArrayError().size() > 0) {
        //there was an error on a previously added one, it may be at the form
        level so we can't easily identify the record to fix, close the form and open it
        again
        cancelAddress();
    }

    //clear errors each time to try again
    setArrayError(new ArrayList());

    try {

        addAction.setFormOID("W01111E");
        GridAction gridAction = new GridAction();

        if (formMode.equals("A")) {
            GridRowInsertEvent gri = new GridRowInsertEvent();
            //set the column values
            gri.setGridColumnValue("34", singleAddressRow.getSAddressType_
            34().getValue());
            gri.setGridColumnValue("26", singleAddressRow.getSAddressLine1_

```

```

26().getValue());
        gri.setGridColumnValue("22", singleAddressRow.getSCity_
22().getValue());
        gri.setGridColumnValueDate("18", singleAddressRow.getDtBeginDate_
18().getDate());

        //add the row
        gridAction.insertGridRow("1", gri);
    } else {
        if (rowIndex == null) {
            rowIndex =

(Integer)AdmfJavaUtilities.evaluateELExpression("#{pageFlowScope.selectedAddressI
ndex}");
        }
        GridRowUpdateEvent gru = new GridRowUpdateEvent();
        //set the column values
        gru.setGridColumnValue("34", singleAddressRow.getSAddressType_
34().getValue());
        gru.setGridColumnValue("26", singleAddressRow.getSAddressLine1_
26().getValue());
        gru.setGridColumnValue("22", singleAddressRow.getSCity_
22().getValue());
        gru.setGridColumnValueDate("18", singleAddressRow.getDtBeginDate_
18().getDate());

        //update the row
        gridAction.updateGridRow("1", rowIndex.intValue(), gru);
    }
    //add the grid action to the events
    saveEvent.addGridAction(gridAction);

    saveEvent.doControlAction("12"); //OK
    //addAction.setReturnControlIDs("32|7|1[28,29,66]");
    addAction.addFSREvent(saveEvent);
    String response = appStackAddress.executeActions(addAction);

    //after save it returns to whos who, open it again to get final list

    if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A")) {
        selectWhosWho();
        if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")) {
            nav = "__back";
        }
    } else {
        if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")) {

            P01111_W01111E_FormParent tempW0111E =
                (P01111_W01111E_
FormParent)JSONBeanSerializationHelper.fromJSON(P01111_W01111E_FormParent.class,
response);

            //check for errors on the form
            if (tempW0111E.getFs_P01111_W01111E() != null &&
                tempW0111E.getFs_P01111_W01111E().getErrors() != null &&

```

```

        tempW0111E.getFs_P01111_W01111E().getErrors().length > 0)
    {
        //show error
        addErrors(tempW0111E.getFs_P01111_W01111E().getErrors());
    }
}

}

} catch (JDERestServiceException e) {
    JDERestServiceProvider.handleServiceException(e);
} catch (Exception e) {
    throw new AdfException(e.getMessage(), AdfException.ERROR);
}

return nav;
}

```

The following sample code shows an example of the method to delete the address. After deleting and saving, the EnterpriseOne form closes, so you have to open it again with the selectWhosWho method.

```

public String deleteAddress(int key) {

    String nav = "";

    ActionRequest deleteActionRequest = new ActionRequest();
    FSREvent deleteEvent = new FSREvent();
    deleteActionRequest.setFormOID("W01111E");
    //deleteActionRequest.setReturnControlIDs("32|7|1[28,29,66]");
    deleteEvent.selectRow("1", key);
    //press Delete button
    deleteEvent.doControlAction("41");
    //press OK button
    deleteEvent.doControlAction("12");
    //add the FSR event to the request
    deleteActionRequest.addFSREvent(deleteEvent);

    try {
        String response = appStackAddress.executeActions(deleteActionRequest);

        if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A")) {
            selectWhosWho();
            if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")) {
                nav = "__back";
            } else {
                //errors on delete?
            }

        } else if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01012_W01012B")) {

            nav = "to_AB";
        }

    } catch (JDERestServiceException e) {
        JDERestServiceProvider.handleServiceException(e);
    }
}

```

```

        } catch (Exception e) {
            AdfException adfe = new AdfException(e.getMessage(),
            AdfException.ERROR);
            throw adfe;
        }

        return nav;
    }

```

6.10.5 ApplicationStack Methods

The following table describes the ApplicationStack methods:

Modifier and Type	Method	Description
String	open(FormRequest formRequest)	Opens a form, allowing further interactions with subsequent service calls.
String	executeActions(ActionRequest actionRequest)	Executes actions on an already open form.
String	close(ActionRequest actionRequest)	Closes all applications in the application stack, after executing the actions requested
String	close ()	Closes all applications in the application stack, without doing any further actions
void	setOverrideDefaultFeature(String overrideDefaultFeature)	Use this to set the default feature if you are using an ApplicationStack from a re-usable component. Similar to how you have to override the default feature for a form service call. Do this call before other requests.
ApplicationStackResponse	getLastAppStackResponse()	Returns the last response which includes the stack ID, state ID, rid and system errors.

6.10.6 ApplicationStackResponse Methods

The following table describes the ApplicationStackResponse methods:

Modifier and Type	Method	Description
boolean	checkSuccess (String appForm)	Returns true if the last response was for the appForm (for example P01012_W01012B) if the form does not match it returns false.

Creating a Sample Mobile Application

This appendix guides you through the steps to create a sample EnterpriseOne mobile enterprise application using Oracle Mobile Application Framework (MAF).

This appendix contains the following topics:

- [Section A.1, "Before You Begin"](#)
- [Section A.2, "Creating the Sample Address Book Mobile Application"](#)
- [Section A.3, "Using the JDE Mobile Helpers"](#)
- [Section A.4, "Connecting to the EnterpriseOne Application Interface Services \(AIS\) Server"](#)
- [Section A.5, "AIS Client Class Generator"](#)
- [Section A.6, "Reading EnterpriseOne Data"](#)
- [Section A.7, "Implementing Filter Fields"](#)
- [Section A.8, "Page Navigation and Getting More Details"](#)
- [Section A.9, "Updating Data in EnterpriseOne"](#)
- [Section A.10, "Device Integration"](#)

A.1 Before You Begin

Before following the instructions in this section, see [Section 3.2, "Prerequisites"](#) in this guide to ensure you have completed the required prerequisites.

The prerequisites include the `JDEMobileSampleApplication.zip`. This zip file contains the components for running the same sample mobile application that this appendix describes how to create. You can use this sample for comparison purposes. To run this sample application, you must configure it with the `JDEMobileFramework.jar` and `Login.jar` as described in [Section A.3.1](#) and [Section A.3.2](#).

Remember, the files in the `JDEMobileSampleApplication.zip` are intended for reference purposes only.

Important: To run the sample mobile application, you must make sure that the `JDEMobileSampleApplication` and the Oracle MAF extension are from the same `JDE_Mobile_Framework` package. In other words, if you installed the 2.0.1 MAF extension from the `JDE_Mobile_Framework_2.0.1` package, then you will need to run the `JDEMobileSampleApplication` that is included in that package.

To verify the version of the Oracle MAF extension:

1. In JDeveloper click the Help menu, About, and then click the Extensions tab.
2. In the "find" field, type `maf` and then press Enter. Scroll to the right to check the version.

A.2 Creating the Sample Address Book Mobile Application

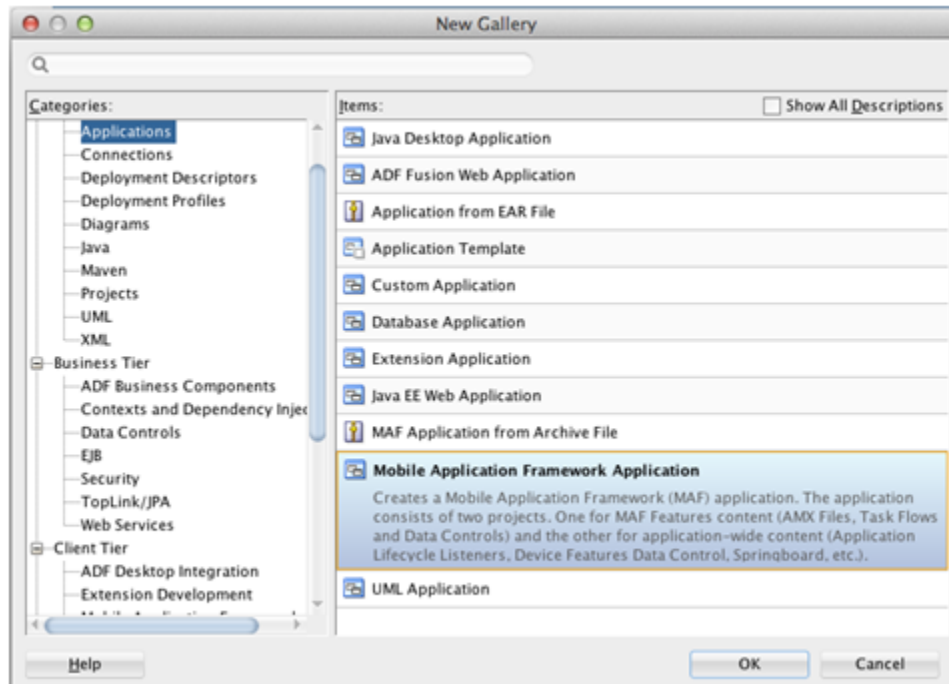
This section contains the following topics:

- Section A.2.1, "Creating a New Mobile MAF Application"
- Section A.2.2, "Running the Mobile Application in the Simulator"

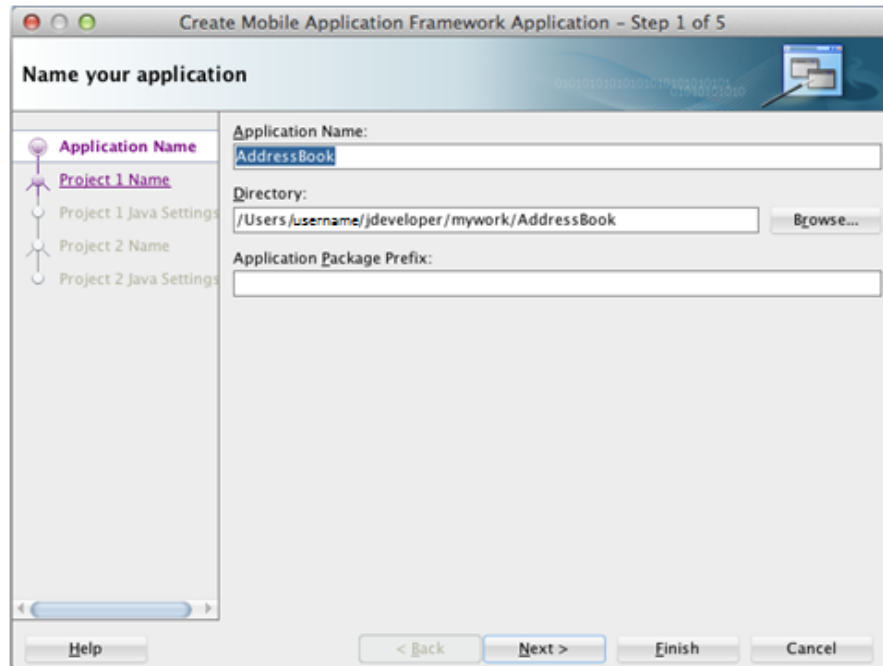
A.2.1 Creating a New Mobile MAF Application

This section describes how to create a new sample mobile application entitled Address Book.

1. Launch JDeveloper.
2. Select the **File** menu > **New** > **Application**.

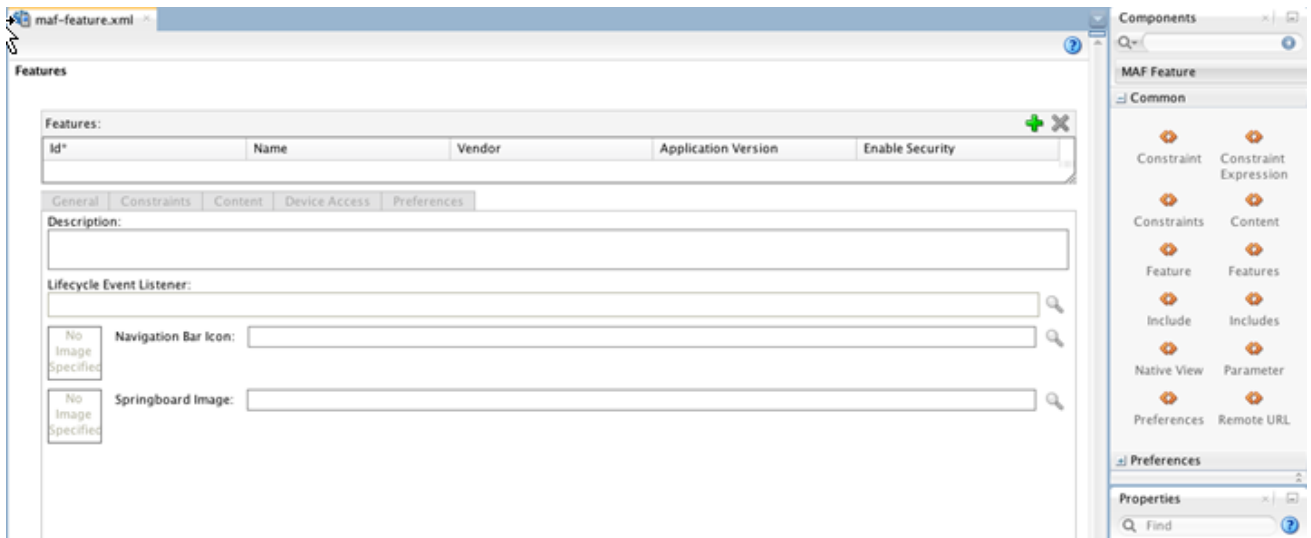


3. In the New Gallery, select **Mobile Application Framework Application** and click **OK**.

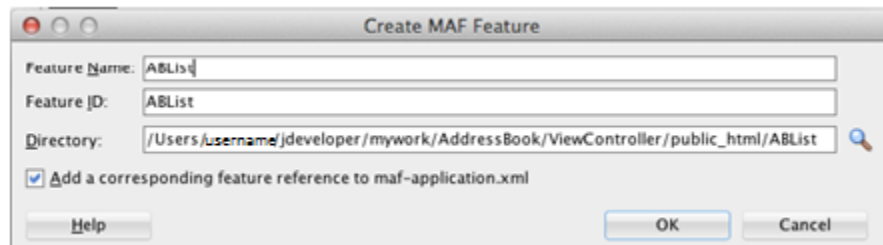


- On "Name your application," enter AddressBook in the **Application Name** field, and then click **Finish** to skip the rest of the steps.

The application displays the Features list.

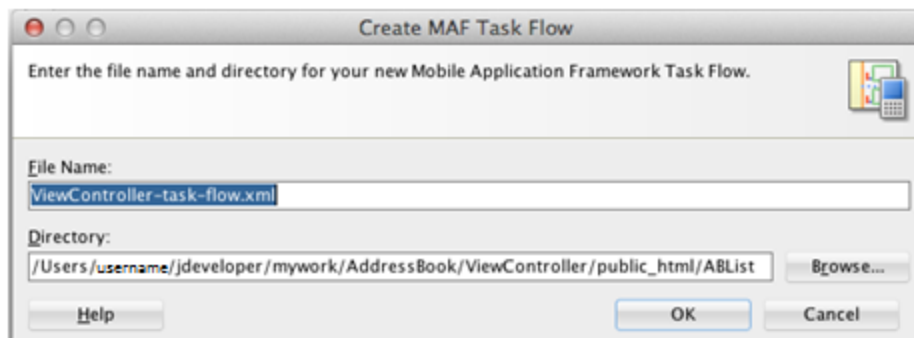


- In the Features section, click the **green plus sign** to add a new feature to the list.



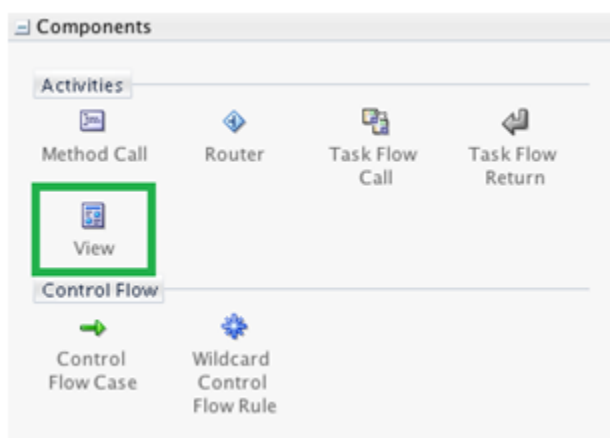
- On Create MAF Feature, enter ABList in the **Feature Name** and **Feature ID** fields.

7. Click **OK**.
8. Select the **Content** tab.
9. Change the Content Type to MAF Task Flow.
10. Click the add button next to the **File** field.

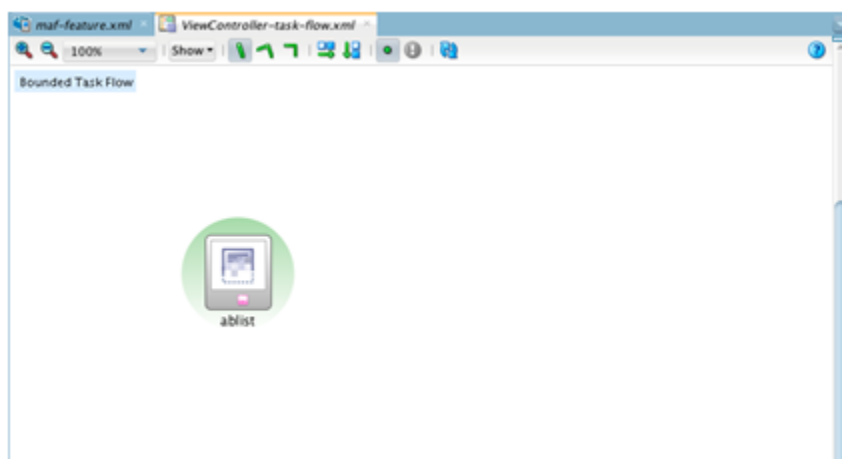


11. On Create MAF Task Flow, use the default name in the **File Name** field for the task flow and click **OK**.

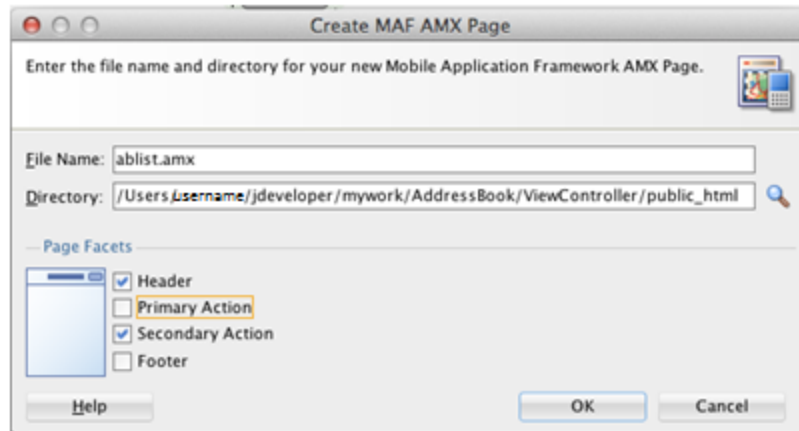
JDeveloper displays the new task flow.



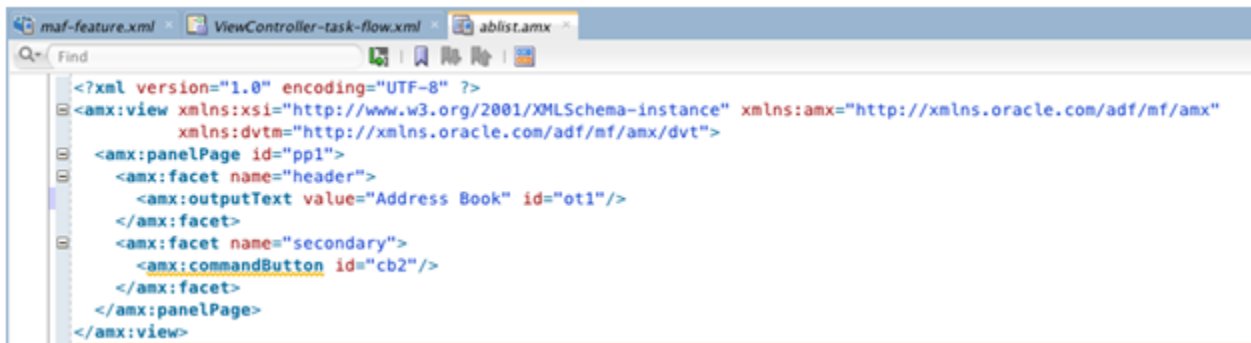
12. In the Components window on the right, click the **View** button to add a new view to the task flow.
13. Name the new view ablist.



14. On the ViewController-task-flow.xml tab, double-click the new **ablist** view element to create the view page.



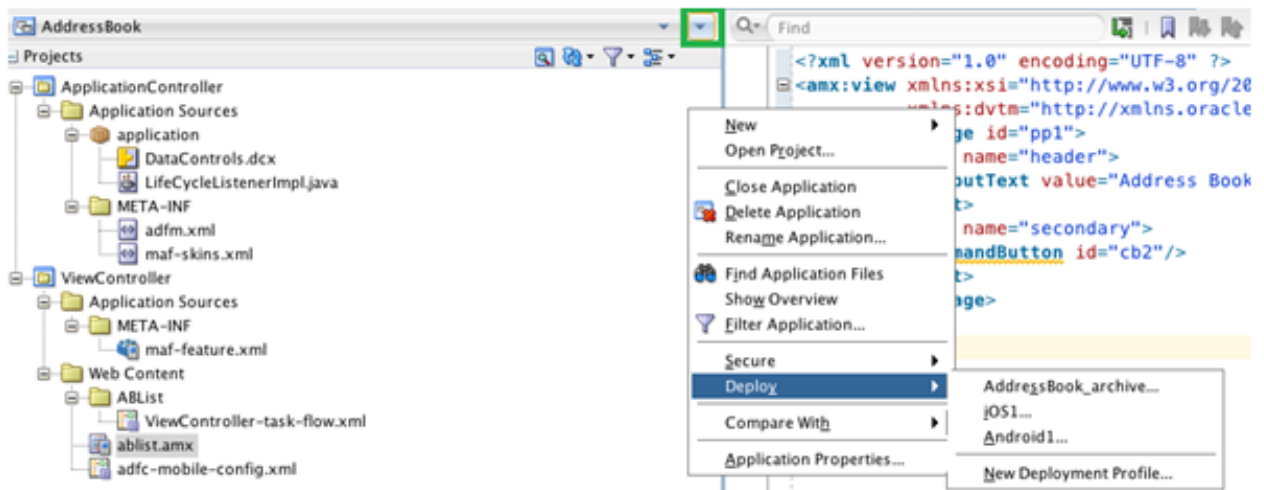
The new page opens automatically.



15. On the ablist.amx tab, change the outputText value to "Address Book" as shown in the preceding example. This is the title for the first page of the sample mobile application.
16. Save your application.

A.2.2 Running the Mobile Application in the Simulator

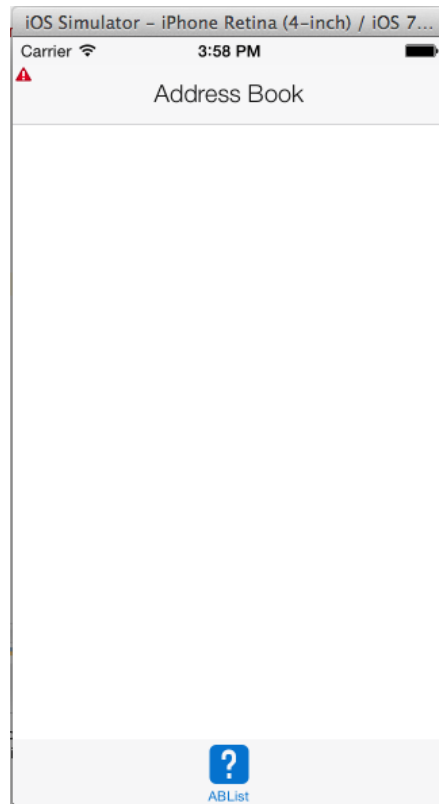
To run the mobile application in the simulator:



1. From the application drop-down, select **Deploy > iOS1**.
2. On Deployment Action, select **Deploy application to simulator** and click **Finish**.
Deployment details can be viewed in the deployment log. Wait for it to finish.
The simulator displays the application on the second page of the home screen.



3. Click the application icon to launch it.
The page that you created appears with the header text "Address Book," as shown in the following example:



If your mobile application failed to build or deploy at this point, please refer to the *Oracle Fusion Middleware Developing Mobile Applications with Oracle Mobile Application Framework* documentation for more information:

<http://docs.oracle.com/middleware/mobile200/mobile/develop/index.html>

A.3 Using the JDE Mobile Helpers

This section describes how to incorporate the JDE Mobile Helpers into the sample mobile application.

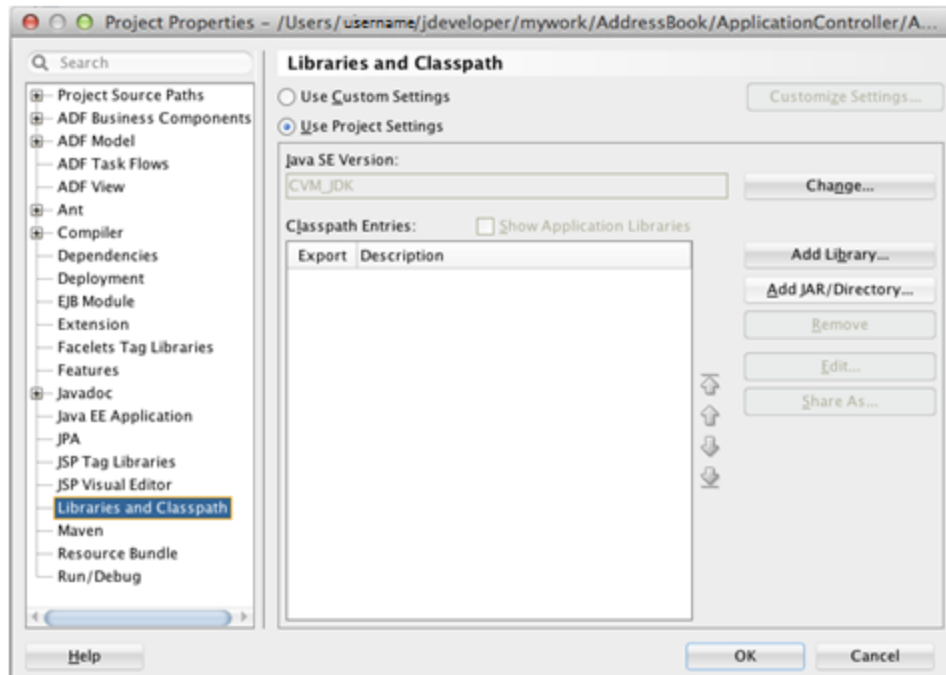
See [Section 2.3.1, "JDE Mobile Helpers"](#) for more information about the JDE Mobile Helpers.

A.3.1 Including the JDEMobileFramework.jar

The JDEMobileFramework.jar provides a set of classes and API methods that enable the mobile application to manage (create, read, update, delete) data in EnterpriseOne through REST services.

To include the JDEMobileFramework.jar:

1. In JDeveloper, in the Projects panel, right-click the **ApplicationController** project and select **Project Properties**.
2. On Project Properties, select **Libraries and Classpath**.



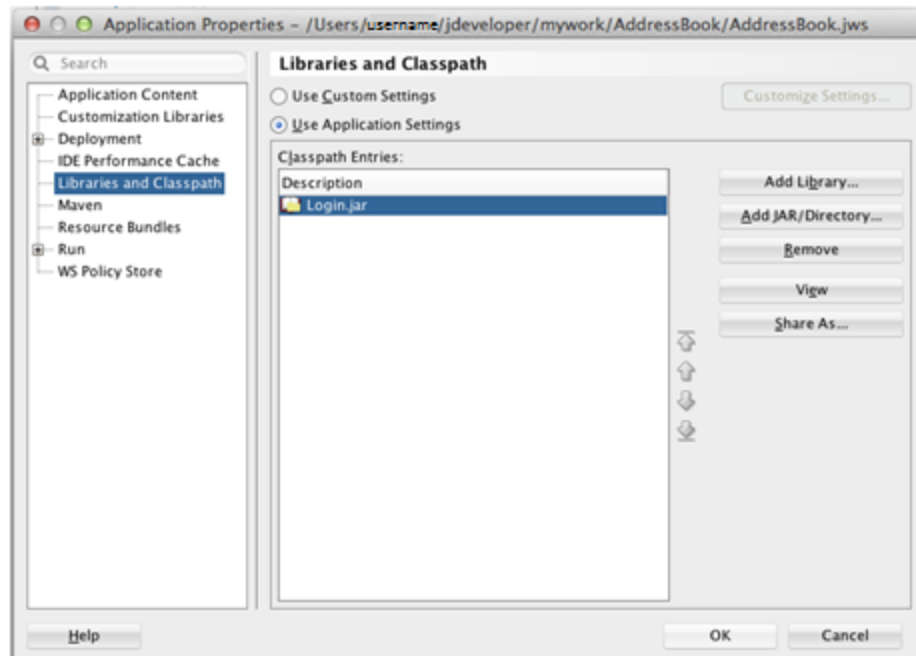
3. Click the **Add JAR/Directory** button and select the `JDEMobileFramework.jar` that you downloaded to your local file system, and click **Open**.
If you do not have `JDEMobileFramework.jar`, see [Prerequisites](#) in this guide.
4. Click **OK** to save the properties. If you receive an error message, ignore it and click **Cancel**.

A.3.2 Including the Login.jar

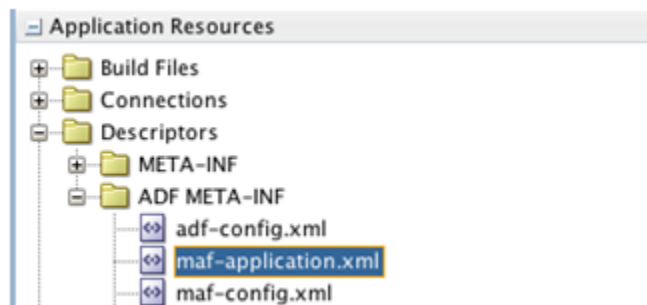
The `Login.jar` provides a configuration page, login page, and a springboard. The springboard contains links to Legal Terms or the End User License Agreement (EULA), About, and Logout.

To include the `Login.jar`:

1. In JDeveloper, click the Applications drop-down menu and select **Application Properties**.



2. On Application Properties, select **Libraries and Classpath**, and then click the **Add JAR/Directory** button to add the Login.jar.
3. Click **OK** to save the properties.

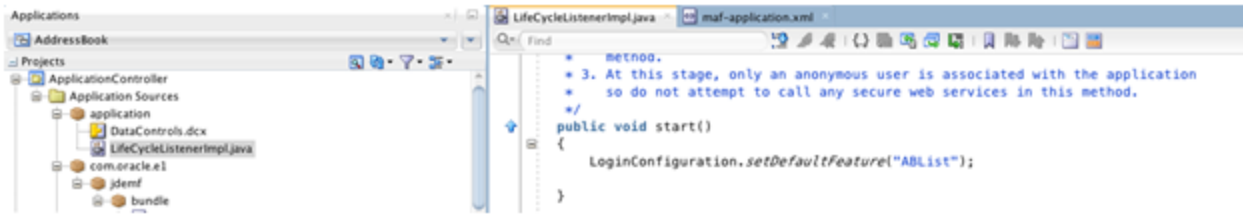


4. In the Application Resources section, expand **Descriptors > ADF META-INF**, and double-click **maf-application.xml** to open it.
5. Click the Feature References tab, and then click the **green plus sign** to add a feature reference.
6. Click the **Id** drop-down menu and select **com.oracle.e1.jdemf.login**. If you receive an error message, ignore it and click **Cancel**.



7. Use the blue arrow on the right to move the login feature to the top of the list.
8. On the Application tab, clear the **Show Navigation Bar on Application Launch** check box.

- In the Projects panel, expand **ApplicationController > Application Sources > application** and double-click **LifeCycleListener.java** to open it.



- On the LifeCycleListener.java tab, place the following line of code in the existing start() method, as shown in the preceding example:

