JD Edwards EnterpriseOne Tools

Deploying and Developing Oracle Application Development Framework (ADF) Applications

9.2

JD Edwards EnterpriseOne Tools
Deploying and Developing Oracle Application Development Framework (ADF) Applications

9.2

Part Number: E66067-14

Contents

Preface

Welcome to the JD Edwards EnterpriseOne documentation.

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 Information

For additional information about JD Edwards EnterpriseOne applications, features, content, and training, visit the JD Edwards EnterpriseOne pages on the JD Edwards Resource Library located at:

http://learnjde.com

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Boldface type indicates graphical user interface elements associated with an action or terms defined in text or the glossary.
Italics	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 examples, text that appears on a screen, or text that you enter.
> Oracle by Example	Indicates a link to an Oracle by Example (OBE). OBEs provide hands-on, step- by-step instructions, including screen captures that guide you through a process using your own environment. Access to OBEs requires a valid Oracle account.

1 Understanding This Guide

Understanding This Guide

This guide describes how to deploy and develop Oracle ADF applications for JD Edwards EnterpriseOne, which are referred to as EnterpriseOne ADF applications in this guide. The guide contains the following parts:

- Part I, "EnterpriseOne ADF Applications Overview" provides an overview of EnterpriseOne ADF applications, the environment required for running EnterpriseOne ADF applications, and an overview of EnterpriseOne ADF application development.

  Part I contains the following chapter:

  - *Chapter 2 - Introduction to Oracle Application Development Framework (ADF) Applications for JD Edwards EnterpriseOne*

- Part II, "EnterpriseOne ADF Application Deployment" describes how to set up an environment for deploying EnterpriseOne ADF applications and the EnterpriseOne ADF Container.

  Part II contains the following chapters:

  - *Chapter 3 - Setting Up the Environment for Deploying EnterpriseOne ADF Applications*
  - *Chapter 4 - Building and Deploying EnterpriseOne ADF Applications*
  - *Chapter 5 - Administering EnterpriseOne ADF Applications*

- Part III, "EnterpriseOne ADF Application Development Resources" describes how to use various tools that aid in the development of EnterpriseOne ADF applications. It also refers to the AIS capabilities and endpoints that enable EnterpriseOne ADF applications to access EnterpriseOne features and perform business processes in EnterpriseOne.

  Part III contains the following chapters:

  - *Chapter 6 - Understanding EnterpriseOne ADF Application Development*
  - *Chapter 7 - Prerequisites for Developing EnterpriseOne ADF Applications*
  - *Chapter 8 - Accessing EnterpriseOne Data from ADF Applications*
  - *Chapter 9 - Executing AIS Calls for Retrieving Data*
  - *Chapter 10 - EnterpriseOne ADF Container*
  - *Chapter 11 - Executing the Build Script from JDeveloper*

- In addition to these parts, *Appendix A - Creating a Sample EnterpriseOne ADF Application* and *Appendix B - Creating a Sample EnterpriseOne ADF Application for Data Service Aggregation Request (Release 9.2.0.2)* provide step-by-step instructions on how to create a sample EnterpriseOne ADF application.

2 Introduction to Oracle Application Development Framework (ADF) Applications for JD Edwards EnterpriseOne

Understanding Oracle ADF Applications for JD Edwards EnterpriseOne

EnterpriseOne applications provide robust features that enable users to perform a wide range of industry business processes and tasks. Using the tools provided with Oracle Application Development Framework (ADF), an add-on to Oracle JDeveloper, developers can design web applications for EnterpriseOne that contain a more focused set of features and fields based on a particular business process. When designed to perform a specific business process, these web applications (otherwise referred to as EnterpriseOne ADF applications) can increase the user experience while decreasing the risk of user error.

EnterpriseOne ADF applications cannot be deployed as standalone applications. An administrator must deploy them with the EnterpriseOne ADF Container application, which provides the login, about information, and other capabilities for deployed EnterpriseOne ADF applications. See *EnterpriseOne ADF Application Deployment* for more information on how to deploy the EnterpriseOne ADF Container and EnterpriseOne ADF applications.

An administrator must associate deployed EnterpriseOne ADF applications to tasks in EnterpriseOne, which users can then access through an EnterpriseOne task view.

Prebuilt EnterpriseOne ADF Application

Oracle provides prebuilt EnterpriseOne ADF applications that you can download from the JD Edwards Update Center on My Oracle Support: `https://updatecenter.oracle.com`.

In the Update Center, search on **EnterpriseOne ADF** in the Type field for a complete list of the applications available for download. Each download contains a readme file with a link to the documentation on how to use the EnterpriseOne ADF application.

For instructions on how to deploy EnterpriseOne ADF applications, see *EnterpriseOne ADF Application Deployment* in this guide.

Runtime Architecture for EnterpriseOne ADF Applications

The following servers are required to run EnterpriseOne ADF applications:

- Oracle WebLogic Server installed with ADF runtime, referred to as an ADF Server.

All Oracle ADF applications, including EnterpriseOne ADF applications, must be deployed to an Oracle WebLogic Server with ADF runtime.

- EnterpriseOne Application Interface Services (AIS) Server.

  The data displayed in EnterpriseOne ADF applications is retrieved by making calls to EnterpriseOne through the EnterpriseOne AIS Server.

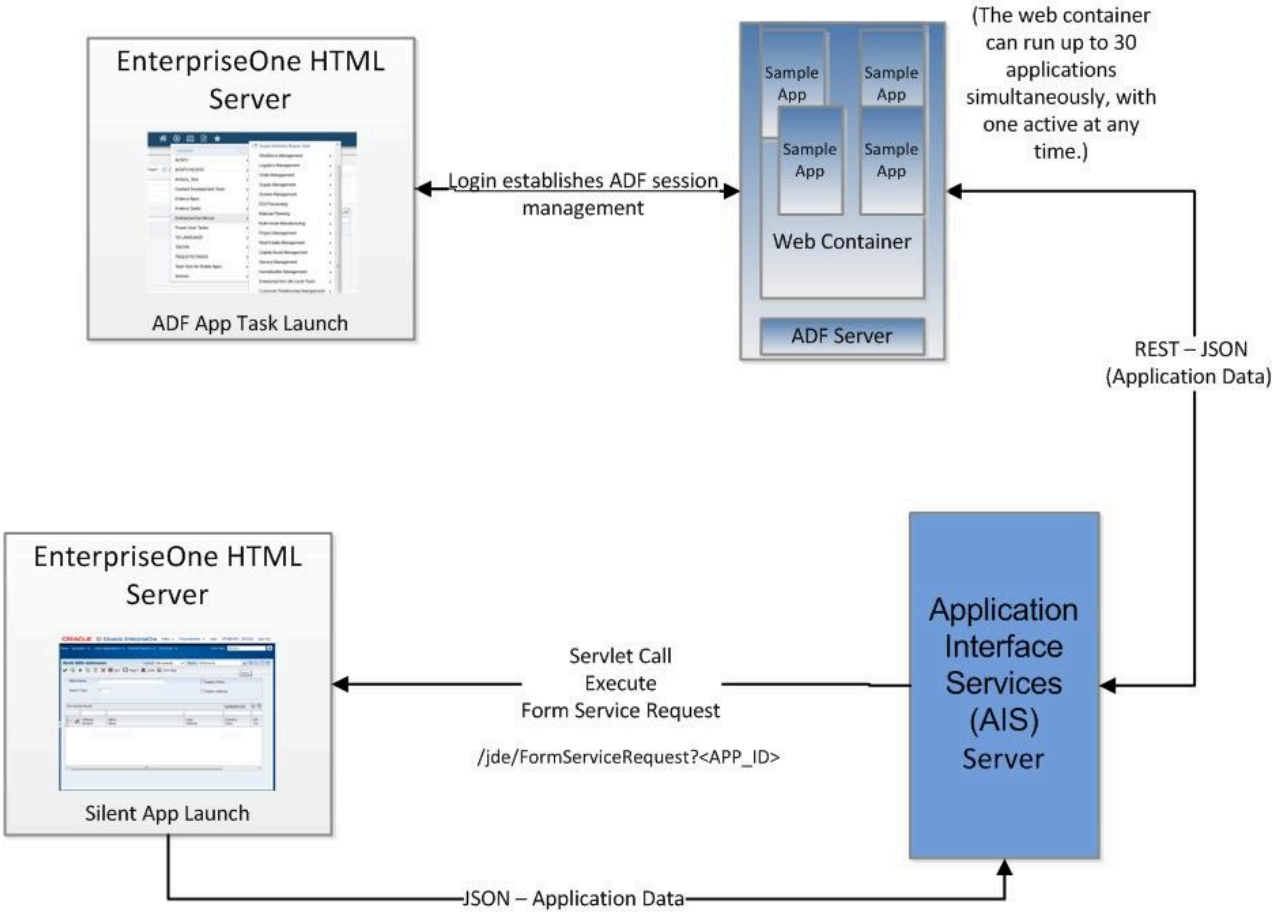
EnterpriseOne ADF applications expose a light interface to manage EnterpriseOne data. The EnterpriseOne AIS Server provides a JSON over REST interface to EnterpriseOne applications and forms through the EnterpriseOne HTML Server. The EnterpriseOne AIS Server exposes this interface to enable communication between EnterpriseOne ADF applications and EnterpriseOne applications.