```
LoginConfiguration.setDefaultFeature("ABList");
```

This code directs the Login feature to navigate to the appropriate page after a successful login.

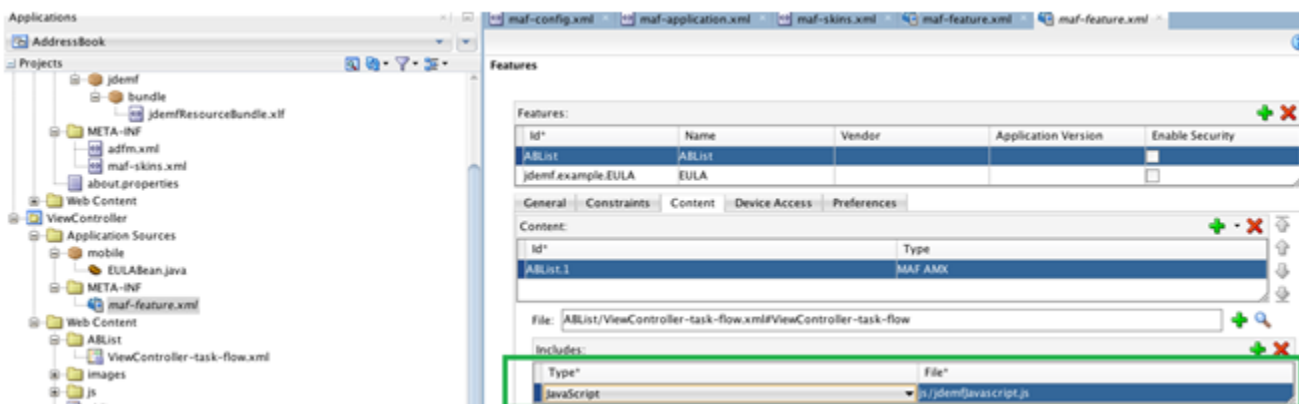
- Save the LifeCycleListener.java.

A.3.3 Including the Javascript and CSS

The Javascript and CSS files are dependencies of the JDEMobileFramework.jar and the Login.jar. The Javascript provides an animated icon to show that the mobile application is processing while making service calls. The CSS provides an extension to the styling skin provided by Oracle MAF. It enables you to make adjustments to the style of the configuration, login, and springboard pages of your mobile application.

To include the Javascript and CSS:

- In the file system, copy the js folder provided in the JDEMobileFramework zip file, and save it to the ViewController/public_html directory.
- In the file system, copy the css folder provided in the JDEMobileFramework zip file, and save it to the ApplicationController/public_html directory.
- In the Projects panel in JDeveloper, expand **ViewController > Application Sources > META-INF**, and double-click the **maf-feature.xml** to open it.

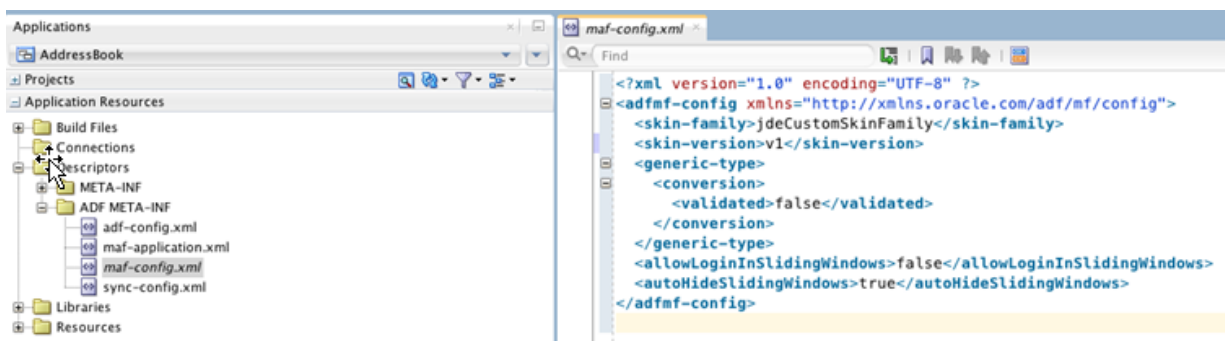


- Select the **ABList** feature, click the Content tab, and click the **plus sign** in the Includes section to add the `jdemajavascript.js` file. This file is located in js folder.
- Under **ApplicationController > Application Sources > META-INF**, open the `maf-skins.xml` and copy the following code into the file:

Note: Remove any code lines that might exist before you copy the following code into the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<admf-fskins xmlns="http://xmlns.oracle.com/adf/mf/skin">
  <skin id="s1">
    <family>jdeCustomSkinFamily</family>
    <id>jdeCustomSkinFamily-v1.Android</id>
    <extends>jdeCustomSkinFamily</extends>
    <style-sheet-name>css/jdemfCustomStyles-Android.css</style-sheet-name>
    <version>
      <name>v1</name>
      <default>true</default>
    </version>
  </skin>
  <skin id="s2">
    <family>jdeCustomSkinFamily</family>
    <id>jdeCustomSkinFamily-v1.iOS</id>
    <extends>jdeCustomSkinFamily</extends>
    <style-sheet-name>css/jdemfCustomStyles-iOS.css</style-sheet-name>
    <version>
      <name>v1</name>
      <default>true</default>
    </version>
  </skin>
  <skin id="s3">
    <family>jdeCustomSkinFamily</family>
    <id>jdeCustomSkinFamily</id>
    <extends>mobileAlta-v1.1</extends>
    <style-sheet-name>css/jdemfCustomStyles-Base.css</style-sheet-name>
    <version>
      <name>v1</name>
      <default>true</default>
    </version>
  </skin>
</admf-fskins>
```

6. In the Application Resources panel, expand **Descriptors** > **ADF META-INF**, and open the **maf-config.xml**.
7. Enter `jdeCustomSkinFamily` for the `<skin-family>` and enter `v1` for the `<skin-version>`, as show in the following example:



A.3.4 Including the Resource Bundle

The Resource Bundle contains text resources for the pages provided in the Login.jar.

To include the Resource Bundle:

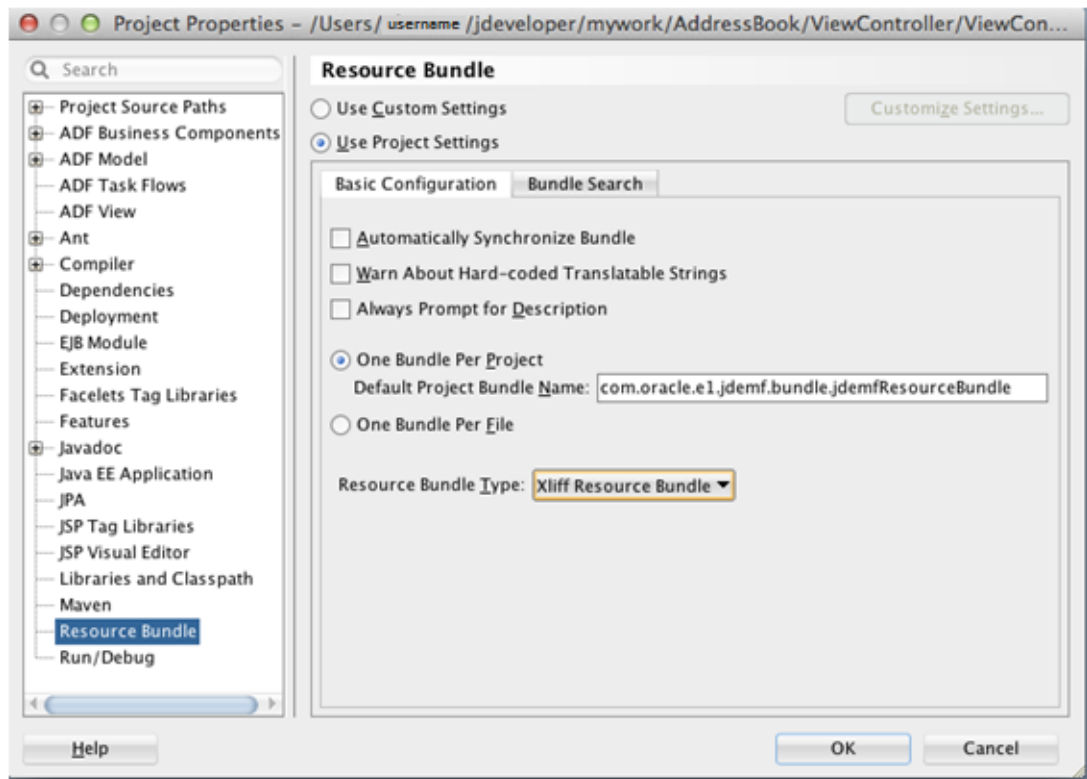
1. Copy the **jdemfResourceBundle.xlf** file provided in the JDEMobileFramework zip file and save it to the following directory:

```
ApplicationController/src/com/oracle/e1/jdemf/bundle
```

You must add each directory in the path if it does not exist.

Note: The ViewController project is dependent on the ApplicationController project. Adding the bundle to the ApplicationController project will also make it available to the ViewController project.

2. In JDeveloper, click the "refresh" button for the bundle to appear under com.oracle.e1.
3. Right-click **ViewController** and select **Project Properties**.
4. On Project Properties, select **Resource Bundle**.



5. Set the Default Project Bundle Name to:
com.oracle.e1.jdemf.bundle.jdemfResourceBundle
6. Click **OK** to save the properties. If a warning message appears, ignore it and click **Ok** or **Cancel** to proceed.

A.3.5 Including Logo Images

You should include a logo image for the pages in the Login.jar. If you do not insert one, a question mark (?) appears in place of the image. Oracle provides the following

image files as examples of the recommended size, quality, and color of images that you should use in your mobile application:

- `jde_transparent_no_jde_small.png`
This is a 242 x 87 transparent image for the login page, intended for a white background.
- `jde_transparent_springboard.png`
This is a 242 x 87 transparent image for the springboard, intended for a dark background.

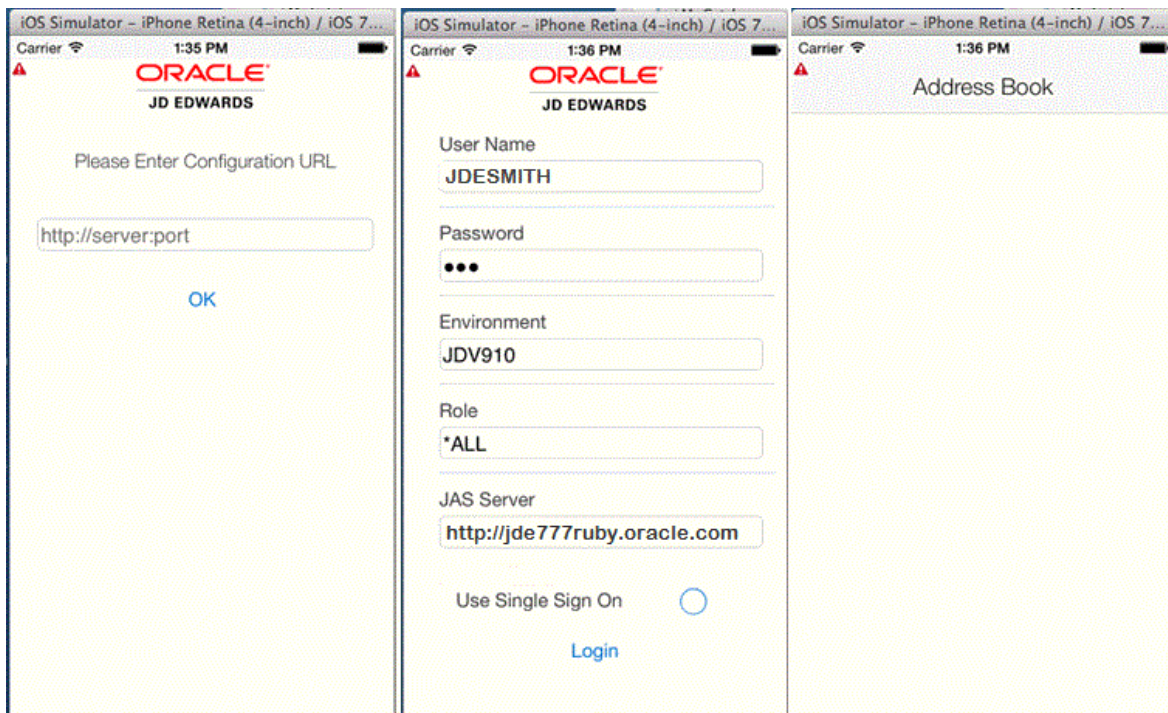
You can use any image as long as it has the same dimensions as the example images. You can use the same name for the images and place them in the same location as instructed in the following steps.