> **Note:** The EnterpriseOne HTML Server also executes some Java processing; therefore, it is sometimes referred to as the Java Application Server (JAS). The terms HTML Server and JAS Server are synonymous.

The EnterpriseOne AIS Server includes support for JSON representation of form service requests so EnterpriseOne ADF applications can easily consume responses the EnterpriseOne AIS Server is providing. The EnterpriseOne AIS Server submits these EnterpriseOne ADF application requests to the EnterpriseOne HTML Server.

The EnterpriseOne AIS Server maintains sessions when EnterpriseOne ADF applications make requests. You can configure the session timeouts for the EnterpriseOne AIS Server through Server Manager. By default, the ADF Server timeout setting is 20 minutes and the AIS Server timeout setting is 30 minutes. If you adjust these settings, the timeout for the ADF Server must remain less than the timeout for the AIS Server.

This graphic shows the runtime architecture that enables the communication between EnterpriseOne ADF applications and EnterpriseOne.



The following list describes how the communication is established between an EnterpriseOne ADF application and EnterpriseOne when a user launches an EnterpriseOne ADF application:

1. When a user logs in to EnterpriseOne, a connection identified by a handshake ID is established between the EnterpriseOne AIS Server, EnterpriseOne HTML Server, and ADF Server (Oracle WebLogic Server with ADF runtime).
2. When an EnterpriseOne ADF application task is launched within EnterpriseOne, the EnterpriseOne HTML Server sends a request to the EnterpriseOne ADF Container (Web Container) to start the EnterpriseOne ADF application, passing the handshake ID.
3. The EnterpriseOne ADF application uses the handshake ID to access the AIS token created for the current session and executes AIS service calls.
4. The EnterpriseOne AIS Server validates the token for each request, forwards the requests to the EnterpriseOne HTML Server to fulfill, and returns the responses to the EnterpriseOne ADF application.

Sample Application

The appendixes in this guide provide step-by-step instructions on how to create a sample EnterpriseOne ADF application. As you read about the JDE ADF Helpers and other features in Part III, "EnterpriseOne ADF Application Development Resources" of this guide, you can refer to the steps in the appendixes to see examples of how the features are used in the development of an EnterpriseOne ADF application.

3 Setting Up the Environment for Deploying EnterpriseOne ADF Applications

Verify 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`.

On the Certifications tab, search for JD Edwards EnterpriseOne ADF Foundation to locate the certifications.

Setting Up an EnterpriseOne AIS Server

Deploy and configure an EnterpriseOne AIS Server. The AIS Server enables communication between AIS client applications (including EnterpriseOne ADF applications) and EnterpriseOne.

See *"Configuring the Application Interface Services Server" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for more information.

Setting Up Oracle WebLogic Server with ADF Runtime

To set up and configure an Oracle WebLogic Server with ADF runtime, perform the following tasks:

- *Download Oracle ADF Runtime*
- *Install ADF Runtime Libraries on Oracle WebLogic Server*
- *Create Database Schemas Using the Repository Creation Utility (RCU)*
- *Extend the Oracle WebLogic Server Domain for ADF Runtime*
- *Create a Managed Server*
- *Apply the JRF Template to the New Managed Server*

Download Oracle ADF Runtime

Access the download from the following location:

`http://www.oracle.com/technetwork/developer-tools/adf/downloads/index.html`

From the Application Development Runtime drop-down menu, select the release that corresponds to the MTRs, and then click the **Download File** button.

The appropriate Oracle ADF Runtime Distribution will download.

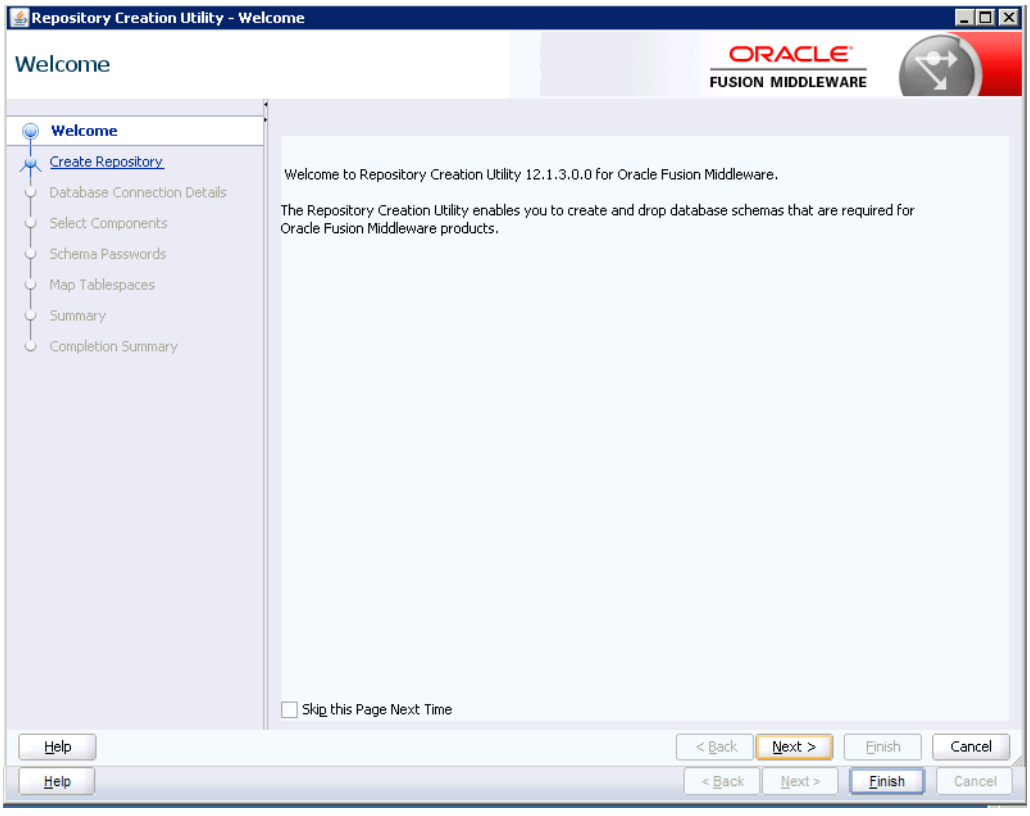
Install ADF Runtime Libraries on Oracle WebLogic Server

1. Access the `java\bin` folder.
2. Execute the java-jar fmw_12.2.1.3.0_infrastructure.jar file, which launches the Repository Creation Utility.
3. On the Welcome screen, click **Next**.
4. On Installation Location, in the Oracle Home field, enter the existing Oracle Home, for example `Oracle\Middleware`, and then click **Next**.
5. On Installation Type, select the **Fusion Middleware Infrastructure** option and click **Next**.
6. On Prerequisite Checks, click **Next** after the prerequisite check completes.
7. On Installation Summary, review the summary and click **Install**.
8. On Installation Progress, after the installation completes, click **Next**.
9. On Installation Complete, review the installation status and click **Finish**.