Important: Do not use the logo image examples provided by Oracle; they are provided for example purposes only.

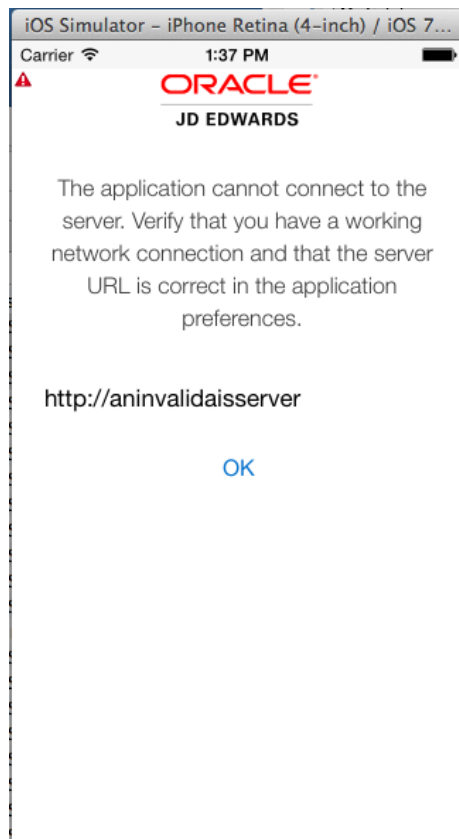
To include logo images:

1. In the file system, copy the `jde_transparent_no_jde_small.png` file to the `ViewController/public_html/images` directory.
2. Copy the `jde_transparent_springboard.png` to the `ViewController/public_html/images` directory.
3. In JDeveloper, deploy the application.

You should see the configuration screen and login screen. After logging in, it should take you to your ABList feature with the page with the "Address Book" title.



The Login.jar also provides information screens if configuration or connection issues arise during login, for example:



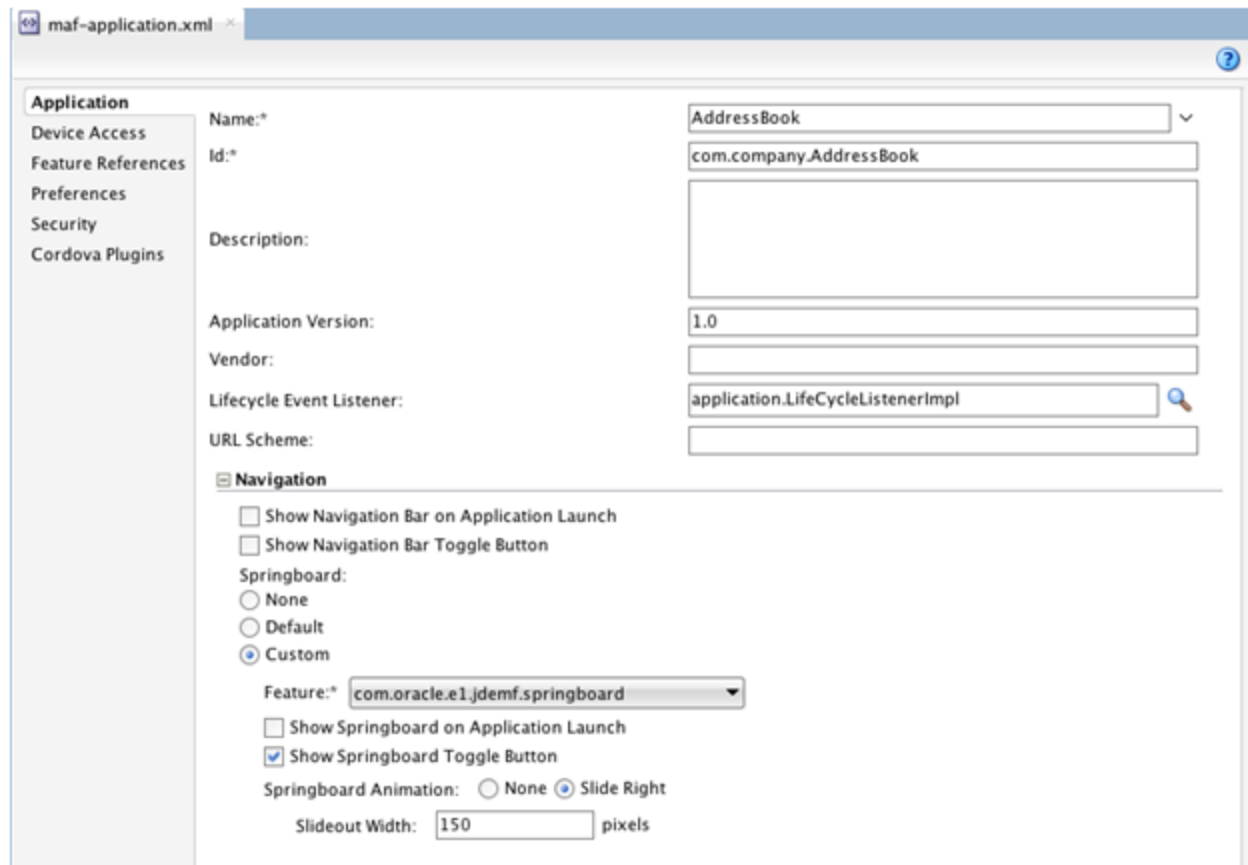
Note: If you receive an error, under the Application Resources panel in JDeveloper, double-click `maf-application.xml`. In the `maf-application.xml` tab, make sure the Files and Network options are selected.

A.3.6 Enabling the Custom Springboard

The springboard provided in the `Login.jar` shows links to the About page, the EULA, and the Logout.

To enable the springboard:

1. Open the `maf-application.xml`.
2. In the Navigation section on the Application tab, clear the **Show Navigation Bar on Application Launch** check box.
3. For Springboard, select the **Custom** option
4. In the Feature drop-down menu, select `com.oracle.e1.jdemf.springboard`.



Application

Name:* AddressBook

Id:* com.company.AddressBook

Description:

Application Version: 1.0

Vendor:

Lifecycle Event Listener: application.LifecycleListenerImpl

URL Scheme:

Navigation

Show Navigation Bar on Application Launch

Show Navigation Bar Toggle Button

Springboard:

None

Default

Custom

Feature:* com.oracle.e1.jdemf.springboard

Show Springboard on Application Launch

Show Springboard Toggle Button

Springboard Animation: None Slide Right

Slideout Width: 150 pixels

5. Select the **Show Springboard Toggle Button** check box.
6. For Springboard Animation, select the **Slide** option.
7. In the Slideout Width field, enter 150.
8. Clear any other check boxes and then click the **Feature References** tab.

Feature References:		
Id*	Show on Navigation Bar	Show on Springboard
com.oracle.e1.jdemf.login	<default> (true)	<default> (true)
ABList	<default> (true)	<default> (true)
com.oracle.e1.jdemf.springboard	false	false

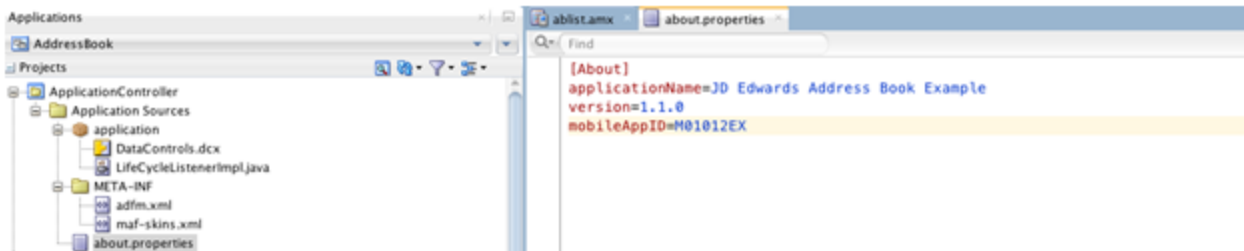
9. In the Feature References section, click the **green plus sign** to add the springboard feature, and then select **false** for both the "Show on Navigation Bar" and "Show on Springboard" options.

A.3.7 Including the about.properties

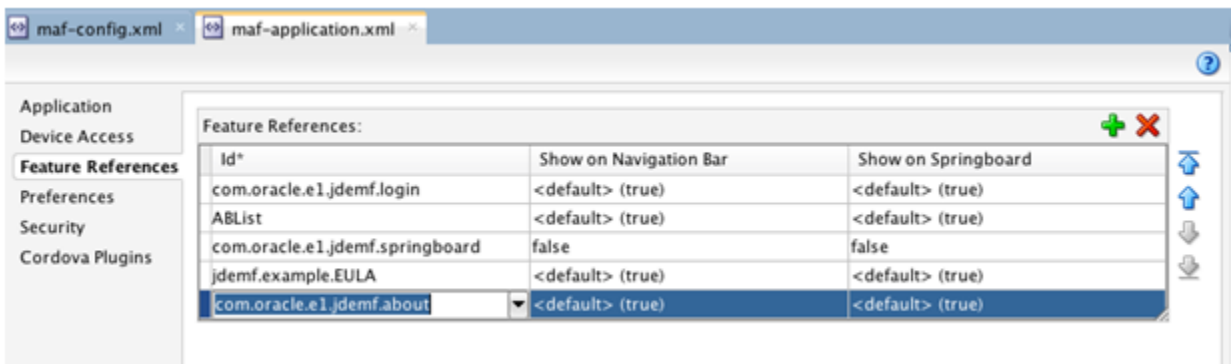
The about.properties enables you to configure information displayed on the About page, including the application name, application version, and the application ID (which is used by EnterpriseOne application security for authorizing user access to the mobile application). If you enable the springboard, you should provide these values so that they appear on the About page.

To include the about.properties:

1. Copy the **about.properties** file provided in the JDEMobileFramework zip file and save it in the ApplicationController/src directory.
2. In the Projects panel, expand **Application Controller > Application Sources**, and double-click **about.properties** to open it.



3. Modify the about.properties file with information for your application, as shown in the preceding example.
4. Click the **Feature References** tab.



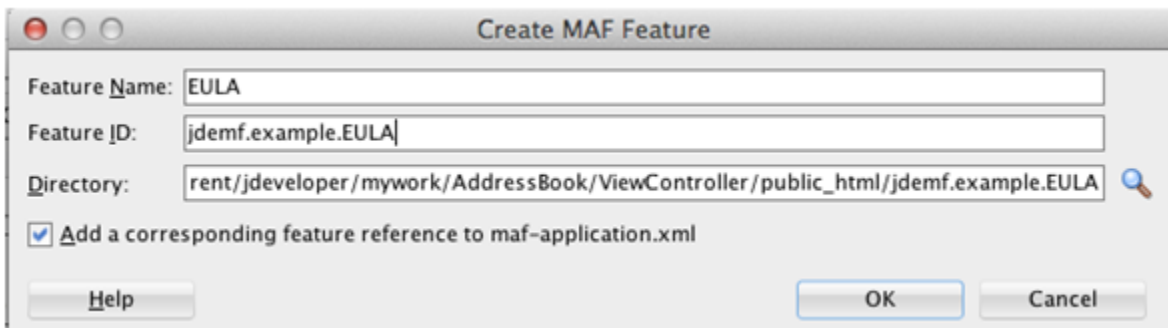
5. In the Feature References section, click the **green plus sign** to add the com.oracle.e1.jdemf.about feature.

A.3.8 Including an End User License Agreement (EULA)

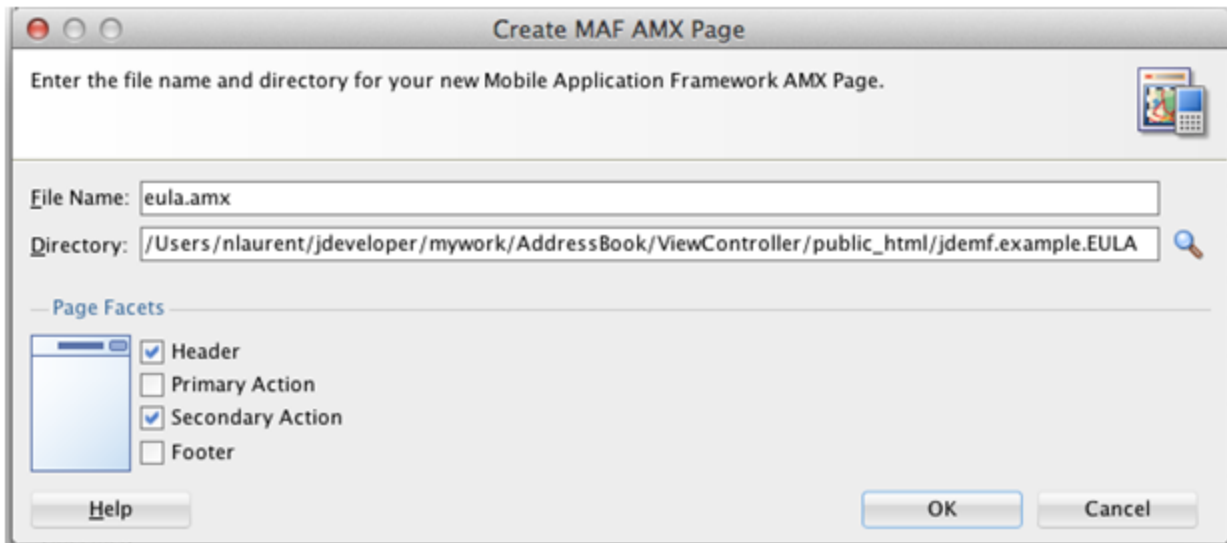
If you enable the springboard, you should include a EULA feature and EULA page so the Legal Terms link has a page to access. You can provide a license agreement for your application here.

To include a EULA:

1. Double-click the **maf-feature.xml**.
2. In the Features section, double-click the **green plus sign** to add a feature for the EULA.



3. On Create MAF Feature, enter EULA in the **Feature Name** field, give it a unique id, and then click **OK**.
4. After you create the EULA feature, select the **Content** tab.
5. Click the **green plus sign** next to the File field.



6. On Create MAF AMX Page, enter a name for the page in the **File Name** field.
7. Clear the **Primary Action** check box, and then click **OK**.
8. In the new eula.amx page, you can design your EULA page as needed, as shown in [Example A-1](#).

Example A-1 Using the Verbatim Tag to Include HTML

The following sample code is an example of a page that uses the verbatim tag to include HTML. It also uses the configured resource bundle (loadBundle tag).

```
<?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:loadBundle basename="com.oracle.e1.jdemf.bundle.jdemfResourceBundle"
var="jdeBundle" id="lb1"/>
<amx:panelPage id="pp1">
<amx:facet name="header">
<amx:outputText value="{jdeBundle.LEGAL_TERMS}" id="ot1"/>
</amx:facet>
<amx:facet name="secondary">
<amx:commandButton text="{jdeBundle.DONE}" id="cb1"
actionListener="{eulaBean.doneEULA}"/>
</amx:facet>
<amx:verbatim id="v1">

<![CDATA[

<p class="MsoNormal"
style="margin-bottom: 0.0001pt; text-align: center; line-height:
normal;"><b><span
style="font-size: 12pt; font-family:
'Arial','sans-serif'; color: black;">MY END
```

```

USER LICENSE
AGREEMENT</span></b>
</p>
    Legal...

    ]]>
    </amx:verbatim>
</amx:panelPage>
</amx:view>

```

9. Save the EULA page.
10. In the Projects panel, right-click **ViewController** and select **Project Properties**.
11. On Project Properties, select **Dependencies**, select the **ApplicationController.jpr**, and click the **green plus sign** button.
12. On Edit Dependencies, click the **Build Output** check box, click **OK**, and then click **OK** again to close the Project Properties.
13. Add code for the Done button, which returns the user to the application after viewing the EULA. To do so:
 - a. In the Projects panel, expand **ViewController > Web Content**, and double-click the **eula.amx**.
 - b. In the eula.amx code, highlight the **commandButton** element.
 - c. In the Button - Done - Properties window on the right, hover over the **Action Listener** field and click the "gear" button, and select **Edit**.
 - d. On "Edit Property: Action Listener," click the **New** button to create a new managed bean.
 - e. On Create Managed Bean, complete the following fields and then click **OK**:
 - Bean Name**: eulaBean
 - Class Name**: EulaBean
 - Scope**: pageFlow
 - f. Click the **New** button next to the **Method** field to create a new method for the bean.
 - g. On Create Method, enter doneEULA in the **Method Name** field and click **OK**.
 - h. On "Edit Property: Action Listener," click **OK** to accept the new bean and method.
This adds the new bean method to the action listener.
 - i. In the code, right-click the doneEula method name and select **Go to Declaration**.
 - j. In the new method, replace the `//add event code here... comment` with this line of code:

```
JDEmUtilities.goToDefaultFeature();
```

Your new method should look like this:

```

package mobile;

import ...;

public class EulaBean {
    public EulaBean() {
    }

    public void doneEula(ActionEvent actionEvent) {
        JDEmfUtilities.goToDefaultFeature();
    }
}

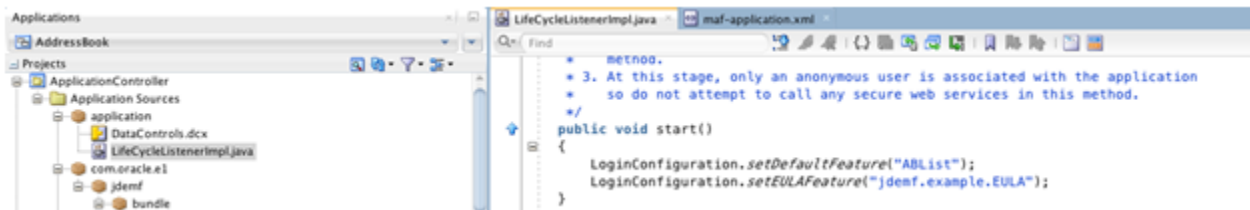
```

14. In the Projects panel, expand **Application Controller > Application Sources > application**, and double-click the **LifeCycleListenerImpl.java**.

15. Add the following line of code to the start() method to set the EULA Feature, and make sure to use the feature ID that you used to create the feature. If you do not remember it, you can find the ID in the maf-feature.xml.

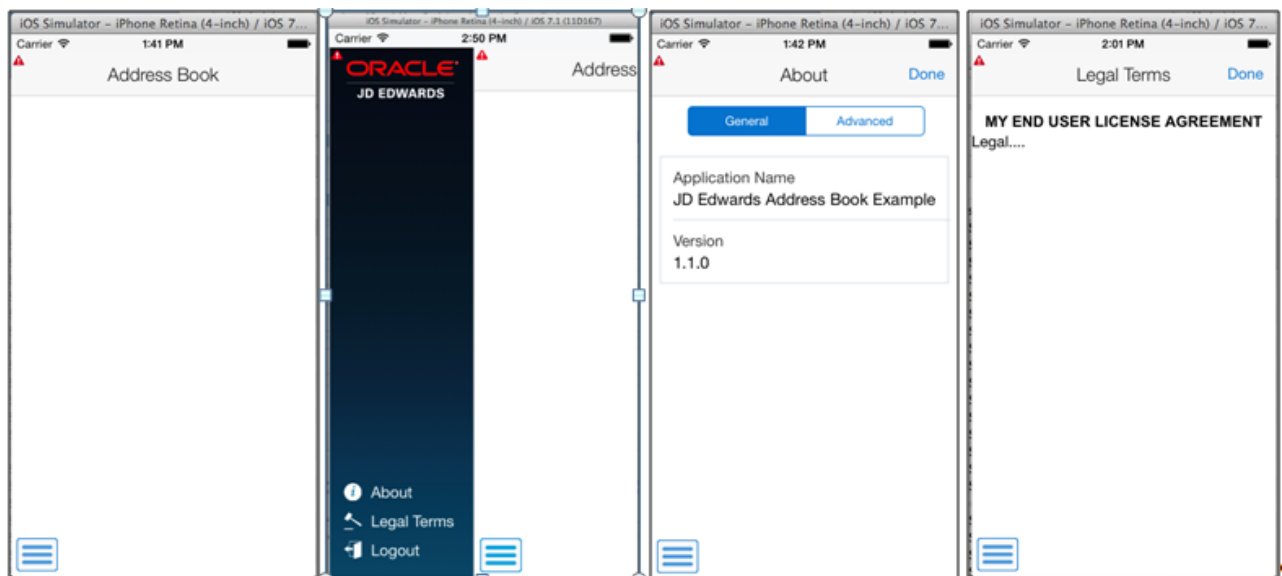
```
LoginConfiguration.setEULAFeature("jdemf.example.EULA");
```

The following example shows the line in the LifeCycleListenerImpl.java:



16. Deploy the application again. You should see a springboard launch icon.

The page displays the About, Legal Terms, and Logout links. The About page shows the information in about.properties; the Legal Terms page shows the eula.amx page:

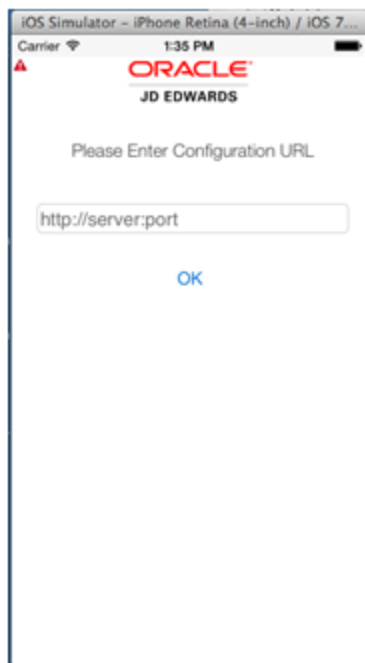


A.4 Connecting to the EnterpriseOne Application Interface Services (AIS) Server

After you configure the Login.jar, the application displays a configuration screen for connecting to the AIS Server.