Create Database Schemas Using the Repository Creation Utility (RCU)

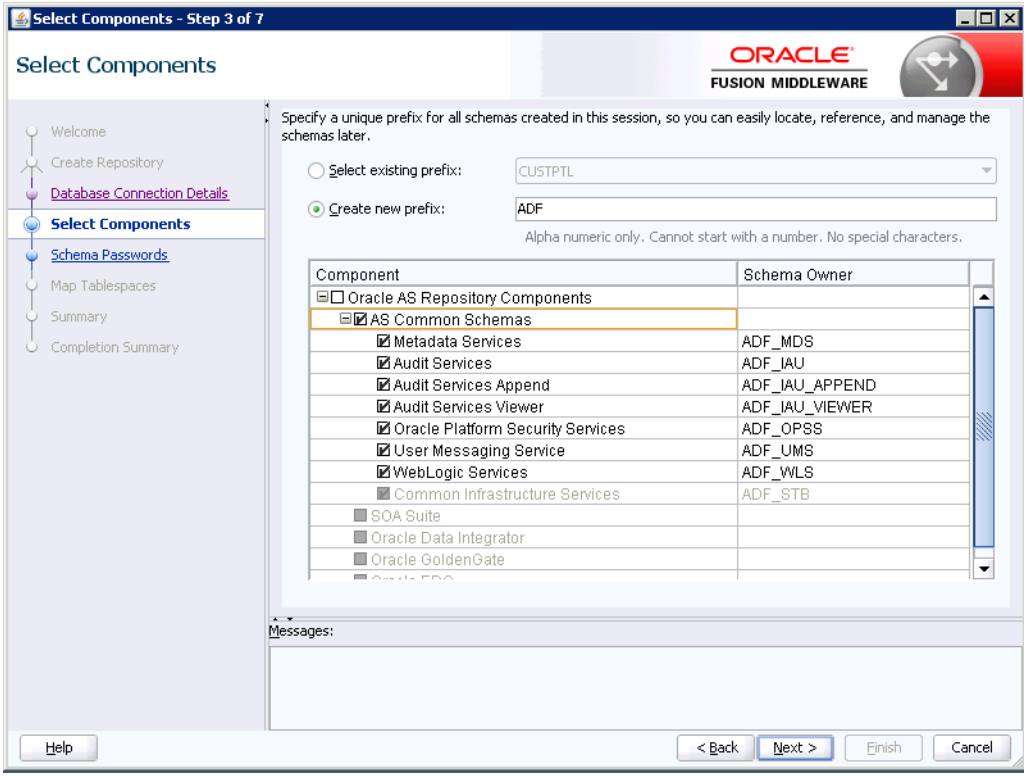
To create the database schemas using RCU:

1. Execute the rcu.bat(sh) from the `Oracle_Home\oracle_common\bin` folder.
   This launches the Repository Creation Utility.



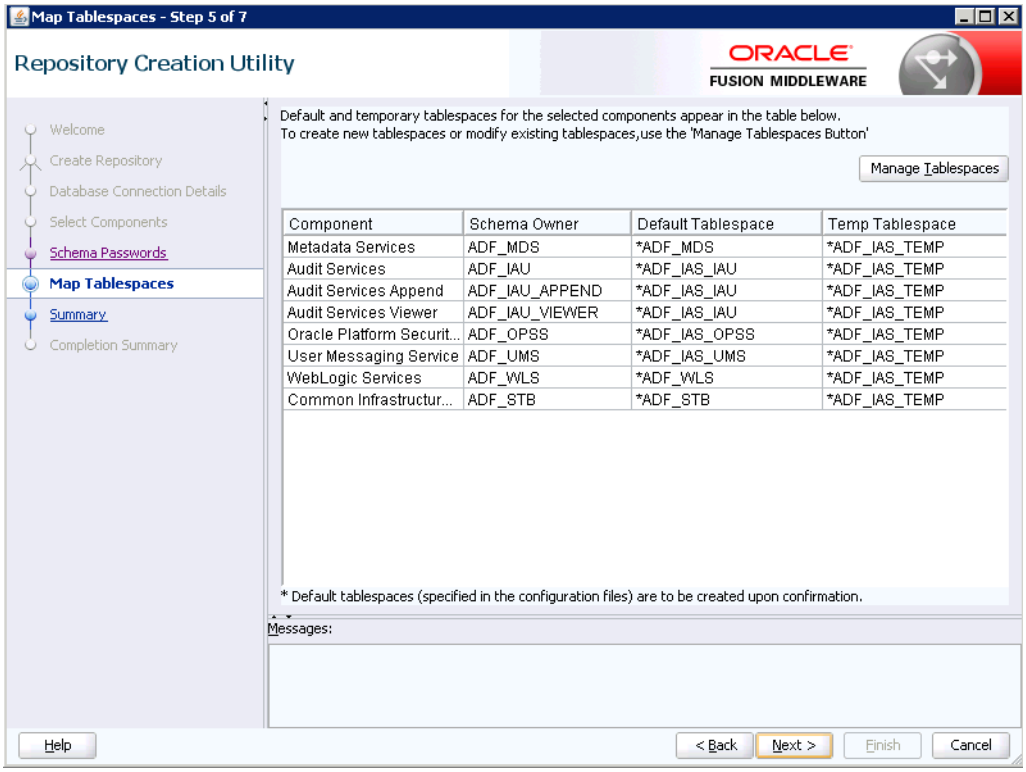
2. On the Welcome screen, click **Next**.

3. On Create Repository, select the **Create Repository** option and then select the **System Load and Product Load** option.
4. On Database Connection Details, complete the following fields to enter the database connection information:
    - **Database Type**
    - **Host Name**
    - **Port**
    - **Service Name**
    - **Username**
    - **Password**
    - **Role**
5. Click **Next**.
6. Click **OK** after the prerequisites check completes.



7. On Select Components, perform these steps:
    a. Select the **Create new prefix** option and enter a new schema prefix.
    b. In the Component column, click the AS Common Schemas check box.
    c. Click **Next**.
    d. After a second prerequisites check, click **OK**.
8. On Schema Passwords, select the **Use same passwords for all schemas** option and then enter a password for all schemas.

9. Click **Next**.



10. On Map Tablespaces, review the tablespaces information and click **Next**.
11. In the Confirmation dialog box, click **OK** to confirm tablespaces creation.
12. On Summary, click **Create**.
13. On Completion Summary, review the summary and then click **Close**.

Extend the Oracle WebLogic Server Domain for ADF Runtime

Because your EnterpriseOne ADF applications will be on the same domain as your ADF libraries, you need to extend the Oracle WebLogic Server domain for ADF runtime.

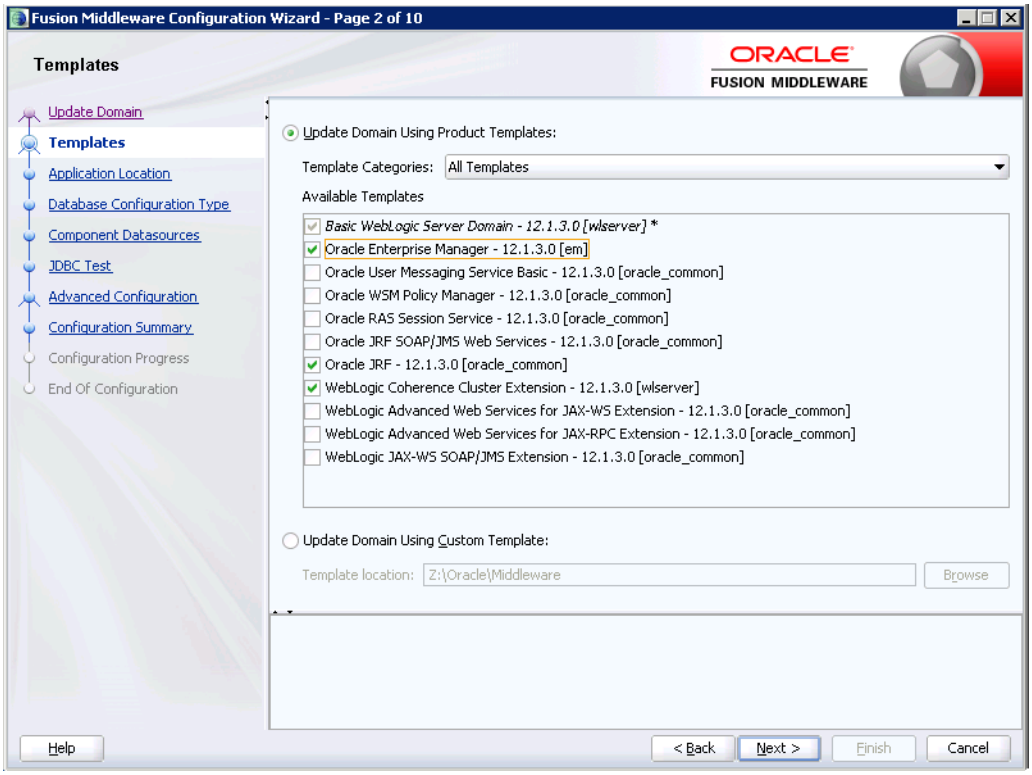
> **Note:** If you have not yet created the Oracle WebLogic Server domain, you can choose "Create a new domain" instead of "Update an existing domain."