To connect to the AIS Server:

1. On the first page in the mobile application, enter the URL to the AIS Server and click **OK**:



2. If the connection is successful, enter your EnterpriseOne user name and password in the following login page:



Based on the AIS Server configuration, the Environment, Role, and JAS Server fields, as well as the "Use Single Sign On" check box, may or may not be displayed here.

3. Click **Login**.

If the application takes you to the ABLIST feature, you have successfully established a session with the AIS Server. Your mobile application can make service calls.

A.5 AIS Client Class Generator

The AIS Client Class Generator is a JDeveloper extension that generates Java classes to store data returned from the REST service calls.

You must configure the AIS Client Class Generator before you can use it to generate classes. See [Section 5.2, "Configuring the AIS Client Class Generator"](#) in this guide.

A.6 Reading EnterpriseOne Data

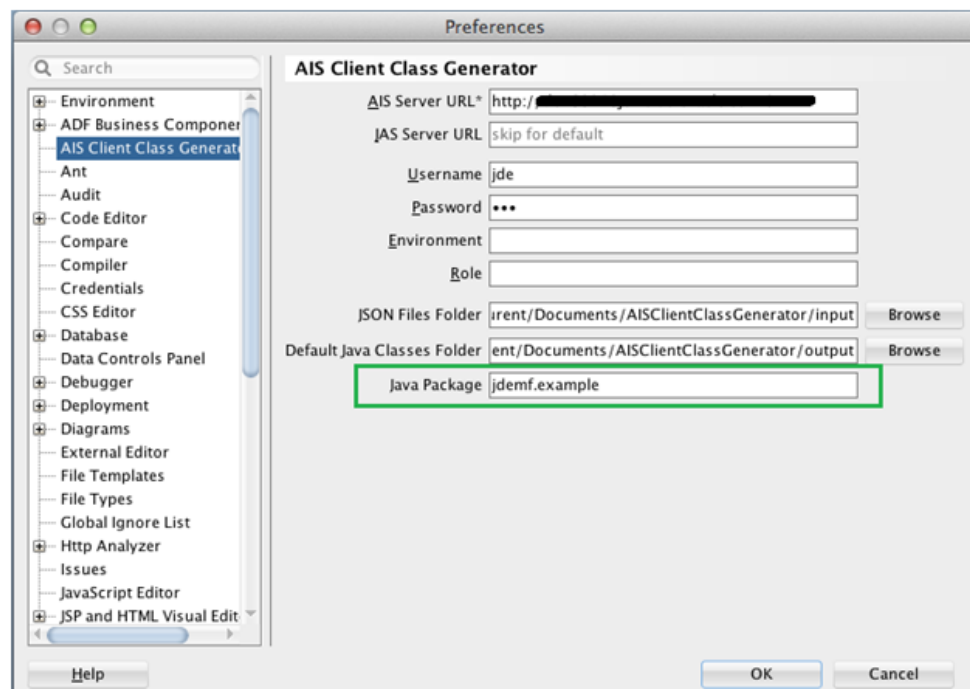
The JDEMobileFramework.jar includes APIs that enable the mobile application to make service calls to EnterpriseOne and read data from EnterpriseOne.

You use the AIS Client Class Generator to generate classes to hold the data returned from those calls. To do so, perform the following tasks:

- Configure the AIS Client Class Generator to generate classes in a custom package.
- Create a new Java class in the application package.
- Call the AIS service for populating the data in the Data Control.
- Create a UI for displaying the data.

To use the AIS Client Class Generator to generate classes in a custom package:

1. In JDeveloper, select the **Tools** menu > **Preferences**.



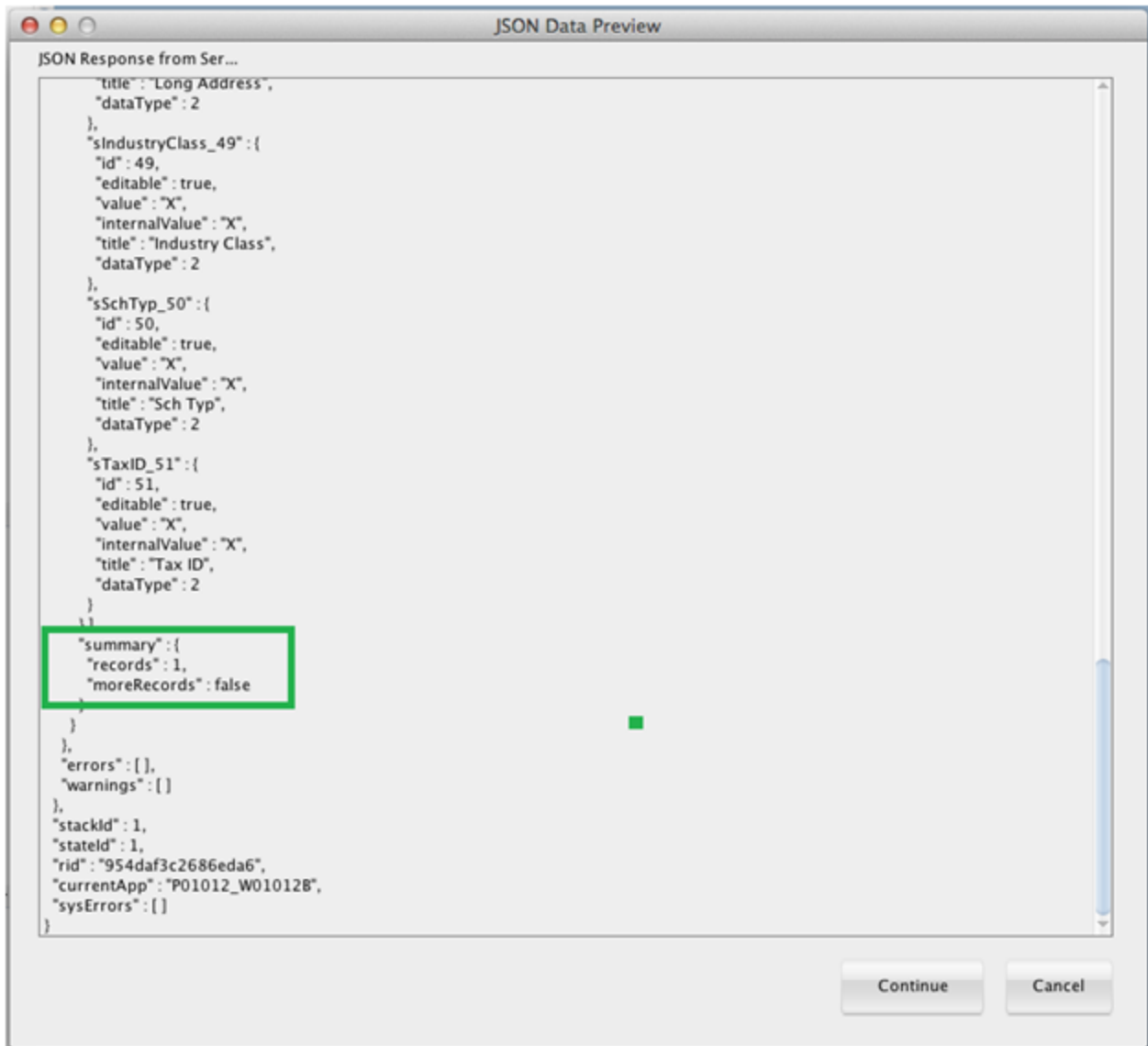
2. Select **AIS Client Class Generator** from the list on the left.
3. In the **Java Package** field, enter a valid package name, and then click **OK**.
4. In the Projects panel, click **ApplicationController** to highlight it. You must make sure that this is highlighted before generating classes.
5. Select the **Tools** menu > **AIS Client Class Generator** to launch it.

The screenshot shows the 'AIS Client Class Generator' dialog box. It is divided into several sections:

- Server URLs:** AIS Server URL: `http://aisserver:port`; JAS Server URL: `default`.
- Credentials:** Username: `username`; Password: `***`; Environment: (empty); Role: (empty).
- Parameters:**
 - Application Name: `P01012` (e.g. P01012)
 - Form Name: `W01012B` (e.g. W01012B)
 - Version: (empty) (e.g. ZJDE0001)
 - MaxPageSize: (empty) (e.g. 10)
 - ReturnControlIDs: (empty) (e.g. 1[19,20]58|62)
 - FormInputs: (empty) (e.g. (C,2);(D,3))
 - FormServiceAction: `R` (e.g. R)
 - FindOnEntry: (Executes FIND on entry.)
 - DemoMode: (Returns a dummy grid row.)
- Options:**
 - Generate for Mobile Applic...
 - Preview JSON Data
 - Keep JSON Files
- Generate** button.

6. In the AIS Client Class Generator, complete these fields to generate classes for the W01012B form in the P01012 application:
 - **Username.** Enter an EnterpriseOne username.
 - **Password.** Enter the EnterpriseOne password.
 - **Application Name.** Enter P01012.
 - **Form Name.** Enter W01012B.
7. Make sure the following check boxes are selected, and then click the **Generate** button.
 - **DemoMode**
 - **Generate for Mobile Application**

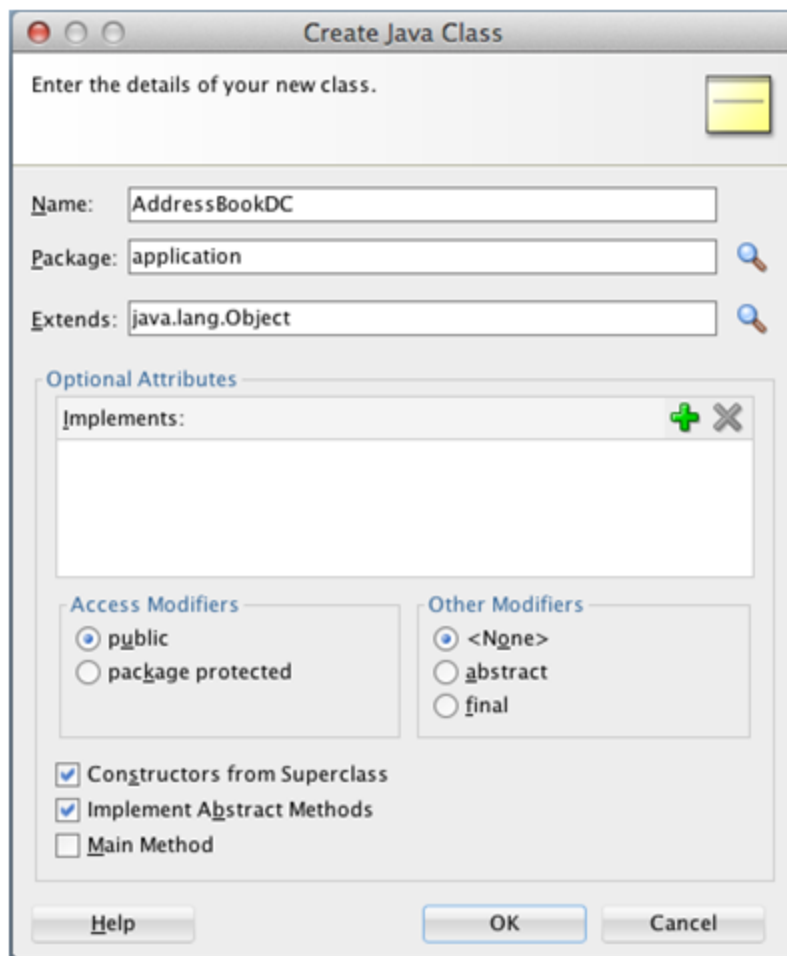
The Preview JSON Data and Keep JSON Files check boxes are optional. If you select the Preview JSON Data check box, the JSON displays a preview of the JSON, as shown in the following example:



8. On JSON Data Preview, make sure the preview shows at least one grid record, and then click **Continue**.
9. After receiving confirmation that the generation was a success, click **OK**.
10. In the Projects panel, click the "**refresh**" button so that the new classes appear in the ApplicationController tree.

To create a new Java class in the application package:

1. In the Projects panel, expand **ApplicationController > Application Sources**, right-click **application**, and then select **New > Java Class**.



2. On Create Java Class, enter AddressBookDC in the **Name** field and then click **OK**. This will be the main data control for the application.
3. Add a member to the AddressBookDC class of the P01012_W01012B_FormParent type. This will hold the data returned from the service call.

The placement of the following code for the member is shown in the image that follows:

```
P01012_W01012B_FormParent p01012_W01012B_FormParent = new P01012_W01012B_FormParent();
```




```

package application;

import jdemp.example.p01012.P01012_W01012B_FormParent;

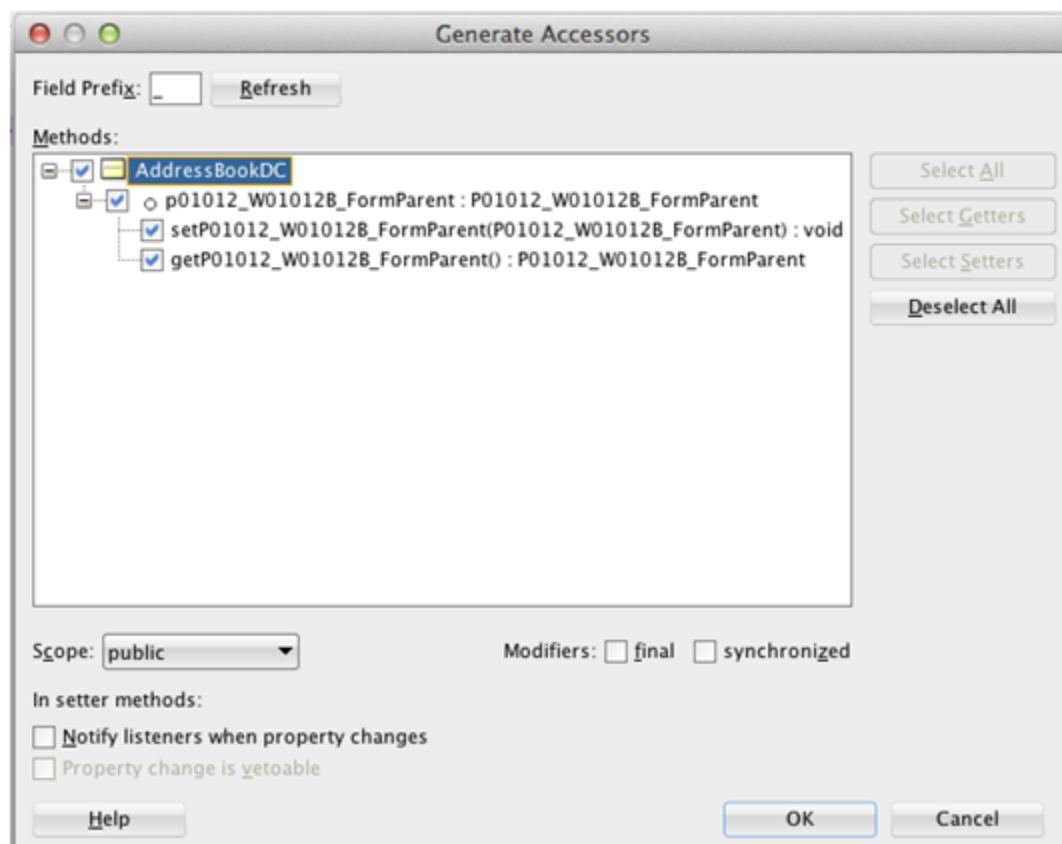
public class AddressBookDC {

    P01012_W01012B_FormParent p01012_W01012B_FormParent = new P01012_W01012B_FormParent();

    public AddressBookDC() {
        super();
    }
}

```

4. Right-click on the page and select **Generate Accessors** to generate accessors for the new member variable.



5. On Generate Accessors, select the check box next to the **AddressBookDC** method and click **OK**.
6. Under Application Controller > Application Sources > application, right-click **AddressBookDC.java** and select **Create Data Control**.
7. On Choose Bean Class, accept the defaults and click **Next**.
8. On Bean Data Control Options, accept the defaults and click **Finish**.
JDeveloper displays the new Data Control in the panel on the left.

To populate the data in the Data Control by calling the AIS service:

1. In the Projects panel, open the **AddressBookDC.java** class, and add the following code, which adds a new method to call the service and put the data in the member variable form parent:

```
public void getAddressBookList(){

    FormRequest formRequest = new FormRequest();
    formRequest.setFormName("P01012_W01012B");
    formRequest.setVersion("ZJDE0001");
    formRequest.setFormServiceAction("R");

    FSREvent w01012BFSREvent = new FSREvent();
    w01012BFSREvent.setFieldValue("54","C"); //set search type to customer

    w01012BFSREvent.doControlAction("15"); // Trigger the Find Button
    formRequest.addFSREvent(w01012BFSREvent); //add the events to the form
    request

    try
    {
        //Serialize the form request to JSON String
        JSONObject jsonObject =
        (JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
        String postData = jsonObject.toString();

        // Call to JDERestServiceProvider with parameters JSON string
        String response =
        JDERestServiceProvider.jdeRestServiceCall(postData,
        JDERestServiceProvider.POST,JDERestServiceProvider.FORM_SERVICE_URI);

        //deserialize the response to the class for the W01012B form
        p01012_W01012B_FormParent = (P01012_W01012B_
        FormParent)JSONBeanSerializationHelper.fromJSON(P01012_W01012B_
        FormParent.class, response);

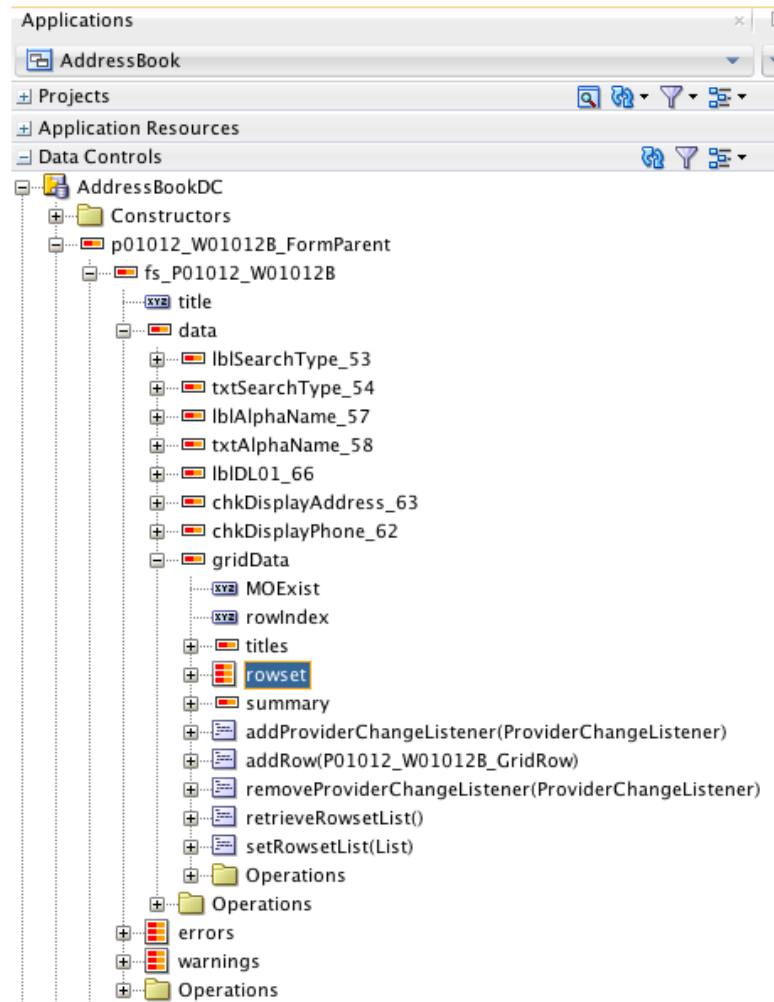
    }
    catch (JDERestServiceException e)
    {
        JDERestServiceProvider.handleServiceException(e);
    }
    catch(Exception e)
    {
        throw new AdfException(e.getMessage(), AdfException.ERROR);
    }
}
```

2. Call this new method from the constructor (for now).

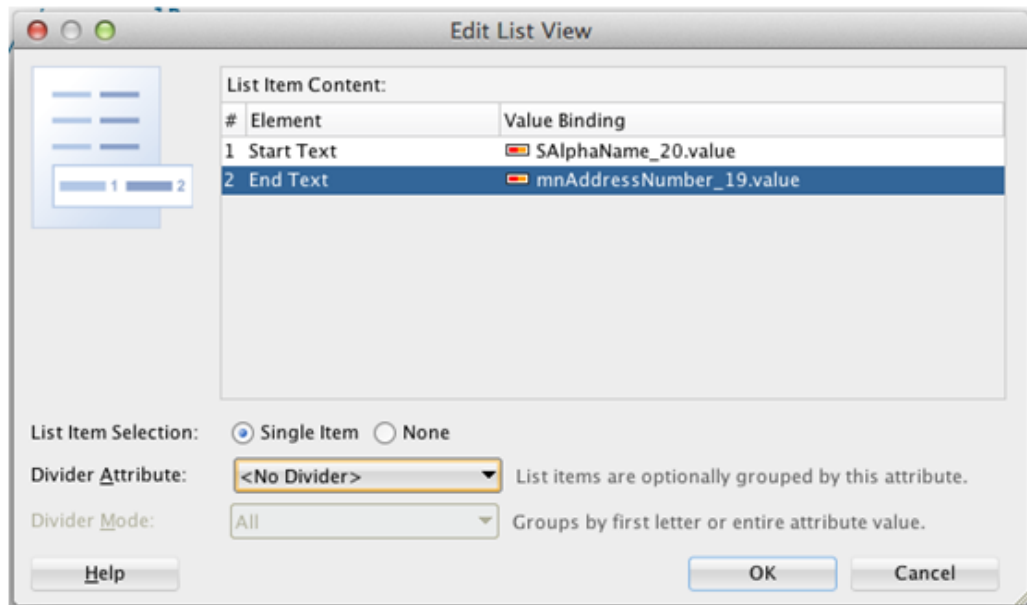
```
public AddressBookDC() {
    super();
    getAddressBookList();
}
```

To create a UI to display the data:

1. Under ViewController > Web Content, open the **ablist.amx** page.



2. In the Data Controls panel, locate the rowset in the data control, which is located under `p01012_W01012B_FormParent > fs_P01012_W01012B > data > gridData`.
3. Drag the **rowset** to the `ablist.amx` page after the `</amx:facet>` end tag.
4. Select **MAF List View**, and in the ListView Gallery, select the **Start-End** type of the list view.