To extend the Oracle WebLogic Server domain for ADF runtime:

1. Execute the config.cmd(sh) from Oracle_Home\oracle_common\common\bin folder.
2. On Configuration Type, click the **Update an existing domain** option.

   If you select the **Create a new domain** option, the following steps will be similar.

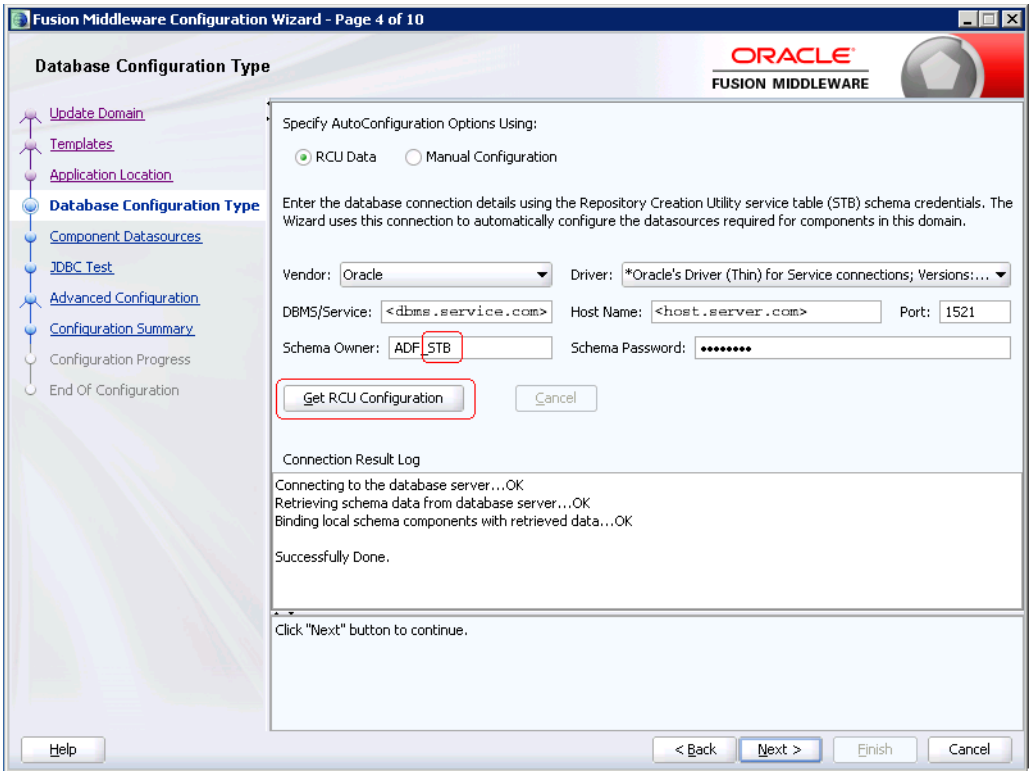
3. Click **Next**.



4. On Templates, select the **Oracle Enterprise Manager** check box.

   The other two check boxes are automatically selected.

5. On Application Location, verify the Domain location, and then click **Next**.



6. On Database Configuration Type, enter the Database Configuration.

   Be sure to use the *prefix*_STB schema.
7. Click **Get RCU Configuration** to verify the connection.
8. Click **Next**.
9. On Component Datasources, review the Data Source Components and verify that the connection info is correct.
10. Click **Next**.
11. On JDBC Test, click **Test Selected Connections**.
12. After the test completes successfully, click **Next**.
13. On Advanced Configuration, click **Next** without selecting any options.
14. On Configuration Progress, review the configuration summary, and then click **Update**.
15. When the upgrade process completes, click **Next**.
16. On Configuration Success, click **Finish** to exit the installer.

Create a Managed Server

To create a managed server:

1. Launch the WebLogic Server Administration Console.
2. Click **Server**, and then click **Lock and Edit**.
3. Click **New** to create a Managed Server.

   Oracle WebLogic Server 12.2.1 automatically applies the JRF template; no further steps are required. With Oracle WebLogic Server 12.1.3, you must apply the JRF template as described in *Apply the JRF Template to the New Managed Server*.

> **Note:** If you have Server Manager agent installed and have registered the Oracle WebLogic Server instance to your Server Manager, you can create the J2EE container from Server Manager.

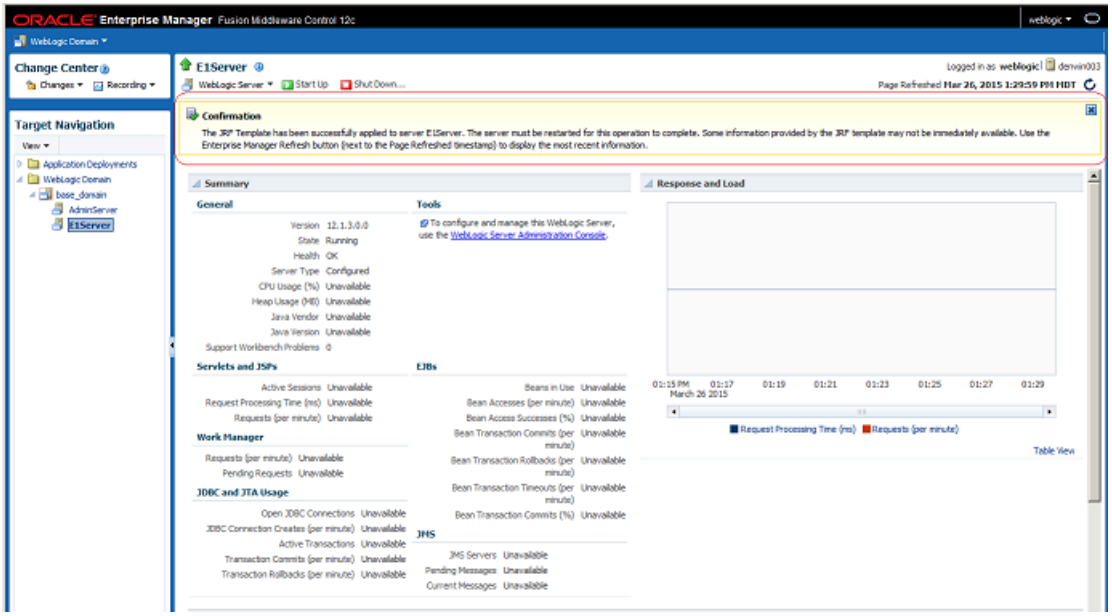
Apply the JRF Template to the New Managed Server

> **Note:** For Oracle WebLogic Server 12.2.1, the JRF template is automatically applied to the new server as long as the managed server was created before extending ADF runtime. Therefore, you can disregard the following instructions. However, if the managed server was created after, then you must apply the JRF template manually as described here.

To apply the JRF template to the new managed server:

1. Sign on to Oracle Enterprise Manager:

   `http://<server>:<port>/em`
2. Select the Managed Server.
3. Click the JRF Template.



4. Review the Confirmation message.

   The server is ready for deploying EnterpriseOne ADF applications.

4 Building and Deploying EnterpriseOne ADF Applications

Building and Deploying EnterpriseOne ADF Applications

Starting with EnterpriseOne Tools 9.2.1.2, you can use Server Manager to deploy EnterpriseOne ADF applications. See *"Install an Oracle Application Development Framework (ADF) Server Instance" in the JD Edwards EnterpriseOne Tools Server Manager Guide* , which describes how to deploy EnterpriseOne ADF applications to an ADF Server instance.

If you are on a release prior to EnterpriseOne Tools 9.2.1.2, you have to manually deploy EnterpriseOne ADF applications. See *Appendix C - Manually Building and Deploying EnterpriseOne ADF Applications* in this guide for more information.

5 Administering EnterpriseOne ADF Applications

Using Proxy Applications to Administer EnterpriseOne ADF Applications

Prior to Release 9.2.1, each EnterpriseOne ADF application that you deploy in EnterpriseOne must be associated with a proxy EnterpriseOne application. An administrator uses the proxy application to set up processing options, versions, and security for the EnterpriseOne ADF application. You configure security on the proxy application through the Security Workbench.

To launch an EnterpriseOne ADF application from an EnterpriseOne proxy application, a Solution Explorer task of type "ADF Application" must first exist that specifies both an EnterpriseOne proxy application name and an ADF task flow ID. Optionally, the proxy application version can also be specified on the task; otherwise, ZJDE0001 is used as the default version. When the Solution Explorer task is launched, the proxy application name, version, jargon code, and ADF task flow ID from the task are sent to the JDEADFContainer. If the ADF task flow is found, the ADF application launches inside an available region of the JDEADFContainer web application.

You can create different versions of the proxy application associated with an EnterpriseOne ADF application.

You can define a different set of processing options for each version of an EnterpriseOne ADF application. For more information, see *Understanding Processing Options for ADF Applications* in this guide.

Using External Forms to Administer EnterpriseOne ADF Applications (Release 9.2.1)

With Release 9.2.1, you can attach your EnterpriseOne ADF application to an external form in Form Design Aid. The external form provides users with a direct link to the EnterpriseOne ADF application. This means it can be a task, within a Composed EnterpriseOne Page, on a menu, linked to from an EnterpriseOne Page, or accessed through Fast Path.

External forms are intended to replace the use of the proxy application. All functionality associated with the proxy application, including processing options, static translated text for ADF labels, and security, are also included with the application containing the external form.

Using this new form type has the following benefits:

- When creating the external form, you set the External Application Type to "Adf", and then instead of specifying the ADF task flow ID on a task type, you can specify this task flow ID in the External Application property.
- Similar to the proxy application concept, you can include any required processing options and translatable text in the application that includes the external form, so the EnterpriseOne ADF application can perform the necessary AIS requests to retrieve this data.
- You can specify a data structure for the external form, so a calling form can pass data values to the EnterpriseOne ADF application when the external form is called using a form interconnect.

- The EnterpriseOne ADF application can be launched directly from Fast Path if the external form is the entry point to the application. It can also be launched from Fast Path by specifying Pxxxxx|Wxxxxx.

Applications using an external form to launch an EnterpriseOne ADF application should be named beginning with a "P" rather than an "E."

For more information on creating external forms, see *"Understanding External Forms" in the JD Edwards EnterpriseOne Tools Form Design Aid Guide .*

> **Note:** Any existing proxy applications will continue to function correctly to launch an EnterpriseOne ADF application from a menu task, but with Release 9.2.1 and forward, you should utilize the new external form type to launch EnterpriseOne ADF applications.

Configuring Mobile Devices for ADF Applications (Release 9.2.0.2)

In order for an Apple iPad to properly display EnterpriseOne ADF applications, you must change the **Block Cookies** setting to "Always Allow." This setting can be found in the Settings application under Safari, Privacy & Security Settings.

6 Understanding EnterpriseOne ADF Application Development

Overview

Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java Platform, Enterprise Edition (Java EE) standards and open-source technologies to simplify and accelerate implementing service-oriented applications. Oracle ADF is an extension to JDeveloper.

With Oracle ADF, you can create Oracle ADF web applications for JD Edwards EnterpriseOne (also referred to as EnterpriseOne ADF applications) that interact with standard or customized EnterpriseOne applications. Oracle ADF enables you to retrieve data from custom EnterpriseOne applications by using existing business logic within your EnterpriseOne applications.

Oracle provides JDE ADF Helpers that include APIs and tools that you can use with Oracle ADF to create EnterpriseOne ADF applications. See the *JDE ADF Helpers* section for more information.

You should refer to the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework as a companion to this guide. The "*Introduction to Building Fusion Web Applications with Oracle ADF*" chapter in this guide provides a description of the architecture and key functionality of Oracle ADF to build web applications. When reviewing this guide, remember the following about EnterpriseOne ADF application development:

- EnterpriseOne ADF applications are created utilizing ADF Task Flows.
- EnterpriseOne ADF applications are packaged in ADF Library JAR files and built and deployed to run in the EnterpriseOne ADF Container.
- You include methods in EnterpriseOne ADF applications to access features provided by the EnterpriseOne ADF Container, features that enable you to launch an EnterpriseOne menu task, launch an EnterpriseOne application, and display error and warning messages using the EnterpriseOne ADF Container's message popup dialog.

JDE ADF Helpers

To expedite the development of ADF applications for EnterpriseOne, Oracle provides additional tools referred to as JDE ADF Helpers. JDE ADF Helpers include:

- AIS_Client.jar
  The AIS_Client.jar contains the AIS Client Java API, which provides classes and methods that enable EnterpriseOne ADF applications 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 foundational classes that are required to consume EnterpriseOne data returned by AIS.

- E1UserSession.jar

  The E1UserSession.jar exposes classes that EnterpriseOne ADF applications use to access both the LoginEnvironment object for the AIS server and additional features provided by the EnterpriseOne ADF Container web application. The LoginEnvironment object is especially important, because it is required for all AIS service requests within the EnterpriseOne ADF application.

To see an example of how to develop an EnterpriseOne ADF application using these utilities, refer to *Creating a Sample EnterpriseOne ADF Application* in this guide.

The Data Model

The data model for EnterpriseOne ADF applications includes a data control class for executing AIS services and foundation classes that hold data returned in the AIS responses. These foundation classes are specific to the EnterpriseOne application forms accessed by an EnterpriseOne ADF application and are created by the AIS Client Class Generator. You typically include the form classes in the Model project with the data control class.

See *"Using the AIS Client Class Generator" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for more information.