5. For the Start Text, select **SAlphaName_20.value**, and for the End Text, select **mnAddressNumber_19.value**.
6. Click **OK**.
7. Locate the **amx:cellFormat** element and change the cell format that contains the alpha name to 80%. Change the cell format that contains the address number to 20%. See [Figure A-1](#) for an example.
8. In the Projects panel, expand **ViewController > Application Sources > mobile.pageDefs**, open the **ablistPageDef.xml**, and select the Source tab.

```

<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" version="12.1.3.9.1" id="ablistP
Package="mobile.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <iterator Binds="root" RangeSize="25" DataControl="AddressBookDC" id="AddressBookDCIterat
    <accessorIterator MasterBinding="AddressBookDCIterator" Binds="p01012_W01012B_FormParent"
      DataControl="AddressBookDC" BeanClass="jdemf.example.p01012.P01012_W010
      id="p01012_W01012B_FormParentIterator"/>
    <accessorIterator MasterBinding="p01012_W01012B_FormParentIterator" Binds="fs_P01012_W010
      DataControl="AddressBookDC" BeanClass="jdemf.example.p01012.P01012_W010
      id="fs_P01012_W01012BIterator"/>
    <accessorIterator MasterBinding="fs_P01012_W01012BIterator" Binds="data" RangeSize="25" D
      BeanClass="jdemf.example.p01012.P01012_W01012B_FormData" id="dataIterat
    <accessorIterator MasterBinding="dataIterator" Binds="gridData" RangeSize="25" DataContro
      BeanClass="jdemf.example.p01012.P01012_W01012B_GridData" id="gridDataIt
    <accessorIterator MasterBinding="gridDataIterator" Binds="rowset" RangeSize="25" DataCont
      BeanClass="jdemf.example.p01012.P01012_W01012B_GridRow" id="rowsetItera
  </executables>
  <bindings>
    <tree IterBinding="rowsetIterator" id="rowset">
      <nodeDefinition DefName="jdemf.example.p01012.P01012_W01012B_GridRow" Name="rowset10">
        <AttrNames>
          <Item Value="value"/>
        </AttrNames>
        <Accessors>
          <Item Value="SAlphaName_20"/>
          <Item Value="mnAddressNumber_19"/>
        </Accessors>
      </nodeDefinition>
      <nodeDefinition DefName="jdemf.example.Field">
        <AttrNames>
          <Item Value="value"/>
        </AttrNames>
      </nodeDefinition>
    </tree>
  </bindings>
</pageDefinition>

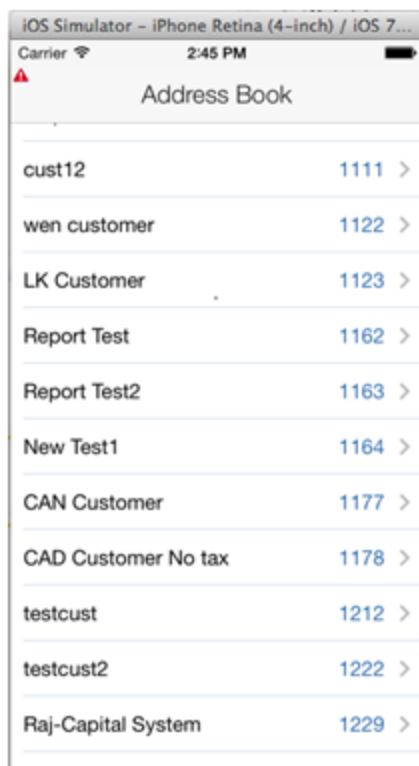
```

9. In the `<AttrName>` element, add the following line of code:

```
<Item Value="value"/>
```

10. Deploy and run the application.

After logging in, it should automatically populate the list with customer names as shown in the following example:



A.7 Implementing Filter Fields

Implement a Name (string) filter. To do so:

1. Select the **AddressBookDC.java** tab, and add the following code to add a member variable:

```
String nameFilter;
```

2. Right-click on the page and select **Generate Accessors**.
3. Add the following code to the `getAddressBookList()` method to use the new filter value:

```
if(nameFilter!=null && nameFilter.trim().length()>0){  
    //set filter name in QBE  
    w01012BFSREvent.setQBEValue("1[20]", "*" + nameFilter.trim() + "*");  
}
```

The following example shows the code in the method:

```

public class AddressBookDC {

    P01012_W01012B_FormParent p01012_W01012B_FormParent = new P01012_W01012B_FormParent();
    String nameFilter;

    public AddressBookDC() {
        super();
        getAddressBookList();
    }

    public void getAddressBookList(){

        FormRequest formRequest = new FormRequest();
        formRequest.setFormName("P01012_W01012B");
        formRequest.setVersion("ZJDE0001");
        formRequest.setFormServiceAction("R");

        FSREvent w01012BFSREvent = new FSREvent();
        w01012BFSREvent.setFieldValue("54","C"); //set search type to customer

        if(nameFilter!=null && nameFilter.trim().length()>0){
            w01012BFSREvent.setQBEValue("1[20]","*" + nameFilter.trim() + "*"); //set filter name in QBE
        }

        w01012BFSREvent.doControlAction("15"); // Trigger the Find Button
        formRequest.addFSREvent(w01012BFSREvent); //add the events to the form request

        try
        {
            //Serialize the form request to JSON String

```

4. Remove the call to `getAddressBookList()` in the constructor. Later in these steps, you will call this method from the search button instead.
5. Change the code after the response is retrieved. This will enable the data to refresh after a new search is performed.

```

//deserialize the response to the class for the W01012B form
P01012_W01012B_FormParent temp_p01012_W01012B_FormParent =
(P01012_W01012B_FormParent)JSONBeanSerializationHelper.fromJSON(P01012_
W01012B_FormParent.class, response);

p01012_W01012B_FormParent.getFs_P01012_
W01012B().getData().getGridData().setRowsetWithList(temp_p01012_W01012B_
FormParent.getFs_P01012_
W01012B().getData().getGridData().retrieveRowsetList());

```

The following example shows the code:

```

//Serialize the form request to JSON String
JSONObject jsonObject = (JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
String postData = jsonObject.toString();

// Call to JDERestServiceProvider with parameters JSON string
String response = JDERestServiceProvider.jdeRestServiceCall(postData, JDERestServiceProvider.POST, JDERestServiceProvider.FORM_SERVICE_

//deserialize the response to the class for the W01012B form
P01012_W01012B_FormParent temp_p01012_W01012B_FormParent = (P01012_W01012B_FormParent)JSONBeanSerializationHelper.fromJSON(P01012_W010
p01012_W01012B_FormParent.getFs_P01012_W01012B().getData().getGridData().setRowsetList(temp_p01012_W01012B_FormParent.getFs_P01012_W01

}
catch (JDERestServiceException e)

```

6. Under **ViewController > Web Content**, open `ablist.amx`.
7. In the Data Controls panel, expand **AddressBookDC** and locate the new `nameFilter` field.

8. Drag the **nameFilter** to the amx page before the start of the list view and after the end of the secondary facet.
9. On the context menu, select **Text > MAF Input Text w/Label**.
10. Change the label to:


```
label= "Name"
```
11. Remove the existing button in the secondary facet.
12. In the Data Controls panel, locate the **getAddressBookList()** method in the AddressBookDC data control.
13. Drag the method inside the secondary facet in the amx page, and click **MAF Button** in the context menu to create a button.
14. To name the new button, change the text attribute to:


```
text="Search"
```
15. Add a panelFormLayout around the name input field, which will delineate it from the list with a box, as shown in the following example:


```
<amx:panelFormLayout labelPosition="start" fieldHalign="start">
  <amx:inputText value="#{bindings.nameFilter.inputValue}" label="Name"
  id="it1"/>
</amx:panelFormLayout>
```
16. Deploy the application and perform a search by entering a value in the Name field, which should return matching results, as shown here:



A.8 Page Navigation and Getting More Details

This section describes how to generate classes that enable the mobile application to retrieve additional details from the Address Book application.

To generate the classes that will hold the additional details:

1. Select the **Tools** menu > **AIS Client Class Generator** to launch the AIS Client Class Generator.

By default, the AIS Client Class Generator should display the values and options that you entered previously.

This time, you need to request the following five fields from the W01012A form in EnterpriseOne, which you specify in the ReturnControlIDs field in the generator:

AlphaName(28), AddressLine1(40), City(52), State(54), PostalCode(50)

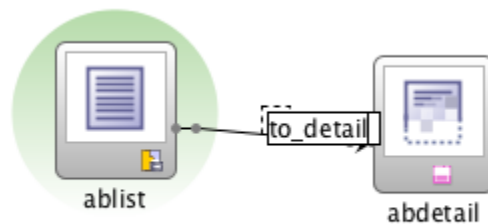
2. To generate classes, complete these fields and click the **Generate** button:
 - **Application Name:** P01012
 - **Form Name:** W01012A
 - **ReturnControlIDs:** 28|40|52|54|50
 - **Demo Mode:** Checked

JDeveloper displays the generated classes in the Projects panel, under **ApplicationController > Application Sources > application** node.

3. Create a member variable in the AddressBookDC class, as shown in the following example:

```
public class AddressBookDC {
    P01012_W01012B_FormParent p01012_W01012B_FormParent = new P01012_W01012B_FormParent();
    String nameFilter;
    P01012_W01012A_FormParent p01012_W01012A_FormParent = new P01012_W01012A_FormParent();
}
```

4. Right-click and select **Generate Accessors**.
5. On Generate Accessors, select the check box next to **AddressBookDC** and click **OK**.
6. Under the **ViewController > Web Content > ABList**, open the **ViewController-task-flow.xml**.
7. To add another view to the task flow, in the Components window, drag View onto the page.
8. Enter **abdetail** for the view name.
9. Add a control flow by dragging it from the Components and extending it from **ablist** icon to the **abdetail** icon.



10. Enter **to_detail** for the name.
11. Double-click the **abdetail** view to generate the amx. You can use the default values on the Create MAF AMX Page dialog box.
12. In the **abdetail.amx** page, enter the following code to configure a back button:

```
<amx:facet name="primary">
  <amx:commandButton id="cb1" action="__back" text="Back"/>
</amx:facet>
```

```
</amx:facet>
```

- Open the **ablist.amx** page and set the action for a list item to use the "to_detail" navigation, as shown in the following example:

Figure A-1 Example of setting page navigation action

```
</amx:panelFormLayout>
<amx:listView var="row" value="#{bindings.rowset.collectionModel}" fetchSize="#{bindings.rowset.rangeSize}"
  selectedRowKeys="#{bindings.rowset.collectionModel.selectedRow}"
  selectionListener="#{bindings.rowset.collectionModel.makeCurrent}" showMoreStrategy="autoScroll"
  bufferStrategy="viewport" id="lv1">
  <amx:listItem id="lil" action="to_detail">
    <amx:tableLayout width="100%" id="tl1">
      <amx:rowLayout id="rl1">
        <amx:cellFormat width="10px" id="cf3"/>
        <amx:cellFormat width="80%" height="43px" id="cf4">
          <amx:outputText value="#{row.SAlphaName_20.bindings.value.inputValue}" id="ot3"/>
        </amx:cellFormat>
        <amx:cellFormat width="10px" id="cf2"/>
        <amx:cellFormat width="20%" valign="end" id="cf1">
          <amx:outputText value="#{row.mnAddressNumber_19.bindings.value.inputValue}"
            styleClass="admf-listItem-highlightText" id="ot2"/>
        </amx:cellFormat>
      </amx:rowLayout>
    </amx:tableLayout>
  </amx:listItem>
</amx:listView>
```

- Return to the **abdetil.amx** file and create a panelFormLayout after the last facet, as shown in the following example:

```
</amx:facet>
<amx:panelFormLayout labelPosition="topStart" fieldHalign="start">
```

- In the Data Controls panel, locate the address number value within the W0102B form, which is located under **AddressBookDC > p01012_W01012B_FormParent > fs_P01012_W01012B > data > gridData > rowset > mnAddressNumber_19**.
- Drag the **value** field under **mnAddressNumber_19** to the page inside the panelFormLayout, and in the context menu, select **Text > MAF Output Text w/Label**.
- Change the label to "Address Number" as shown in the following example:

```
<amx:panelFormLayout labelPosition="topStart" fieldHalign="start">
  <amx:panelLabelAndMessage label="Address Number" id="plam1">
    <amx:outputText value="#{bindings.value.inputValue}" id="ot2"/>
  </amx:panelLabelAndMessage>
</amx:panelFormLayout>
```

- Locate the data area of the W01012A form, which is under **AddressBookDC > p01012_W01012A_FormParent > fs_P01012_W01012A > data**.
- Drag the **value** that is within the **txtAlphaName_28** data element of the **AlphaName** to the page after the panelLabelAndMessage for **AddressNumber**.
- In the context menu, select **Text > MAF Input Text w/Label**.
- Repeat for the other detail fields, and set the labels with their names, as shown in the following example:

```

</amx:facet>
<amx:panelFormLayout labelPosition="topStart" fieldHalign="start">
  <amx:panelLabelAndMessage label="Address Number" id="plam1">
    <amx:outputText value="{bindings.value.inputValue}" id="ot2"/>
  </amx:panelLabelAndMessage>
  <amx:inputText value="{bindings.value1.inputValue}" label="Name" id="it1"/>
  <amx:inputText value="{bindings.value2.inputValue}" label="Address" id="it2"/>
  <amx:inputText value="{bindings.value3.inputValue}" label="City" id="it3"/>
  <amx:inputText value="{bindings.value4.inputValue}" label="State" id="it4"/>
  <amx:inputText value="{bindings.value5.inputValue}" label="Zip" id="it5"/>
</amx:panelFormLayout>
</amx:panelPage>
/amx:view>

```

22. Open the `AddressBookDC.java` and create this new method:

```

public void getAddressBookDetail(String addressNumber) {
    FormRequest formRequest = new FormRequest();
    formRequest.setFormName("P01012_W01012A");
    formRequest.setVersion("ZJDE0001");
    formRequest.setFormServiceAction("R");
    formRequest.addToFISet("12", addressNumber);

    try {
        //Serialize the form request to JSON String
        JSONObject jsonObject = (JSONObject)
        JSONBeanSerializationHelper.toJSON(formRequest);
        String postData = jsonObject.toString();

        // Call to JDERestServiceProvider with parameters JSON string
        String response =
            JDERestServiceProvider.jdeRestServiceCall(postData,
            JDERestServiceProvider.POST,
            JDERestServiceProvider.FORM_SERVICE_URI);

        //deserialize the response to the class for the W01012A form
        p01012_W01012A_FormParent =
            (P01012_W01012A_FormParent)
            JSONBeanSerializationHelper.fromJSON(P01012_W01012A_FormParent.class,
            response);

        } catch (JDERestServiceException e) {
            JDERestServiceProvider.handleServiceException(e);
        } catch (Exception e) {
            throw new AdfException(e.getMessage(), AdfException.ERROR);
        }
    }
}

```

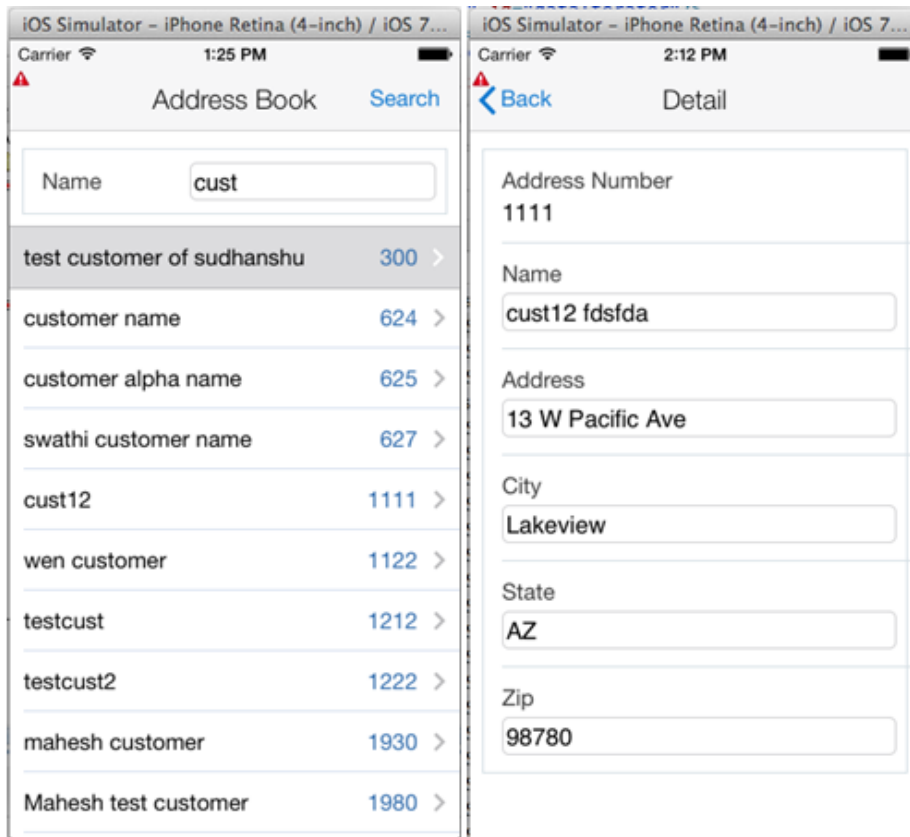
23. Configure the application so that the method is called when the user loads the detail page.

- a. Open the `abdetail.amx` and use the tabs at the bottom to open the Bindings.
- b. Click the **green plus sign** to add a binding to the Bindings column.
- c. On Insert Item, add a `methodAction`.
- d. Select the `getAddressBookDetails(String)` method, and enter the following value for the input. This is the value of the `mnAddressNumber_19` field.

```
{bindings.value.inputValue}
```

- e. In the Executables column, click the **Add** icon.
- f. On Insert Item, select **invokeAction** and click **OK**.
- g. On Insert invokeAction, enter `getDetails` for the **id**.
- h. Select the **getAddressBookDetail** method and then click **OK**.
- i. In the Executables page, select the **getDetails** action and drag it up in the list, directly after `mnAddressNumber_19Iterator`.
- j. Highlight **getDetails**, click the **Pencil** icon, and set Refresh to "always."
- k. Click **OK** and then test the application in the simulator.

If successful, the simulator displays the Address Book details when you select a record, as shown in the following illustration:



A.9 Updating Data in EnterpriseOne

Next, use the details form to update the values in the database.

To update the values:

1. In the `AddressBookDC` class, add the following new method to update the values:

```
public void updateAddressBookDetail(String addressNumber) {
    FormRequest formRequest = new FormRequest();
    formRequest.setFormName("P01012_W01012A");
    formRequest.setVersion("ZJDE0001");
    formRequest.setFormServiceAction("U");
    formRequest.addToFISet("12", addressNumber);
}
```

```

        FSREvent w01012AUpdateFSREvent = new FSREvent();//set all the field
values
        w01012AUpdateFSREvent.setFieldValue("28", p01012_W01012A_
FormParent.getFs_P01012_W01012A().getData().getTxtAlphaName_28().getValue());
        w01012AUpdateFSREvent.setFieldValue("40",p01012_W01012A_
FormParent.getFs_P01012_W01012A().getData().getTxtAddressLine1_40().getValue()
);
        w01012AUpdateFSREvent.setFieldValue("52",p01012_W01012A_
FormParent.getFs_P01012_W01012A().getData().getTxtCity_52().getValue() );
        w01012AUpdateFSREvent.setFieldValue("54", p01012_W01012A_
FormParent.getFs_P01012_W01012A().getData().getTxtState_54().getValue());
        w01012AUpdateFSREvent.setFieldValue("50", p01012_W01012A_
FormParent.getFs_P01012_W01012A().getData().getTxtPostalCode_50().getValue());

        w01012AUpdateFSREvent.doControlAction("11"); // Trigger the OK
Button
        formRequest.addFSREvent(w01012AUpdateFSREvent); //add the events to
the form request

        try {
            //Serialize the form request to JSON String
            JSONObject jsonObject = (JSONObject)
JSONBeanSerializationHelper.toJSON(formRequest);
            String postData = jsonObject.toString();

            // Call to JDERestServiceProvider with parameters JSON string
            String response =
                JDERestServiceProvider.jdeRestServiceCall(postData,
JDERestServiceProvider.POST,
JDERestServiceProvider.FORM_SERVICE_URI);

            //after the call navigate back to the list page
AdfmfContainerUtilities.invokeContainerJavaScriptFunction("ABLlist",
"adf.mf.api.amx.doNavigation", new Object[]{"__back"});

        } catch (JDERestServiceException e) {
            JDERestServiceProvider.handleServiceException(e);
        } catch (Exception e) {
            throw new AdfException(e.getMessage(), AdfException.ERROR);
        }
    }
}

```

2. Open the **abdetail.amx** page.
3. Select the **updateAddressbookDetail(String)** method from the data control.
4. Drag it between the secondary facet (remove the existing button first).
5. On Edit Action Binding, set the parameter value to: **#{bindings.value.inputValue}**
This is the binding value that corresponds to the address number.
6. Run the application in the simulator to test the following actions:
 - a. Change the values.
 - b. Click **Save**.

- c. Search on a new name.

A.9.1 Handling Errors

Some of the fields may potentially have errors such as an invalid state code. This section describes how to handle those errors.