Form Service Requests

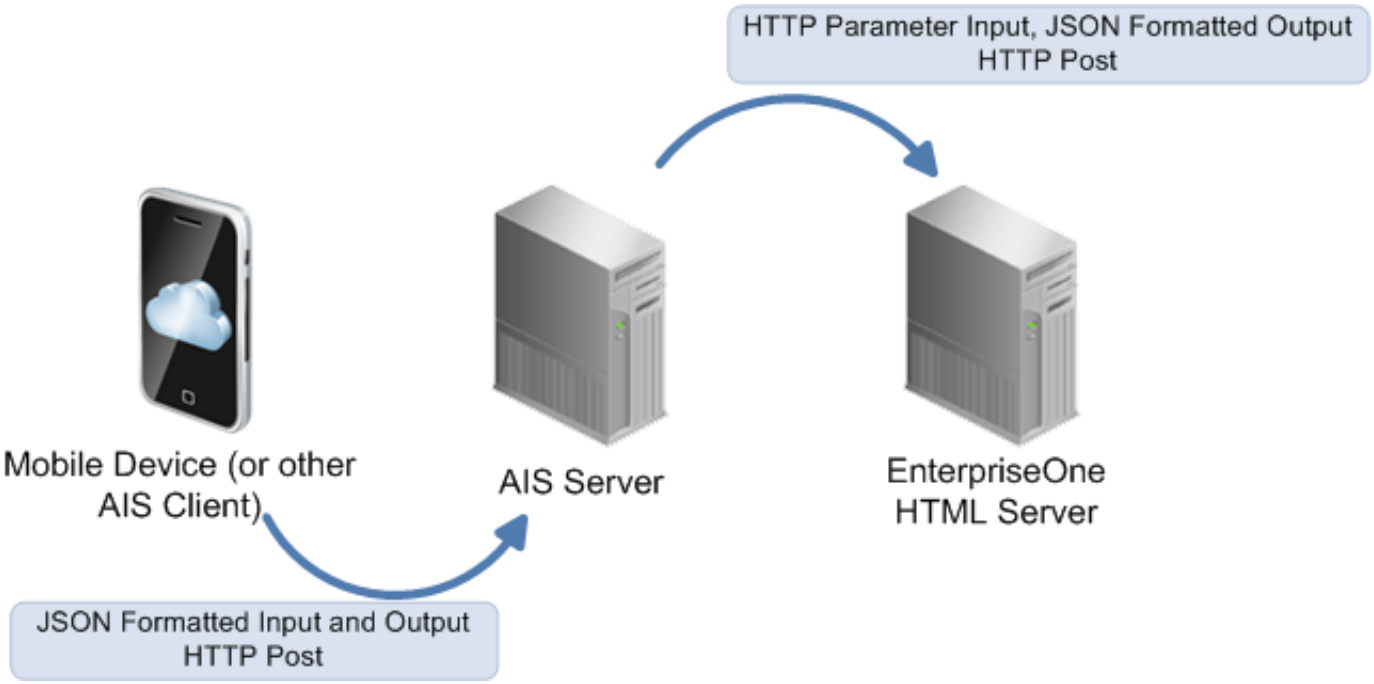
AIS Server calls used to retrieve data from forms in the EnterpriseOne web client are referred to as form service requests. EnterpriseOne ADF applications use form service requests to interact with EnterpriseOne web client forms. Form service requests, formatted as REST service calls that use POST, contain 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
```

EnterpriseOne Rest Services Interface

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



This illustration shows the communication between the EnterpriseOne ADF application deployed with the EnterpriseOne ADF Container (Web Container) and the AIS Server. It also shows the AIS Server's communication with the EnterpriseOne HTML Server, where the EnterpriseOne forms run and the data is gathered. All communication with the AIS Server is in the form of JSON formatted strings.

You can make REST calls directly to the AIS Server without using the AIS Client Java API. 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.

General Process Flow for Developing EnterpriseOne ADF Applications

The following list describes the high-level tasks involved in creating an EnterpriseOne ADF application. It also provides links to sections in this guide that discuss each task in more detail. Some of the links take you to *Creating a Sample EnterpriseOne ADF Application* which provides step-by-step instructions on how to create a sample EnterpriseOne ADF application, which you can use as a model for creating your own EnterpriseOne ADF application.

1. Complete the prerequisites as described in *Prerequisites for Developing EnterpriseOne ADF Applications* in this guide.
2. Create a **new ADF web application** in JDeveloper.

   See the "*Creating a New EnterpriseOne ADF Application*" topic.
3. Add the "**about.properties**" file.

   See the "*Supporting About Information*" topic.

4. Change the default skin from Skyros to Alta if necessary.

   See the "*Skinning*" topic.
5. Create the **EnterpriseOne proxy application** and version or with Release 9.2.1, create an **external form**.

   See the "*Understanding Processing Options for ADF Applications*" topic.
6. Generate **form service classes** using the AIS Client Class Generator extension for JDeveloper.

   See the "*Generating Form Service Foundation Classes for the Data Model*" topic.
7. Create the **Data Control class.**

   See the "*Creating the Data Control Class*" and "*Populating Address Book List and Creating the Data Control*" topics.
8. **Retrieve the EnterpriseOne proxy** application's processing option values.

   See the "*Retrieving Processing Options for ADF Applications*" topic and the example in the "*Populating Address Book List and Creating the Data Control*" topic.
9. **Retrieve Jargon text** from EnterpriseOne for component labels in EnterpriseOne ADF application.

   See *"Jargon Service" in the JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide* .
10. **Perform AIS service requests** to retrieve data from or insert/update data to EnterpriseOne.

    See *Executing AIS Calls for Retrieving Data* and the example in the "*Populating Address Book List and Creating the Data Control*" topic.
11. **Create a bounded task flow.**

    See the "*Creating a Task Flow*" topic.
12. **Create view page fragments** and bind the Data Control to ADF UI components.

    See the "*Creating Address Book Search Panel*" and "*Creating Address Book Detail Panel*" topics.
13. **Create and run test page** for EnterpriseOne ADF application.

    See the "*Creating the ADF Test Page*" and "*Running the Test Page*" topics.
14. **Find the ADF Bounded Task Flow ID.**

    See the "*Finding the ADF Bounded Task Flow ID*" topic.
15. **Package ADF bounded task flow** into ADF Library JAR file.

    See the "*Creating ADF Library JAR File*" topic.
16. **Deploy application JAR** in the ADF Library WAR on the ADF Server.

    See *Building and Deploying EnterpriseOne ADF Applications* for details.
17. **Create an EnterpriseOne menu task** to launch the EnterpriseOne ADF application.

    See *"Creating a Task" in the JD Edwards EnterpriseOne Tools Solution Explorer Guide* .

7 Prerequisites for Developing EnterpriseOne ADF Applications

Prerequisites for Developing EnterpriseOne ADF Applications

- Set up an environment for deploying EnterpriseOne ADF applications as described in *Setting Up the Environment for Deploying EnterpriseOne ADF Applications* in this guide.
- Install JDeveloper. See the "Oracle JDeveloper Get Started" documentation for installation instructions:

  `http://docs.oracle.com/middleware/12211/jdev/index.html`
- If you have not already done so, download the latest release of E1_ADF_Foundation_x.x from the JD Edwards Update Center (`https://updatecenter.oracle.com`). To locate the download, use the Type field to search on EnterpriseOne ADF.

  In addition to script files for building and deploying EnterpriseOne ADF applications, this download includes the E1UserSession.jar that provides helper classes for developing EnterpriseOne ADF applications.
- Download the latest AIS Client Java API 2.x.x.

  Locate the download on the JD Edwards Update Center on My Oracle Support at `https://updatecenter.oracle.com` . Use the Type field to search on "EnterpriseOne ADF." The download is available in a package titled AIS_Client_Java_API_2.x.x.

  The download contains the following files:

  - AIS_Client.jar. This contains the AIS Client Java API, which provides classes and methods that enable EnterpriseOne ADF applications to manage (create, read, update, delete) data in EnterpriseOne through REST services.
  - Jackson 2.x.x library, which includes the jackson-databind, jackson-core, and jackson-annotations jar files.
  - AISCGE 12c_v2.x.zip. This is the AIS Client Class Generator extension for JDeveloper. See *"Using the AIS Client Class Generator" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for instructions on how install this extension.
- Starting with Tools Release 9.2.1.2, if you are deploying the ADF Server using Server Manager and plan to include custom ADF applications on your ADF Server, you need to bundle the scf-manifest.xml file with your custom ADF application so that Server Manager recognizes it as an ADF application component.

  For a sample scf-manifest.xml file, see *Modifying ADF Source Code*. You can also use the scf-manifest.xml files bundled with the ADF Applications on the Update Center as a reference.

8 Accessing EnterpriseOne Data from ADF Applications

Creating an EnterpriseOne ADF Application Connection to the AIS Server

An EnterpriseOne ADF application's bounded task flow includes handshakeId as one of its required input parameters. For a list of parameters, see *#unique_43/unique_43_Connect_42_BCEFHHCG*. The application data control uses this handshakeId parameter to determine whether the ADF application is running locally on JDeveloper's Integrated WebLogic Server or executing inside the EnterpriseOne ADF Container (JDEADFContainer).

The following example shows the standard code you should add to your data control class to connect to the AIS Server:

Example: AIS connection code for Address Book Data Control Class

```
public class AddressBookDC
{
    // Session variables.
    private E1UserSessionBean userBean;
    private LoginEnvironment loginEnv;
    private boolean runningInJDEADFContainer = true;

    // Hard coded connection values.
    private static final String AIS_SERVER = "http://host:port";
    private static final String USER_NAME = "username";
    private static final String PASSWORD = "password";
    private static final String ROLE = "role";
    private static final String ENVIRONMENT = "environment";
    private static final String DEVICE = "E1ADFApps";

    public AddressBookDC()
    {
        String handshakeId = (String)
 ADFContext.getCurrent().getPageFlowScope().get("handshakeId");
        if (handshakeId == null || (handshakeId != null && handshakeId.length() == 0))
        {
            runningInJDEADFContainer = false;
            userBean = new E1UserSessionBean(AIS_SERVER, USER_NAME, PASSWORD, ENVIRONMENT,
 ROLE, DEVICE);
        }
        else
        {
            // Only initialize application About properties when running in the
 JDEADFContainer.
            E1AdfUtils.intializeAppInstance("/com/oracle/e1/E01012/");
        }

        loginEnv = E1AdfUtils.getLoginEnvironment(userBean);
```

```
        if (loginEnv != null)
        {
            // Specify all AIS capabilities required by data control.
            List<String> reqCapabilities = loginEnv.getRequiredCapabilities();
            reqCapabilities.add("processingOption");

            retrievePOValues();
        }
    }
}
```

When the EnterpriseOne ADF application is running locally, the handshakeId input parameter will be null. As a result, the EnterpriseOne ADF application creates a direct connection to the AIS Server using hard coded AIS credentials and the E1UserSessionBean class included in E1UserSession.jar. E1AdfUtils will then use the E1UserSessionBean connection to create a LoginEnvironment object, which is used in all subsequent service requests to the AIS Server.

> **Note:** You must initialize the AIS credential constants with values specific to your AIS Server when testing on JDeveloper's Integrated WebLogic Server. Before deploying an EnterpriseOne ADF application to an ADF Server, you should remove these hard coded values from the data control class, so connection issues can be identified at runtime.

When an EnterpriseOne ADF application launches from an EnterpriseOne menu and executes within the JDEADFContainer, a valid handshakeId value is passed to the application's bounded task flow as an input parameter. The data control will then inherit the LoginEnvironment object maintained by the JDEADFContainer, which was created after the user signed into EnterpriseOne and a connection was established between the EnterpriseOne HTML Server, EnterpriseOne AIS Server, and the JDEADFContainer.

Consuming Form Interconnect Values from an External Form (Release 9.2.1)

You can define a form data structure for external forms. External forms support passing data into the ADF application via the data structure. Currently, you can only send data into the external form, you cannot receive data values back.

For ADF, the form interconnect structure is a task flow parameter, called "formDataStructure". You can get the object value of this parameter into an object using this API:

Example: E1 Form Data Structure API

```
E1FormDataStructure formDataStructure = (E1FormDataStructure)
 ADFContext.getCurrent().getPageFlowScope().get("formDataStructure");
```

See Table A-1 in *Creating a Task Flow* for more information.

You can get the values out of the structure by ID, and save them to variables. The getValue(class) method will return the value as the type requested. You still must be aware of the type of data you expect in each form interconnect or you may have casting exceptions.

Example: Consuming Form Interconnect Values

```
if (formDataStructure != null && formDataStructure.getMembers() != null)
{
      E1DataStructureMember member = formDataStructure.getMember(4);
      String alphaName = null;
      try
      {
            if (member != null)
            {
                  alphaName = member.getValue(String.class);
            }
      }
      catch (ClassCastException e)
      {
            //handle the exception
      }
}
```

Form interconnect values are only passed if the calling application or Composite Application Framework layout is configured to pass values to the ADF application. Only the values mapped in the call to the external form will be included in the formDataStructure member list in ADF. So you must provide null checking in case an expected value is not there. Only call the getValue() method on an E1DataStructureMember that is not null.

9 Executing AIS Calls for Retrieving Data

Introduction

Use the AIS Client Java API to develop EnterpriseOne ADF applications that invoke AIS services on the AIS Server. The AIS Client Java API provides classes and methods to perform form service requests, retrieve processing options and jargon, and interact with text and photo media objects. For more information, see:

- *"Performing AIS Form Service Calls" in the JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide* . This guide describes the AIS services that you can invoke and provides examples of API code for invoking the services.
- *JD Edwards EnterpriseOne Application Interface Services (AIS) Client API Reference* located at:

  `http://docs.oracle.com/cd/E53430_01/EOTJP/index.html`

  This reference guide provides descriptions of the AIS Client Java API classes and methods.

Understanding the JD Edwards EnterpriseOne ADF Container

The JD Edwards EnterpriseOne ADF Container (JDEADFContainer) is a web application that executes EnterpriseOne ADF applications created as bounded task flows and packaged in ADF Library JAR files. See *EnterpriseOne ADF Container* for information about how to configure EnterpriseOne ADF applications to use the EnterpriseOne ADF Container.

Understanding AIS Server Capabilities

The AIS Server exposes various capabilities on which client applications may or may not depend. If your EnterpriseOne ADF application requires a certain capability, you must first identify the capability used by your application and then verify that the capability is provided by the AIS Server.

You can access the capabilities that are currently available on your AIS Server using the following URL:

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

You can find a description of all AIS Server capabilities in the *"AIS Server Capabilities and Services" section in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* .

Each AIS capability used by your EnterpriseOne ADF application only needs to be added to the required capability list once, so you should perform this setup in your data control's class constructor after the LoginEnvironment object is available.

Example: Defining Used Capabilities in the ADF Application

```
public void DataControlDC()
{
    String handshakeId = (String)
 ADFContext.getCurrent().getPageFlowScope().get("handshakeId");
    if (handshakeId == null || (handshakeId != null && handshakeId.length() == 0))
    {
        userBean = new E1UserSessionBean(AIS_SERVER, USER_NAME, PASSWORD, ENVIRONMENT,
 ROLE, DEVICE_NAME);
    }
    else
    {
        // Initialize application About properties when running in the container.
        E1AdfUtils.intializeAppInstance("/com/oracle/e1/E0801/");
    }

    loginEnv = E1AdfUtils.getLoginEnvironment(userBean);

    // Load required capabilities.
    if (loginEnv != null)
    {
        List<String> reqCapabilities = loginEnv.getRequiredCapabilities();
        reqCapabilities.add(AISClientCapability.GRID);
        reqCapabilities.add(AISClientCapability.PROCESSING_OPTION);
    }
}
```

Service requests for a required capability can throw a CapabilityException, so the request needs to be embedded within a try-catch block. When you execute a service request for a capability that is not provided by the AIS Server, control will transfer to the CapabilityException catch block, which you can use to perform alternate logic in place of the service request. Alternatively, you can use the AISClientCapability.isCapabilityAvailable(LoginEnvironment loginEnv, String capability) method to verify that a required capability exists before executing a service request. This also gives you the option to add alternate logic when the capability is not available.

Example: Capability Exception Handling Methods

```
private void retrieveJargonLabels()
{
    try
    {
        if (AISClientCapability.isCapabilityAvailable(loginEnv,
 AISClientCapability.JARGON))
        {
            JargonRequest jargonReq = new JargonRequest(loginEnv, jargonCode);
            jargonReq.addDataItem("STRT");
            jargonReq.addDataItem("DRQJ");

            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
 jargonReq,
                                JDERestServiceProvider.POST_METHOD,
                                JDERestServiceProvider.JARGON_SERVICE);
            JargonResponse jargonResp = loginEnv.getObjectMapper().readValue(response,
 JargonResponse.class);
```

```
            if (jargonResp != null)
            {
                // Process jargon response.
            }
        }
        else
        {
            // Perform alternate logic for missing AIS capability.
        }
    }
    catch (CapabilityException e)
    {
        // Perform alternate logic for missing AIS capability, like notifying the user or
 populating
        // values/list from alternate source.
    }
    catch (Exception e)
    {
        System.out.println(e);
    }
}
```

Using a REST Services Client to Interact with AIS

You can make REST calls directly to the AIS Server without using the AIS Client Java API. 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.

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: Acceptable Output 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.domainexample.com",
   "jasPort": "8412",
   "jasProtocol": "http",
   "defaultEnvironment": "DV910",
   "defaultRole": "*ALL",
   "displayEnvironment": true,
   "displayRole": true,
   "displayJasServer": false,
   "defaultJasServer": "http://jasserver.domainexample.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: Acceptable Input for the tokenrequest Service in JSON


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

Example: Acceptable Output for the token request Service in JSON Response


```
{
   "username": "jde",
   "environment": "JDV910",
   "role": "*ALL",
   "jasserver": "http://jasserver.domainexample.com:port",
   "userInfo":    {
      "token": "044kbwMICIsL8P4jttMj/VtpnEhrSK17i7fa2q8V
+hVDlc=MDE4MDEwNjA1NTMwODkwMzQ1ODY0MTkyUkVTVGNsaWVudDE0MTI2MDg3MDgzODY=",
      "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": "W3XmCk8k6vhb3H-dwxSdBpCGZWAb7kh5D4gzemFoqoFqcoWgwqF_!-1217528743!
1412608708388"
}
```


Form Request Attributes

The following list contains a description of the form request attributes:

- **maxPageSize**. (optional) Maximum number of rows to return to the grid. If you do not specify a value, then AIS will return a maximum of 100 rows. The maximum number you can enter for this setting is 500.

  Starting with EnterpriseOne Tools 9.2.1, there is no maximum limit for rows returned to the grid. However, a maximum value is required. If you want to retrieve all records, enter `No Max` for this attribute.

  The value in the "DB Fetch Limit in the Web Runtime Interactivity" setting in the Web Runtime Interactivity section of the EnterpriseOne HTML Server configuration file (jas.ini) supersedes the maxPageSize setting. The "DB Fetch Limit in the Web Runtime Interactivity" setting is typically set to a higher value (2000 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.
- **returnControlIDs**. (optional) Indicates which form and grid fields the service passes back. Grid is usually 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: Form Request

```
{
    "token":
 "044RnByWbW3FbLzpxWjSg55QzZWguAGnYqUNMlyB30IgyU=MDE5MDA2ODg0MDE5NjYwNzM1ODcyNDA5NlNvYXBVSTEzODc0ODc4OTEzNTc=",
    "maxPageSize": "10",
    "returnControlIDs": "1[19,20]58|62",
    "version": "ZJDE0001",
    "formInputs": [
        {
            "value": "E",
            "id": "2"
        }
    ],
    "formActions": [
        {
            "value": "E",
            "command": "SetQBEValue",
            "controlID": "1[50]"
        },
        {
            "value": "Al*",
            "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"
```

```
}
```

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 ADF 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 making the following changes 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://dnnlaurentvm1:8888",
   "environment":"DV910",
   "role":"*ALL"
   }
   ```

10 EnterpriseOne ADF Container

How EnterpriseOne Launches EnterpriseOne ADF Applications

Prior to Release 9.2.1, you must create an EnterpriseOne proxy application to associate with each EnterpriseOne ADF application that you deploy. An administrator uses the proxy application to set up processing options, versions, and security for the EnterpriseOne ADF application. You configure security on the proxy application through the Security Workbench.