1. Add the following items to the AddressBookDC class:

- An error array member variable.
- A property change support member variable.
- A provider change member variable.
- Setters and getters.
- Provider change support methods.

This code also provides a getter for an `errorCount`, which is set when the array is set using the property change listener, as shown in the following code sample:

```
FormErrorWarningMobile[] errors;
    private PropertyChangeSupport propertyChangeSupport = new
PropertyChangeSupport(this);
    protected transient ProviderChangeSupport errorChangeSupport = new
ProviderChangeSupport(this);

public void addProviderChangeListener(ProviderChangeListener l)
{
    errorChangeSupport.addProviderChangeListener(l);
}

public void removeProviderChangeListener(ProviderChangeListener l)
{
    errorChangeSupport.removeProviderChangeListener(l);
}

public void setErrors(FormErrorWarningMobile[] errors) {
    int oldCount = 0;
    if(this.errors!=null){
        oldCount = this.errors.length;
    }
    int newCount = 0;
    if(errors != null){
        newCount = errors.length;
    }
    this.errors = errors;
    errorChangeSupport.fireProviderRefresh("errors");
    propertyChangeSupport.firePropertyChange("errorCount",
oldCount,newCount);
}

public FormErrorWarningMobile[] getErrors() {
    return errors;
}

public int getErrorCount()
{
    if(errors != null){
        return errors.length;
    }else{
        return 0;
    }
}
```

```

    }

    }

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

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

```

2. Modify the code in the `updateAddressBookDetail` method of the `AddressBookDC` class. Instead of always navigating back to the list first, check the errors and then return to the list.

Replace this line:

```

//after the call navigate back to the list page
AdfmfContainerUtilities.invokeContainerJavaScriptFunction("ABLlist",
"adf.mf.api.amx.doNavigation", new Object[]{"__back"});

```

With this code:

```

//deserialize the response to the class for the W01012A form
P01012_W01012A_FormParent temp_p01012_W01012A_FormParent =
(P01012_W01012A_FormParent)
JSONBeanSerializationHelper.fromJSON(P01012_W01012A_FormParent.class,
response);

clearErrors();
if(temp_p01012_W01012A_FormParent.getFs_P01012_
W01012A().getErrors() != null
&& temp_p01012_W01012A_FormParent.getFs_P01012_
W01012A().getErrors().length > 0
){

//set error list
setErrors(temp_p01012_W01012A_FormParent.getFs_P01012_W01012A().getErrors());
}else{

//no errors navigate back
AdfmfContainerUtilities.invokeContainerJavaScriptFunction("ABLlist",
"adf.mf.api.amx.doNavigation", new Object[]{"__back"});
}

public void clearErrors(){
setErrors(null);
}

```

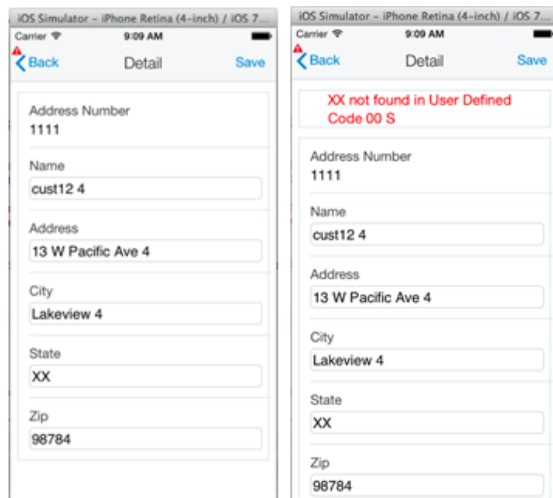
3. Open `abdetail.amx`.
4. In the Data Controls panel, select **errors** under `AddressBookDC` and drag it to the page, placing it after the last facet to create a list view.
5. On the context menu, select **MAF List View**.
6. On List View Gallery, select the default view.
7. On Edit List View, select **MOBILE** from the Text drop-down menu and click **OK**.
8. In the Data Control panel, locate the **errorCount** under `AddressBookDC`.

9. Drag **errorCount** onto the page and in the context menu, select **Text > MAF Output Text**.
10. Remove the list view and output text that you previously added; they were just added to create bindings.
11. After the last facet, insert the following list view code:

```
<amx:listView var="row" value="#{bindings.errors.collectionModel}"
fetchSize="#{bindings.errors.rangeSize}"
                styleClass="admf-listView-insetList" id="lv1"
rendered="#{bindings.errorCount.inputValue gt 0}">
  <amx:listItem showLinkIcon="false" id="li1">
    <amx:panelGroupLayout>
      <amx:inputText rows="3" value="#{row.MOBILE}"
id="ot3" inlineStyle="color:Red;word-wrap:break-word;"
                                readOnly="true"/>
    </amx:panelGroupLayout>
  </amx:listItem>
</amx:listView>
```

12. Run the application in the simulator and test the following tasks in the application:
 - a. Enter an invalid state code.
 - b. Click **Save**.

The simulator should display the "not found" error as shown in the following example:



If you correct it with a valid value and click **Save**, the application will successfully save the record.

A.10 Device Integration

This example shows how to enable the mobile application user to add a record they are viewing to the contacts list on the device. To do so:

1. Open the **maf-application.xml** and select the **Plugins** tab.
2. Select the check box next to **Contacts** to enable access to the Contacts on the device.

3. Add a member variable to the AddressBookDC.java class to hold the confirmation message.
4. Generate accessors with property change support.

After defining the member variable and generating accessors, the resulting code should look like this:

```
private String addContactResultMsg = null;

    public String getAddContactResultMsg() {
        return addContactResultMsg;
    }

public void setAddContactResultMsg(String addContactResultMsg) {
    String oldAddContactResultMsg = this.addContactResultMsg;
    this.addContactResultMsg = addContactResultMsg;
    propertyChangeSupport.firePropertyChange("addContactResultMsg",
oldAddContactResultMsg, addContactResultMsg);
}
```

5. Add the following new method, which enables the record to be added to the device's contacts list:

```
public void createDeviceContact() {
    String resultMessage = "Contact Exists";

    // Call FindContacts to determine if the contact already exists on
device.
    String contactSearchName =
        p01012_W01012A_FormParent.getFs_P01012_
W01012A().getData().getTxtAlphaName_28().getValue();

    try {
        Contact[] foundContacts =
            DeviceManagerFactory.getDeviceManager().findContacts("name",
contactSearchName, true);

        if (foundContacts != null && foundContacts.length == 0) {
            Contact newContact = new Contact();

            // Assign contact name.
            ContactName name = new ContactName();
            name.setFormatted(contactSearchName);
            name.setGivenName(contactSearchName);
            newContact.setName(name);

            // Assign address to contact.
            ContactAddresses[] addresses = new ContactAddresses[1];
            ContactAddresses address = new ContactAddresses();
            address.setStreetAddress(p01012_W01012A_FormParent.getFs_
P01012_W01012A().getData().getTxtAddressLine1_40().getValue());
            address.setLocality(p01012_W01012A_FormParent.getFs_P01012_
W01012A().getData().getTxtCity_52().getValue());
            address.setPostalCode(p01012_W01012A_FormParent.getFs_P01012_
W01012A().getData().getTxtPostalCode_50().getValue());
            address.setRegion(p01012_W01012A_FormParent.getFs_P01012_
W01012A().getData().getTxtState_54().getValue());
            addresses[0] = address;
            newContact.setAddresses(addresses);
        }
    }
}
```

```

        try {
            // Add contact to device.

DeviceManagerFactory.getDeviceManager().createContact(newContact);

            // Device contact now exists.
            resultMessage = "Contact Added";

        } catch (Exception e) {
            throw new AdfException(e.getMessage(),
DeviceManagerFactory.getDeviceManager().createContact(newContact);
            AdfException.ERROR);
        }
    }
} catch (Exception e) {
    // Unable to interact with device's contact list - display error.
    resultMessage = "Unable to Add Contact";
}

    setAddContactResultMsg(resultMessage);
}

```

6. Open the **abdetail.amx** page and locate the new method called **createDeviceContact** under the **AddressBookDC** data control.
7. Drag the method to the page as a button, and then delete the button.
This creates the bindings.
8. Locate the **addContactResultMsg** under the **AddressBookDC** data control.
9. Drag it to the page as an output text, and then delete the output text.
This creates the bindings.
10. Enter the following code for the footer after the end of the panel form layout (`</amx:panelFormLayout>`):

```

<amx:facet name="footer">
    <amx:panelGroupLayout id="pg16" layout="horizontal" halign="end">
        <!-- Add Contact To Device-->
        <amx:commandButton
actionListener="#{bindings.createDeviceContact.execute}" text="Create Device
Contact "

disabled="#{!bindings.createDeviceContact.enabled}" id="cb4">
            <amx:showPopupBehavior id="spb1" popupId="p1"
align="overlapTopCenter" alignId="it2"/>
        </amx:commandButton>
    </amx:panelGroupLayout>
</amx:facet>

```

11. Enter the following code for the popup message after the end panel page tag (`</amx:panelPage>`):

```

<!-- display add contact result message. -->
    <amx:popup id="p1" autoDismiss="true">
        <amx:panelGroupLayout id="pg13" halign="center" valign="middle">
            <amx:commandLink text="#{bindings.addContactResultMsg.inputValue}"
id="cl3"

inlineStyle="white-space:pre;font-size:larger;color:black;text-decoration:none

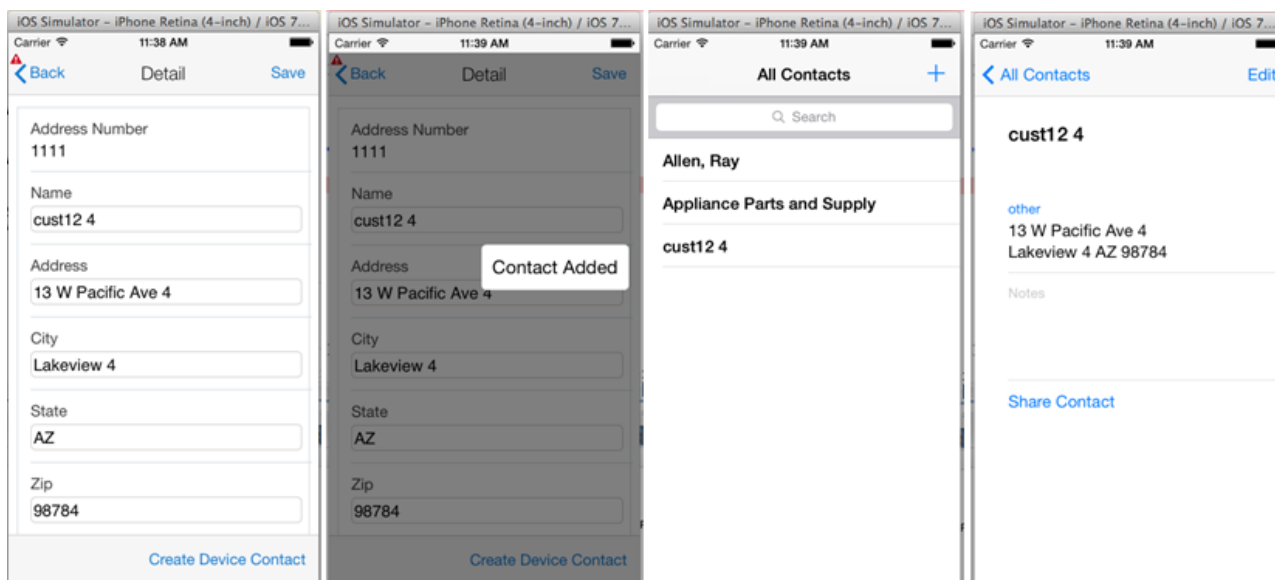
```

```

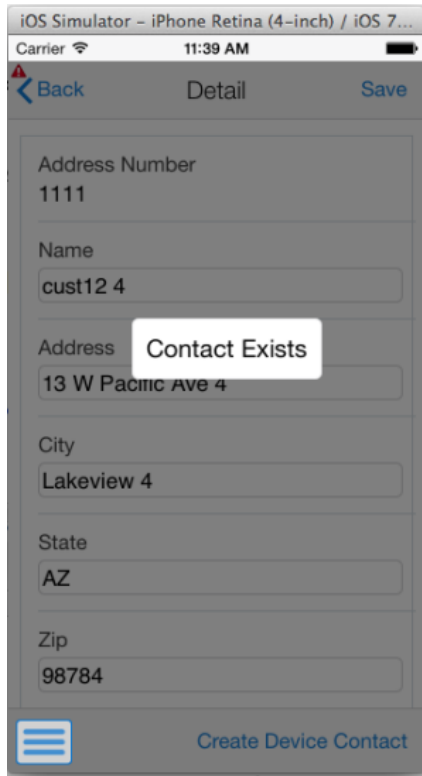
; ">
        <amx:closePopupBehavior id="cpb1" popupId="p1"/>
    </amx:commandLink>
</amx:panelGroupLayout>
</amx:popup>

```

12. Run the application in the simulator to test performing the following tasks in the application:
 - a. Open a record and click the **Create Device Contact** button in the footer, which displays the "Contact Added" message.
 - b. Go to the device's contacts and you can see the contact listed with the address, as shown in the following example:



If you attempt to add it again, you get the contact exists message, as shown in the following example:



Extending Mobile Application Archives

This appendix contains the following topics:

- Section B.1, "Before You Begin"
- Section B.2, "Understanding Mobile Application Archives"
- Section B.3, "Generating a New Application from the Deployment Profile"
- Section B.4, "Customization Options"
- Section B.5, "Extension Options"

B.1 Before You Begin

To use mobile application archives, in addition to the prerequisites described in Section 3.2, "Prerequisites" in this guide, perform the following tasks.

- Download the JD Edwards EnterpriseOne mobile application archive (.maa) files from the JD Edwards Update Center on My Oracle Support (<https://support.oracle.com/>). In the Update Center, enter "EnterpriseOne Mobile Enterprise Applications" in the Type field to locate the MAAs.
- Remember that to deploy to the Apple store or to your own enterprise store for iOS, you will need an Apple development profile.
- You must read the "JD Edwards EnterpriseOne Mobile Archive Restricted Use Notice" in the *JD Edwards EnterpriseOne Licensing Information User Manual* before using JD Edwards EnterpriseOne mobile archives.

Important: The Oracle MAF extension for JDeveloper must be the same version that was used to create the mobile application archive.

- MAA files developed with version 2.1 require the use of Oracle MAF extension 2.1, which is included in the JDE_Mobile_Framework_2.1 package on the Update Center.
- MAA files created with version 2.0 require the use of Oracle MAF extension 2.0, which is included in the JDE_Mobile_Framework_2.0 package on the Update Center.

To verify the version of the Oracle MAF extension:

1. In JDeveloper click the Help menu, About, and then click the Extensions tab.
2. In the "find" field, type `maf` and then press Enter. Scroll to the right to check the version.

B.2 Understanding Mobile Application Archives

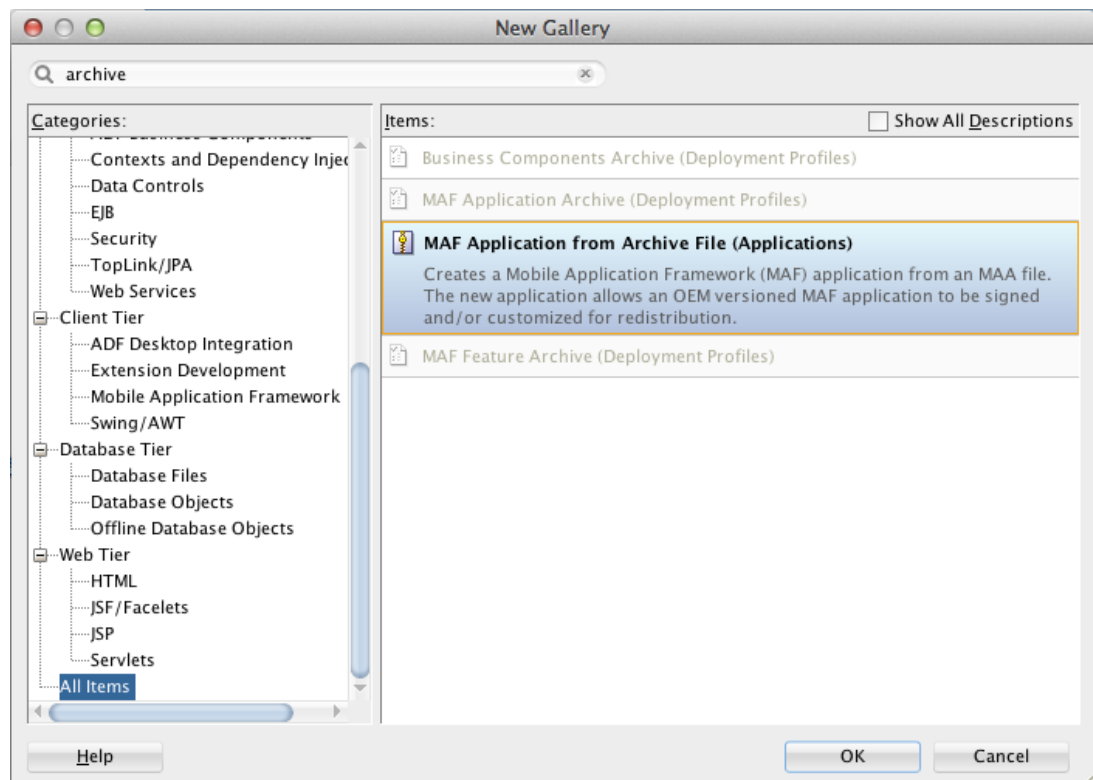
A mobile application archive (MAA) is a specially built JDeveloper project that enables developers to take an existing MAF application and generate their own application binary (apk or ipa) files. Developers can choose to customize the application and then sign it and deploy it with their own distribution profile. You can use an archive to generate a mobile application that you can deploy and manage locally in your environment without having to download mobile applications from an online web app store.

You can partially customize an MAA to suit your particular business needs. To do so, you copy the archive, customize it, and then publish it as your own "new" application. For iOS, you must publish it with a different application ID in order to install it with your own iOS development profile.

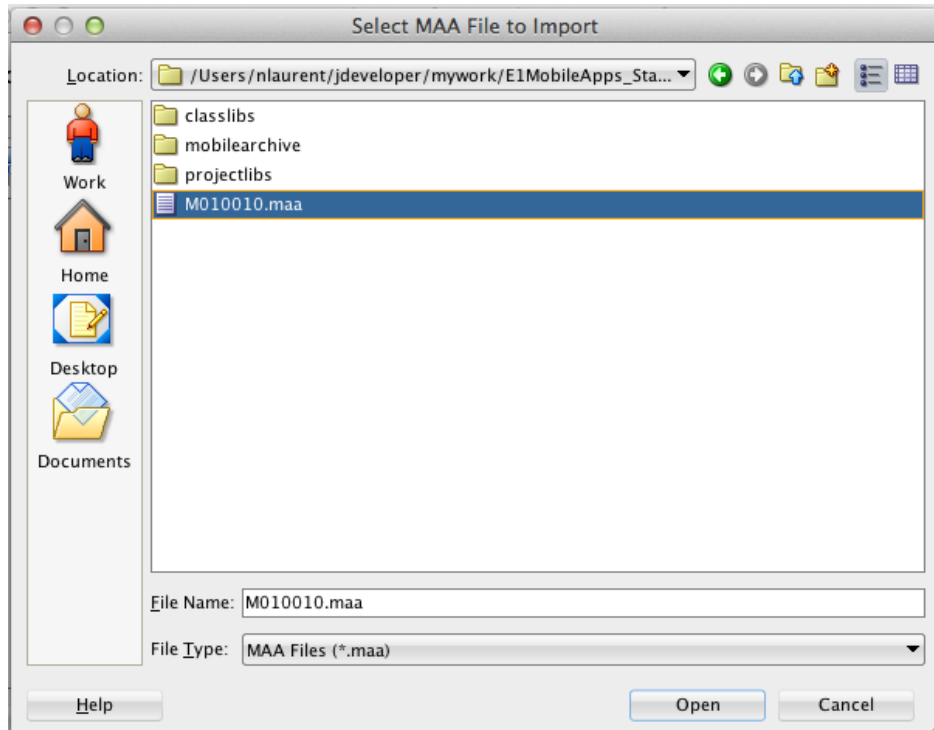
B.3 Generating a New Application from the Deployment Profile

To create a custom copy of an MAA:

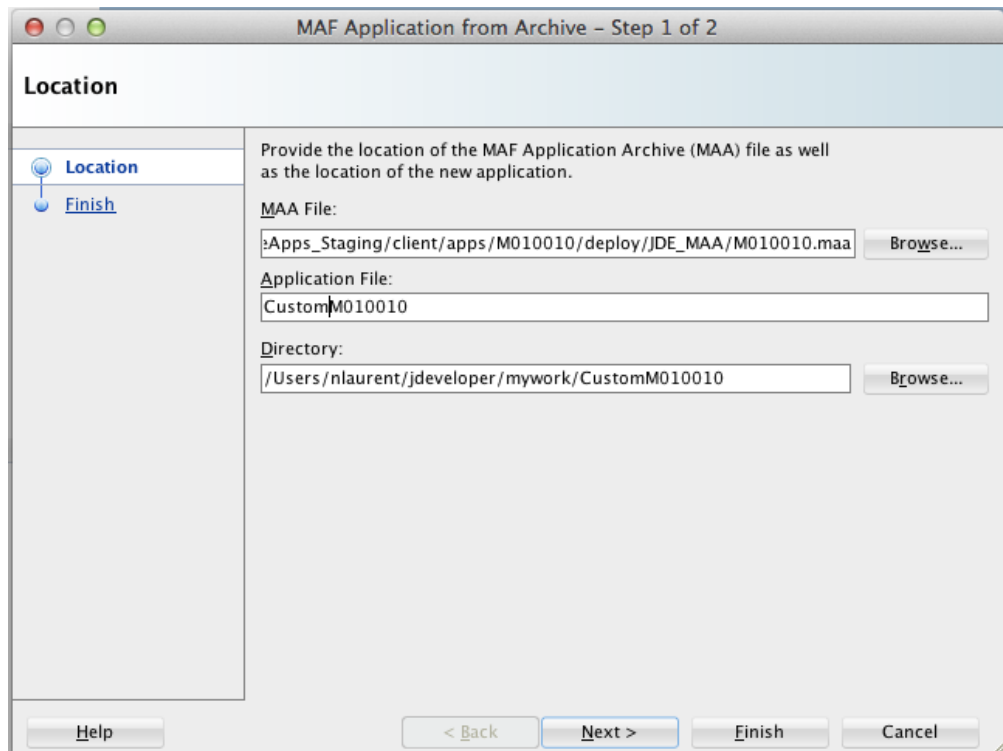
1. In JDeveloper, select the **File** menu > **New** > **Application**.



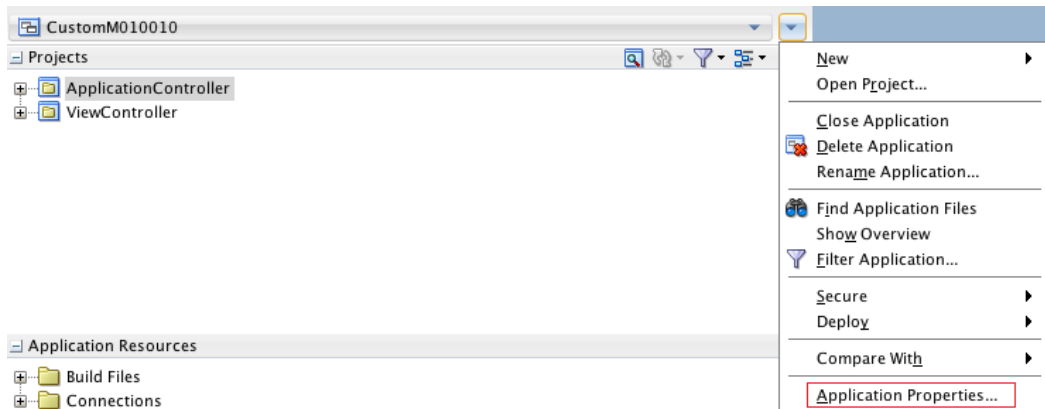
2. On New Gallery, select **MAF Application from Archive File (Applications)** and click **OK**.



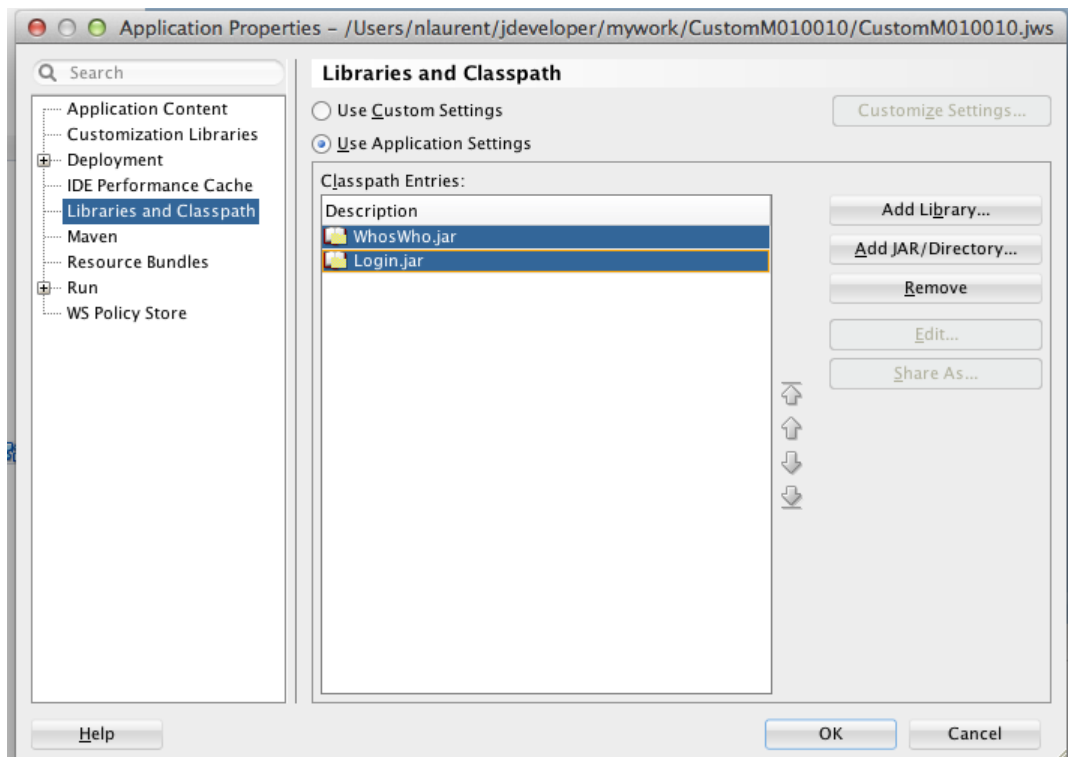
3. On Select MAA File to Import, select the .maa file that you want to customize, and then click **Open**.



4. On "MAF Application from Archive - Step 1 of 2," modify the name of the application in the Application File field.
5. Click **Next**, and then click **Finish**.



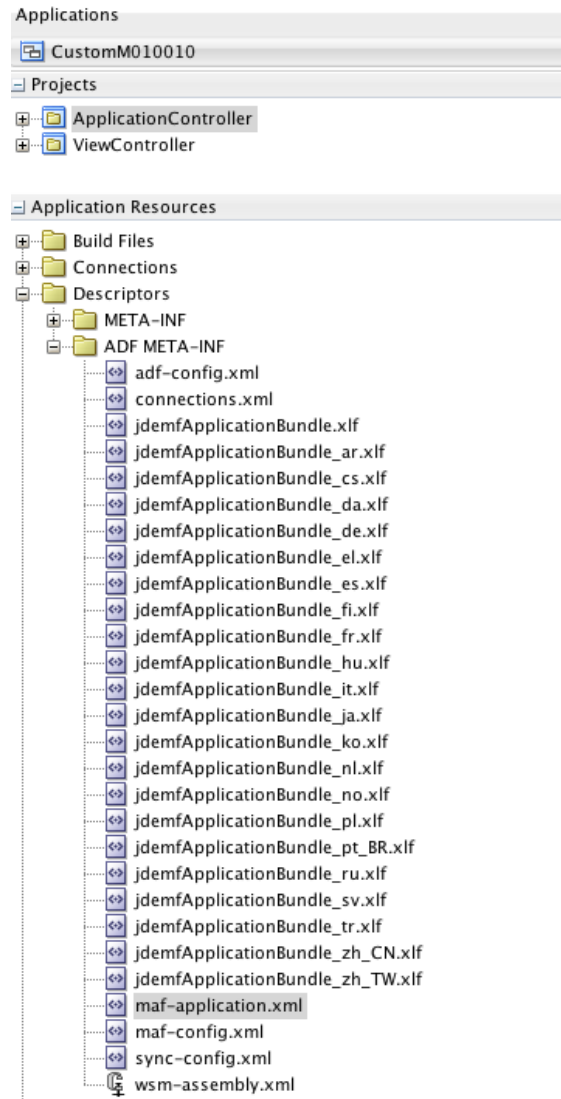
6. Click the drop-down menu next to the new application, and select **Application Properties**.
7. Navigate to **Libraries and Classpath**.



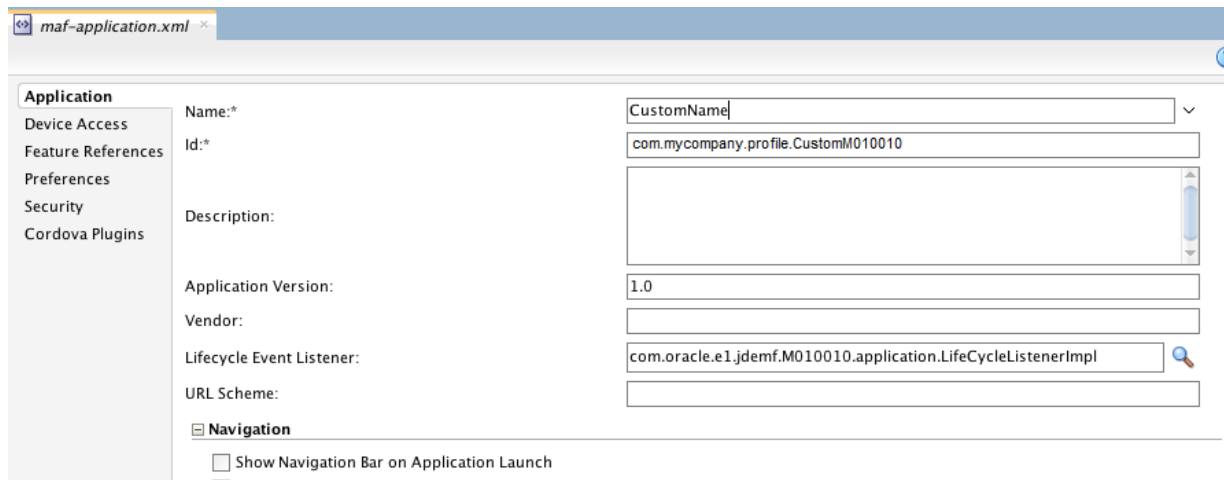
8. On Libraries and Classpath, select the entries listed, and then click the **Remove** button to delete them.
9. Click **Add JAR/Directory**.
10. On Add Archive or Directory, select the **ExternalLibs** folder.
11. Highlight all entries to add them, and then click **Open**.
This points both JAR files to the right location.
12. Click **OK** to save it.
13. Right-click **ApplicationController** and select **Project Properties**.

14. On Libraries and Classpath, click the check box next to JDEMobileFramework.jar, and then click the **Remove** button to remove it.
15. Select **Add JAR/Directory**.
16. On Add Archive or Directory, select the **ExternalLibs** folder, and then click **Open**.
17. Select **JDEMobileFramework.jar** and click **Open**.

This points the JDEMobileFramework.jar to the correct location.



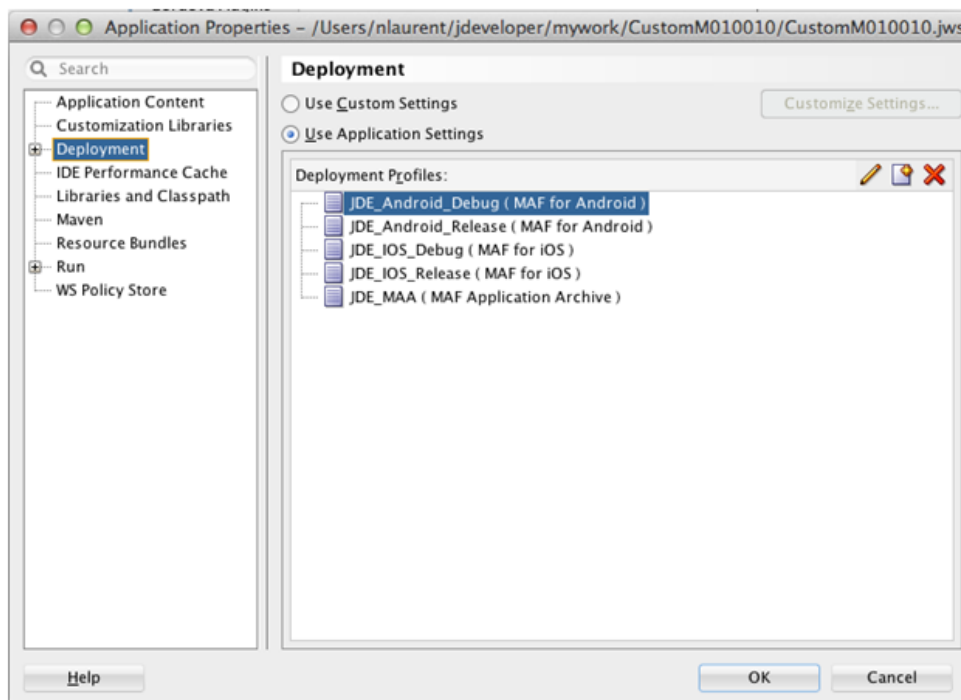
18. Under Application Resources, open the **maf-application.xml**.



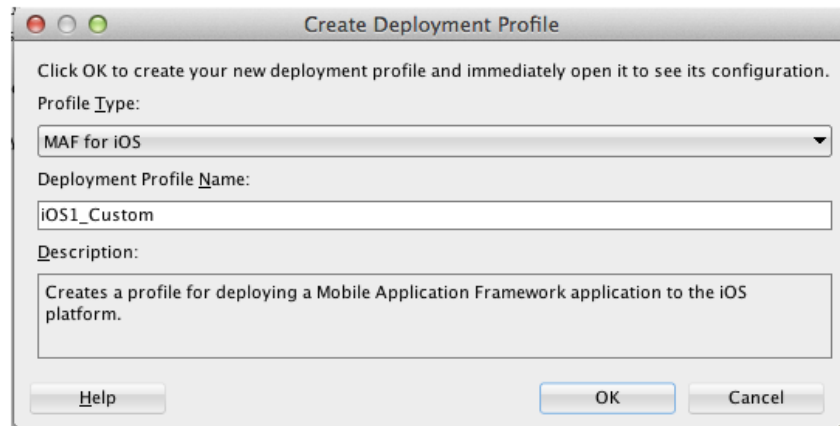
19. Change the Name and Id of the application. Use the ID associated with your Apple Distribution Profile.

This ensures that the application will not overwrite or interfere with the original application when deployed.

20. Create a new deployment profile that will use the new name and Id. To do so:
 - a. Right-click the application and select **Properties**.

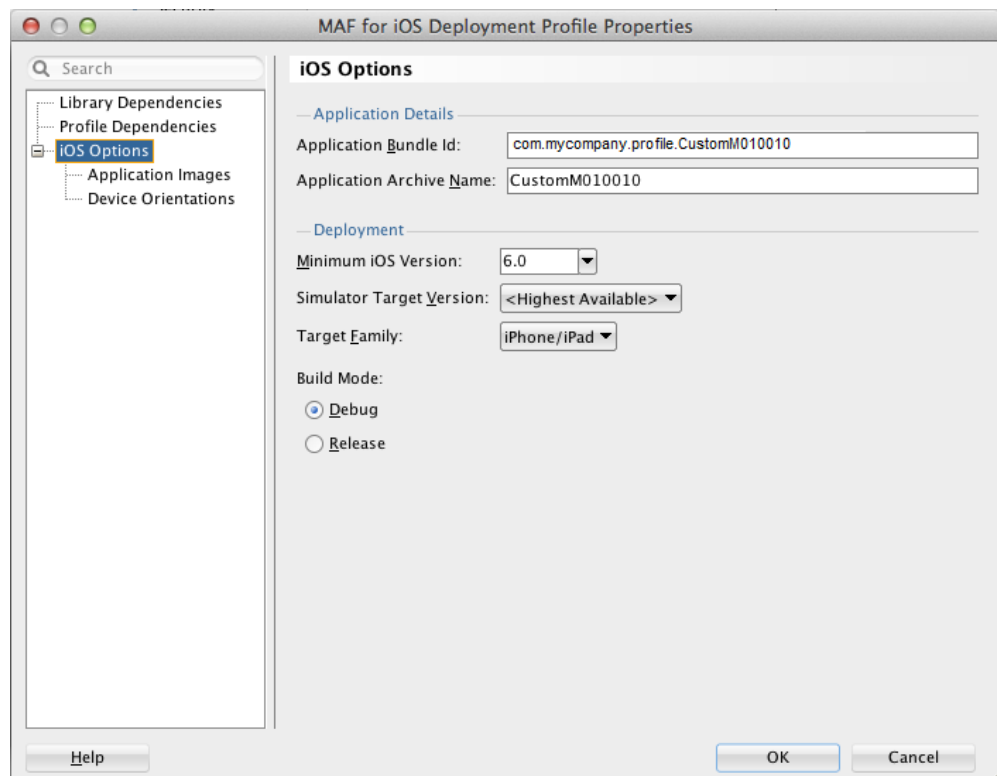


- b. On Application Properties, select **Deployment**, and then click the "new" icon.



- c. Select **MAF for iOS** (or **MAF for Android**) and give your profile a name, for example `iOS1_Custom`. Click **OK**.

The Profile Properties dialog box displays the application and deployment details:



- d. Click **OK**.

21. Use the new deployment profile to deploy and test the application in the simulator/emulator.

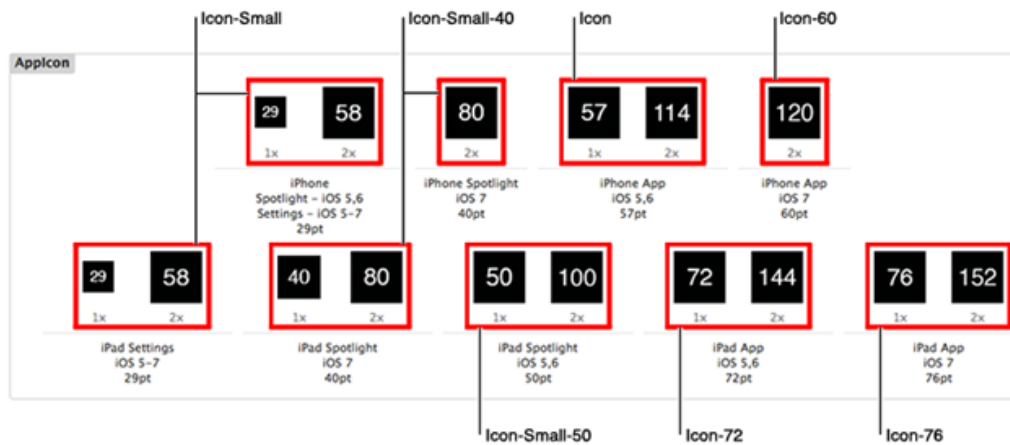
If simulator tests are successful, you can generate distribution packages to deploy the application to devices or stores.

B.4 Customization Options

This section describes the features you can customize in an MAA.

B.4.1 Customizing the Application Icons and Splash Screens

The application icon represents the application on the home screen of the device. The following diagram shows the icon size requirements for iOS 6 and iOS 7:

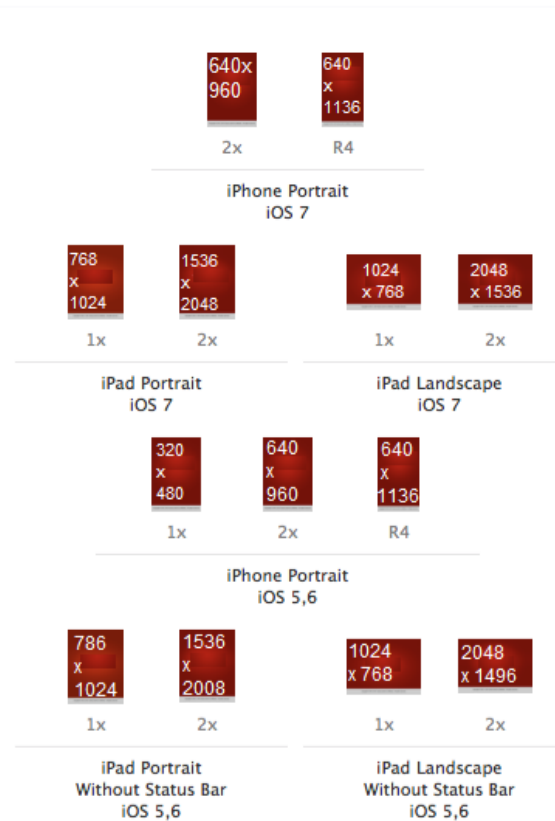


For Android, these are the size requirements for the application icon:

- XHDPI: 96 x 96px
- HDPI: 72 x 72px
- MDPI: 48 x 48px
- LDPI: 24 x 24px

The launch image or splash screen appears immediately after the application is started.

The following diagram shows the sizes that are required to override the launch image for iOS 6 and iOS 7:

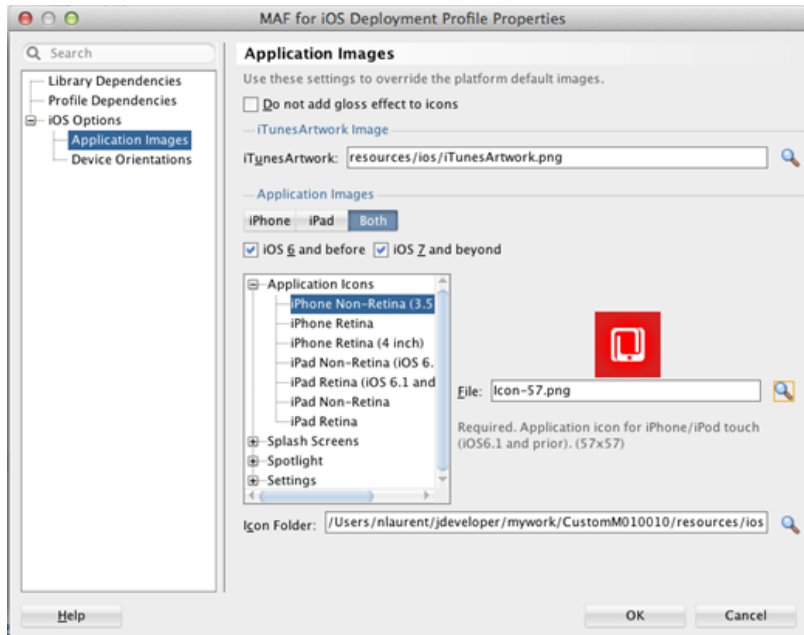


For Android, these are the size requirements for the application splash screen:

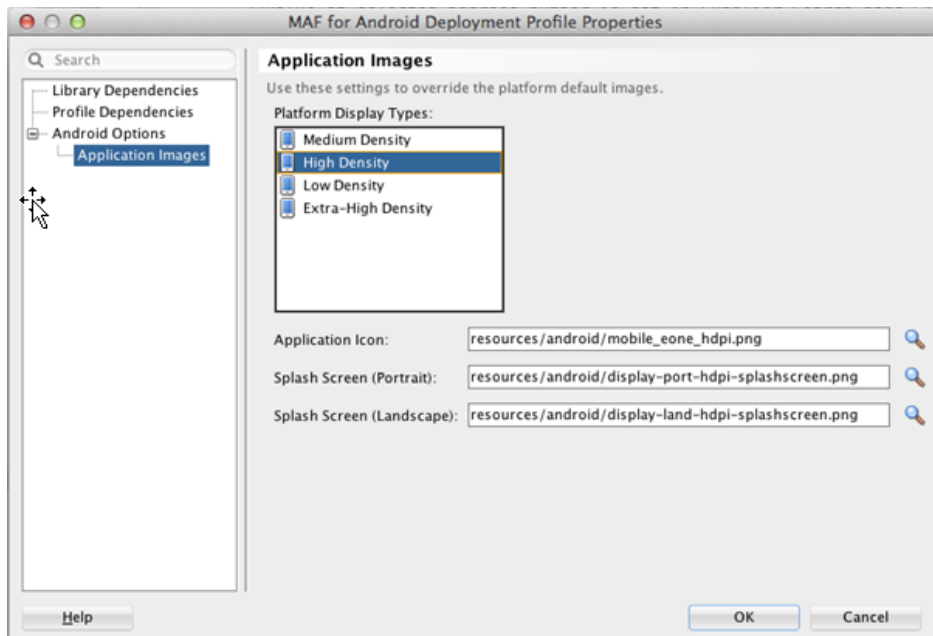
- LDPI: Portrait: 200 x 320px. Landscape: 320 x 200px.
- MDPI: Portrait: 320 x 480px. Landscape: 480 x 320px.
- HDPI: Portrait: 480 x 800px. Landscape: 800 x 480px.
- XHDPI: Portrait: 720 x 1280px. Landscape: 1280 x 720px.

The deployment profile contains all the application and launch screen configurations. Modify the definitions in the deployment profile to use your custom icons and deploy with that profile.

The following example shows the iOS deployment profile properties for application images:



The following example shows the Android deployment profile properties for application images:



B.4.2 Customizing the Brand Images

An Oracle JD Edwards logo image will appear on the configuration screen, login screen, and on error screens when there are connection issues. A second logo image appears on the springboard.

Do not use the logo images provided by Oracle; they are provided for example purposes only. If you do not insert an image, a question mark (?) appears where the image should be.

To override the login logo in your custom application, replace the image in the following location with an image of the same name and dimensions (242x87):

ViewController/public_html/images/jde_transparent_no_jde_small.png

To override the springboard logo in your custom application, replace the image in the following location with an image of the same name and dimensions (242x87).

ViewController/public_html/images/jde_transparent_springboard.png

Tip: For the springboard, you should use a transparent image that works well on a dark background.

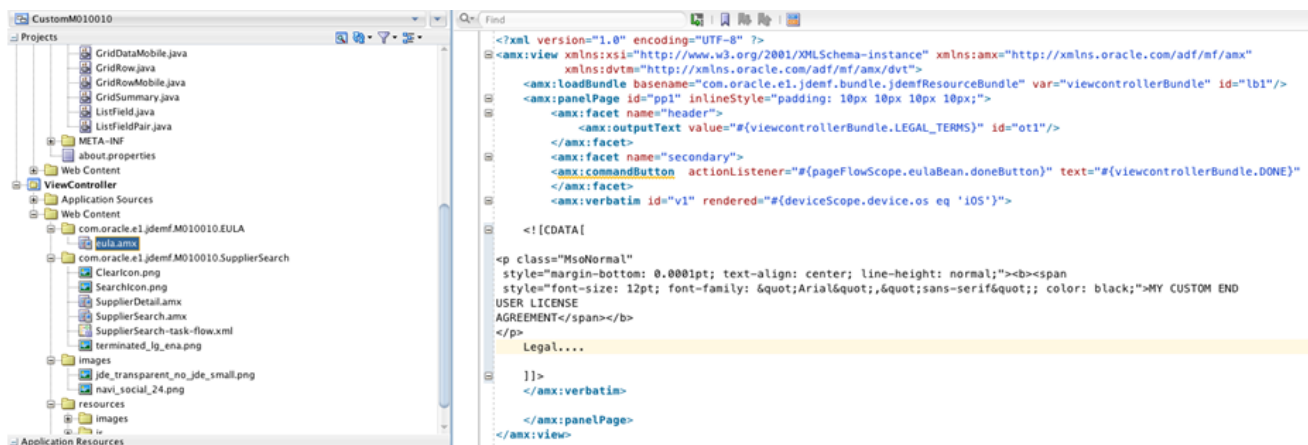
Additionally, there might be other logo icons depending on the specific application you are customizing. These icons are located in the ViewController/public_html folder. To determine which image you need to replace, you must view the amx page where it appears and determine the name and location of the image file.

B.4.3 Customizing the End User License Agreement (EULA)

Users can access the EULA from the Legal Terms link on the springboard. The EULA is included in the application in a feature ending in EULA. To change the content of the EULA, modify the eula.amx page within the EULA feature.

See "JD Edwards EnterpriseOne Mobile Archive Restricted Use Notice" in the *JD Edwards EnterpriseOne Licensing Information User Manual* for a list of terms that must be included in the EULA.

The delivered page includes HTML formatted text within a `<! [CDATA[` tag. You can use this structure to include your EULA if it is already formatted with HTML. Or you can design this EULA page just like any other amx page and use amx formatting, as shown in the following example:



B.4.4 Customizing the About Page

The About page is accessed from the About link on the springboard. The About page contains the following information:

- Application name and version (displayed on the General tab).
- Mobile application ID (displayed on the Advanced tab).

The information on the About page is configured in the mobile application and is stored in the about.properties file. In JDeveloper, you can open the about.properties file, which is included in the mobile download files from Oracle Software Delivery Cloud, and view the details of the About page.

Important: The mobile application ID is also part of the security configuration for the application. This ID corresponds to an Object Management Workbench (OMW) application object in EnterpriseOne. The security for the mobile application is inherited from the EnterpriseOne security configured for the OMW object of this name. If you modify this value in the mobile application, you must:

- Create a corresponding object in OMW.
- Use Security Workbench to configure application security for the new object.

B.4.5 Customizing the Pages

You can customize an amx page in the ViewController by reformatting, moving fields, hiding fields (`rendered=false`), and changing the look of the page in general.

You can change labels on the page by directly entering a new text value for the label or by modifying the associated translation string in the translation bundles. The bundles are located in the `com.oracle.e1.jdemf.bundle` package of the Application Controller.

important: When performing these types of customizations, do not remove any property listeners or action listeners, otherwise you risk compromising the application functionality.

B.5 Extension Options

This section describes how you can extend an MAA to:

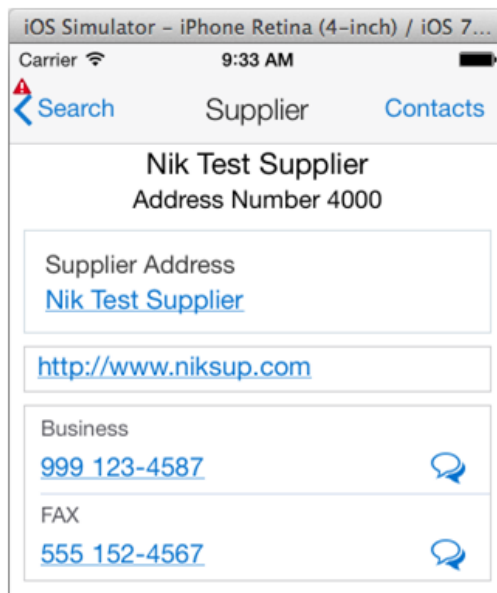
- Display additional data.
- Remove existing fields or data.
- Include new pages.

B.5.1 Displaying Additional Data

To display additional data in an MAA, you have to configure additional service calls to retrieve the data from EnterpriseOne. You use Java code to enable the MAA to perform additional service calls.

Example B-1 Adding Additional Data to the Supplier Search Mobile Application

This is an example of the address, URL, and phones details for a supplier in the Supplier Search application:



You can use Java code to add category code information from the EnterpriseOne Address Book record to this Supplier details page.

The Address Number is the key to this information. When a record is selected, the address number of that record is used to fetch the category codes. So in the SupplierSearch.amx page, a new setPropertyListener is added when a list item is selected. The new setPropertyListener is highlighted in **bold** in this example code:

```
<amx:.listView var="row" value="#{bindings.rowset1.collectionModel}"
fetchSize="#{bindings.rowset1.rangeSize}"
    inlineStyle="position:absolute; top:72px; bottom:0; left:0;
right:0; border-top:1px solid #BBBBBB;" id="lv1">
    <amx:listItem id="li1" action="Detail">
        <amx:tableLayout width="100%" id="tl2">
            <amx:rowLayout id="r13">
                <amx:cellFormat width="10px" rowSpan="2" id="cf7"/>
                <amx:cellFormat width="100%" height="28px" id="cf8">
                    <amx:outputText value="#{row.SAlphaName_
26.bindings.value.inputValue}" id="ot4" truncateAt="30"/>
                </amx:cellFormat>
            </amx:rowLayout>
            <amx:rowLayout id="r12">
                <amx:cellFormat width="100%" height="12px" id="cf5">
                    <amx:outputText value="#{viewControllerBundle1.ADDRESS_NUMBER}
#{row.mnAddressNumber_25.bindings.value.inputValue}"
                    styleClass="adfmf-listItem-captionText" id="ot3"
truncateAt="30"/>
                </amx:cellFormat>
            </amx:rowLayout>
        </amx:tableLayout>
        <amx:setPropertyListener id="spl4" from="#{bindings.searchName.inputValue
ne '' ? 'true' : 'false'}"
                                to="#{pageFlowScope.searchExecuted}"/>
        <amx:setPropertyListener id="spl5" from="#{row.rowKey}"
to="#{pageFlowScope.selectedRowKey}"/>
        <amx:setPropertyListener id="spl6" from="#{row.mnAddressNumber_
25.bindings.value.inputValue}" to="#{pageFlowScope.selectedABNumber}"/>
        <amx:actionListener id="all"
binding="#{bindings.processSelectedSupplier.execute}"/>
    </amx:listItem>
</amx:.listView>
```

```
</amx:listItem>
```

You must create objects for storing the category code data retrieved from the Address Book. Use the AIS Client Class Generator to generate the classes necessary. If you have not already installed and configured the AIS Client Class Generator, see [Installing the AIS Client Class Generator Extension for JDeveloper](#) and [Configuring the AIS Client Class Generator](#) in this guide.

When naming the package, make sure to use a custom package name in the configuration so there are no conflicts with existing classes from the original application:

Java Package

Category code 4 and 5 will be retrieved from the Address Book. The field IDs for these fields are 363 and 366, as shown here:

AIS Client Class Generator

AIS Server URL: <http://den60210jems.us.oracle.com:8474>
 JAS Server URL: default

Credentials

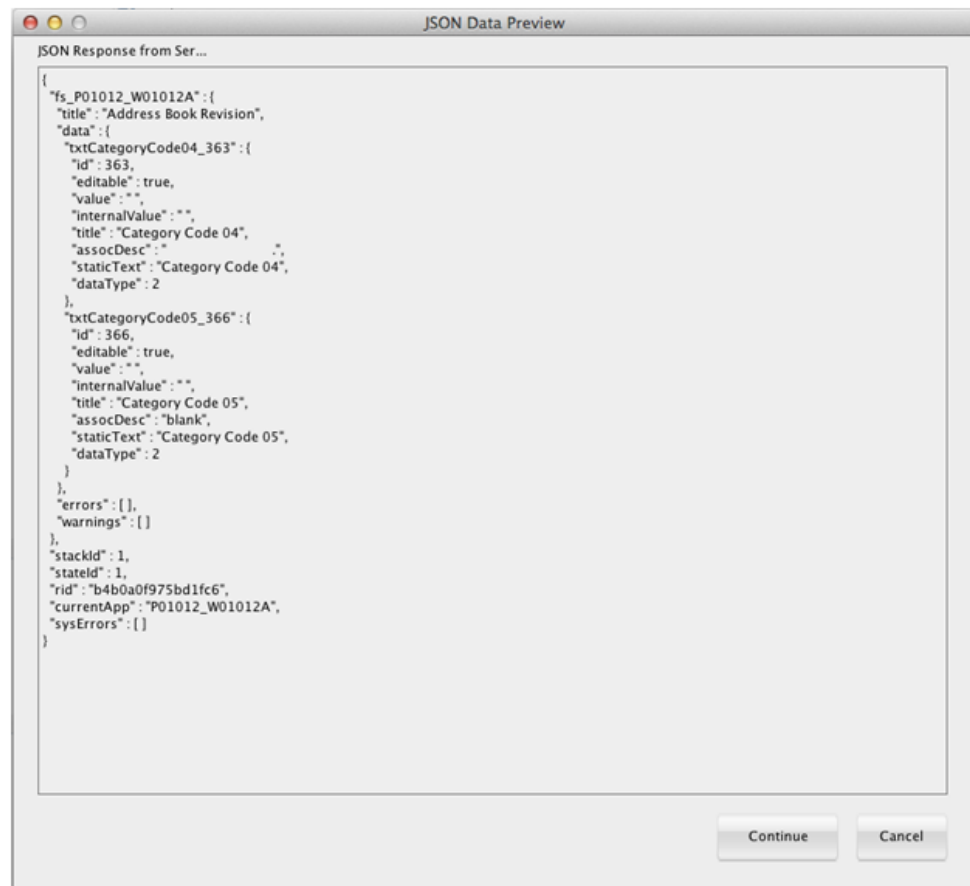
Username: Password:
 Environment: Role:

Parameters

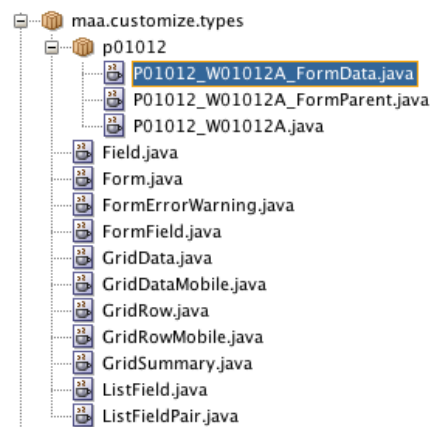
Application Name: e.g. P01012
 Form Name: e.g. W01012B
 Version: e.g. ZJDE0001
 MaxPageSize: e.g. 10
 ReturnControlIDs: e.g. 1[19,20]58|62
 FormInputs: e.g. (C,2);(D,3)
 FormServiceAction: e.g. R
 FindOnEntry: Executes FIND on entry.
 DemoMode: Returns a dummy grid row.

Generate for Mobile Applic... Preview JSON Data Keep JSON Files

After verifying the JSON includes category codes 4 and 5, generate the classes.



After refreshing the project, the classes are displayed, as shown here:



After the objects are created to hold the category code data and the selectedABNumber is set when the row is selected, the fetch for the category codes can be written. Add a new class in the ApplicationController project, in the application package. See the code example below for the content of the class. After the class is complete, create a Data Control from it.

```

package com.oracle.e1.jdemf.M010010.application;

import com.oracle.e1.jdemf.FormRequest;
import com.oracle.e1.jdemf.JDERestServiceException;
import com.oracle.e1.jdemf.JDERestServiceProvider;

```

```

import com.oracle.el.jdemf.M010010.formservicetypes.p04012.P04012_W04012D_
FormParent;
import maa.customize.types.p01012.P01012_W01012A_FormParent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.framework.api.JSONBeanSerializationHelper;
import oracle.adfmf.framework.exception.AdfException;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;
import oracle.adfmf.json.JSONObject;

public class ExtendAppDC {

    private P01012_W01012A_FormParent catCodeParent = new P01012_W01012A_
FormParent();
    private PropertyChangeSupport propertyChangeSupport = new
PropertyChangeSupport(this);

    public ExtendAppDC() {
        super();
    }

    public void setCatCodeParent(P01012_W01012A_FormParent catCodeParent) {
        P01012_W01012A_FormParent oldCatCodeParent = this.catCodeParent;
        this.catCodeParent = catCodeParent;
        propertyChangeSupport.firePropertyChange("catCodeParent",
oldCatCodeParent, catCodeParent);
    }

    public P01012_W01012A_FormParent getCatCodeParent() {
        return catCodeParent;
    }

    public void getCatCodesForSupplier(){

        //value of selected address number is set in Supplier Search page when the
row is selected.
        String selectedABNumber =
(String)AdfmfJavaUtilities.evaluateELEExpression("#{pageFlowScope.selectedABNumber}
");

        //call P01012_W01012A form to get category codes 4 and 5
FormRequest formRequest = new FormRequest();

        formRequest.setReturnControlIDs("21|363|366");
        formRequest.setFormName("P01012_W01012A");
        formRequest.setFormServiceAction(formRequest.ACTION_READ);
        formRequest.addToFISet("12",selectedABNumber);

        try{

            // For POST request, set data payload is header delimited with | and
service input class
            JSONObject jsonObject =
(JSONObject)JSONBeanSerializationHelper.toJSON(formRequest);
            String postData = jsonObject.toString();

            String response = JDERestServiceProvider.jdeRestServiceCall(postData,
"POST", "formservice");

            catCodeParent = (P01012_W01012A_

```

```

FormParent)JSONBeanSerializationHelper.fromJSON(P01012_W01012A_FormParent.class,
response);

        //remove the associated description when it is a .
if(catCodeParent.getFs_P01012_W01012A().getData().getTxtCategoryCode04_
363().getAssocDesc() != null &&
        catCodeParent.getFs_P01012_
W01012A().getData().getTxtCategoryCode04_363().getAssocDesc().trim().equals("."))
    {
        catCodeParent.getFs_P01012_
W01012A().getData().getTxtCategoryCode04_363().setAssocDesc("");
    }

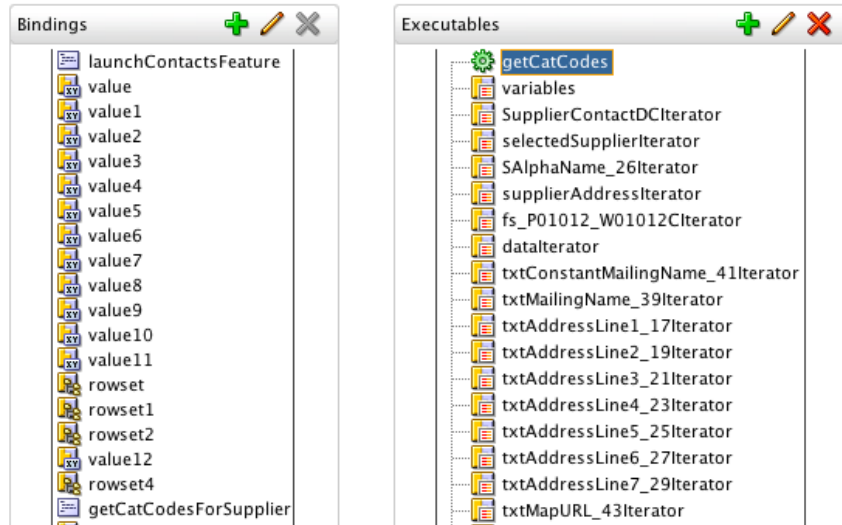
        if(catCodeParent.getFs_P01012_
W01012A().getData().getTxtCategoryCode05_366().getAssocDesc() != null &&
        catCodeParent.getFs_P01012_
W01012A().getData().getTxtCategoryCode05_366().getAssocDesc().trim().equals("."))
    {
        catCodeParent.getFs_P01012_
W01012A().getData().getTxtCategoryCode05_366().setAssocDesc("");
    }
    }
    catch (JDERestServiceException e)
    {
        JDERestServiceProvider.handleServiceException(e);
    }
    catch(Exception e)
    {
        throw new AdfException(e.getMessage(), AdfException.ERROR);
    }
}

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

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

```

Next, the `getCatCodesForSupplier()` method is invoked from the amx executable bindings, as shown here:



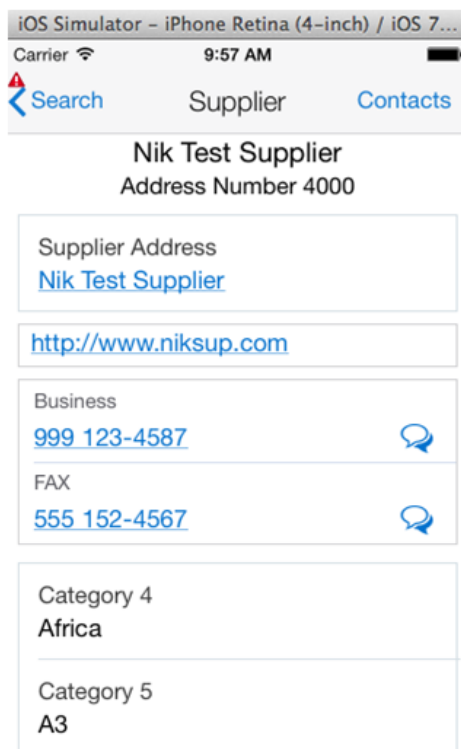
The two fields from the DataControl are added to the detail page in a `panelFormLayout` after the listview with the phones, as shown here:

```

</amx:listView>
  <amx:panelFormLayout labelPosition="topStart" fieldHalign="start">
    <amx:panelLabelAndMessage label="Category 4" id="plam2">
      <amx:outputText value="{bindings.assocDesc.inputValue}" id="ot3"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label="Category 5" id="plam3">
      <amx:outputText value="{bindings.assocDesc1.inputValue}" id="ot5"/>
    </amx:panelLabelAndMessage>
  </amx:panelFormLayout>

```

The associated description for each field in EnterpriseOne is used for each field value in the mobile application. The following example shows the result of customizing the mobile application to show the two category code values. The labels in this example are generic, you can make the label appropriate for what the value is in your customization:



B.5.2 Removing Data from Pages

For each field you want to remove, set the rendered property for that field to false in the SupplierDetail.amx page. The field will not appear in the mobile application, and the space for that field will be reclaimed as if the field was never there.

B.5.3 Adding New Pages

You can add new forms and navigation actions, as well as a new link or button to navigate to the new forms. Refer to the Oracle MAF documentation for these types of modifications:

<http://docs.oracle.com/middleware/mobile200/mobile/index.html>