Oracle's standard is to name both EnterpriseOne proxy and EnterpriseOne ADF applications beginning with the letter E, for example E01012.

To launch an EnterpriseOne ADF application from EnterpriseOne, a Solution Explorer task of type "ADF Application" must first exist that specifies both an EnterpriseOne proxy application name and an ADF task flow ID. Optionally, the proxy application version can also be specified on the task; otherwise, ZJDE0001 is used as the default version. When the Solution Explorer task is launched, the proxy application name, version, jargon code, and ADF task flow ID from the task are sent to the EnterpriseOne ADF Container (JDEADFContainer). If the ADF task flow is found, the ADF application launches inside an available region of the JDEADFContainer web application.

With Release 9.2.1, you can attach your EnterpriseOne ADF application to an external form in Form Design Aid. The external form provides users with a direct link to the EnterpriseOne ADF application and allows it to be displayed within an EnterpriseOne web client. This means it can be a task, within a Composed EnterpriseOne Page, on a menu, linked to from an EnterpriseOne page, or accessed through FastPath.

External forms are intended to replace the use of the proxy application. All functionality associated with the proxy application, including processing options, static translated text for ADF labels, and security, are also included with the application containing the external form.

For more information on creating external forms, see *"Understanding External Forms" in the JD Edwards EnterpriseOne Tools Form Design Aid Guide .*

Understanding Processing Options for ADF Applications

If an EnterpriseOne application called by an EnterpriseOne ADF application has versions associated with it, then the EnterpriseOne *proxy* application must have a processing option field that specifies the version of the EnterpriseOne application to be invoked by the EnterpriseOne ADF application. The versions specified in the processing options can then be used when the EnterpriseOne ADF application executes form service requests in the data control methods.

Retrieving Processing Options for ADF Applications

You use the AIS Server processing option service to retrieve processing options for EnterpriseOne proxy applications and applications called by the EnterpriseOne ADF application.

The EnterpriseOne ADF Container will pass the proxy application's name and version to the EnterpriseOne ADF application as input parameters, which are stored as Page Flow Scope variables. For details on how to use the

processing option service, see *"Processing Option Service" in the JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide* .

When an EnterpriseOne ADF application is launched from an external form, the name of the application containing the external form is passed to the ADF task flow. (Release 9.2.1)

Using the E1ADFUtils Helper Class to Access EnterpriseOne ADF Container Features

E1ADFUtils is a helper class included in the E1UserSession.jar that provides several static methods for accessing functionality available through the EnterpriseOne ADF Container, including launching an EnterpriseOne menu task, launching an EnterpriseOne application, and displaying error and warning messages using the EnterpriseOne ADF Container's message popup dialog.

> **Note:** These features are only available when the EnterpriseOne ADF application launches from an EnterpriseOne Solution Explorer task or external form (Release 9.2.1) and executes in the EnterpriseOne ADF Container. If the EnterpriseOne ADF application is run on JDeveloper's Integrated Oracle WebLogic Server, the application code to utilize these features will execute without error, but the EnterpriseOne ADF Container is not available to provide the desired functionality.

Return Type	Method	Description
void	launchE1Task(String taskId)	Launches the EnterpriseOne object associated with the specified task ID.
void	launchE1Application(E1FormInterconne formInterconnect)	Launches the specified EnterpriseOne application.
void	addErrorMessages(List<E1Message> messages)	Adds a list of E1Message objects to the JDEADFContainer's error message list.
void	addErrorMessage(String message)	Adds a single message to the JDEADFContainer's error message list.
void	addWarningMessages(List<E1Message messages)	Adds a list of E1Message objects to the JDEADFContainer's warning message list.
void	addWarningMessage(String message)	Adds a single message to the JDEADFContainer's warning message list.
void	addInformationMessages(List<E1Mess messages)	Adds a list of E1Message objects to the JDEADFContainer's information message list.

Return Type	Method	Description
void	addInformationMessage(String message)	Adds a single message to the JDEADFContainer's information message list.
void	launchMessagePopup()	Displays the JDEADFContainer's message popup dialog. Once the dialog is displayed, the message lists are cleared.
void	initializeAppInstance(String path)	Retrieves values from the About.properties file for the EnterpriseOne ADF application to display in the About dialog.

Launching EnterpriseOne Menu Task

To launch an EnterpriseOne menu task from your EnterpriseOne ADF application, you only need to specify the task ID when calling the launchE1Task() method.

Example: Launching EnterpriseOne Menu Task

```
// Launch Address Book menu task.E1AdfUtils.launchE1Task("11/G1341");
```

Launching an EnterpriseOne Application

To launch an EnterpriseOne application from your EnterpriseOne ADF application, you need to first create an E1FormInterconnect object to identify the application and form to run and the form interconnect values to pass to the EnterpriseOne application form. Then you call the launchE1Application() method, passing the E1FormInterconnect object as the sole parameter.

Example: Launching an EnterpriseOne Application

```
// Launch Address Book Work With Addresses form.
E1FormInterconnect form = new E1FormInterconnect();
form.setApplication("P01012");
form.setForm("W01012B");
form.setVersion("ZJDE0001");
form.addFormInterconnectValue("1", "7500");
form.addFormInterconnectValue("2", "E");
E1AdfUtils.launchE1Application(form);
```

Displaying Error and Warning Messages

When a form service request executes on the EnterpriseOne HTML Server, any errors and warnings displayed on the EnterpriseOne form are returned to the EnterpriseOne ADF application in the JSON response. You can retrieve these

messages from the response object and display them to the user using the EnterpriseOne ADF Container's message popup, which provides a modal dialog window for error, warning, and information messages. You will typically perform the following steps to display messages to the user:

1. To display a single message, call the E1AdfUtils method to add the individual message to one of the exception lists in the EnterpriseOne ADF Container (JDEADFContainer).
2. To display multiple messages, collect error, warning, and information messages in separate lists and call the E1AdfUtils methods to add these lists to the JDEADFContainer's exception lists.
3. Call the launchMessagePopup() method to display the JDEADFContainer's message popup.

Example: Display a Single Error Message

```
if (AISClientCapability.isCapabilityAvailable(loginEnv, AISClientCapability.GRID))
{
    // Code to perform Form Service Request and process response.
}
else
{
    E1AdfUtils.addErrorMessage("The grid capability is not supported on AIS Server.");
    E1AdfUtils.launchMessagePopup();
}
```

Example: Display Multiple Error Messages

```
FormRequest formRequest = new FormRequest(loginEnv);

// Populate form request.
...condensed...

// Execute form request to add new employee to Employee Master using P060116Q.
String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,

 JDERestServiceProvider.POST_METHOD,

 JDERestServiceProvider.FORM_SERVICE_URI);

P060116Q_W060116QA_FormParent p060116FormParent =
 loginEnv.getObjectMapper().readValue(response, P060116Q_W060116QA_FormParent.class);

if (p060116FormParent != null)
{
    // If insert was successful, clear input fields.  Otherwise, display errors.
    FormErrorWarning[] errors = p060116FormParent.getFs_P060116Q_W060116QA().getErrors();
    List<E1Message> errorList = new ArrayList<E1Message>();
    if (errors.length > 0)
    {
        for (int i = 0; i < errors.length; i++)
        {
            errorList.add(new E1Message(E1Message.Severity.ERROR, errors[i].getMOBILE()));
        }

        E1AdfUtils.addErrorMessages(errorList);
 E1AdfUtils.launchMessagePopup();
    }
    else
    {
        clearInputFields();
    }
}
```

As mentioned previously, the EnterpriseOne ADF Container's message popup will only display when the EnterpriseOne ADF application is running inside the EnterpriseOne ADF Container. In order to display messages when the application is running on JDeveloper's Integrated Oracle WebLogic Server, you can conditionalize your error reporting code to use the Java Server Faces messaging feature, as shown in the below example:

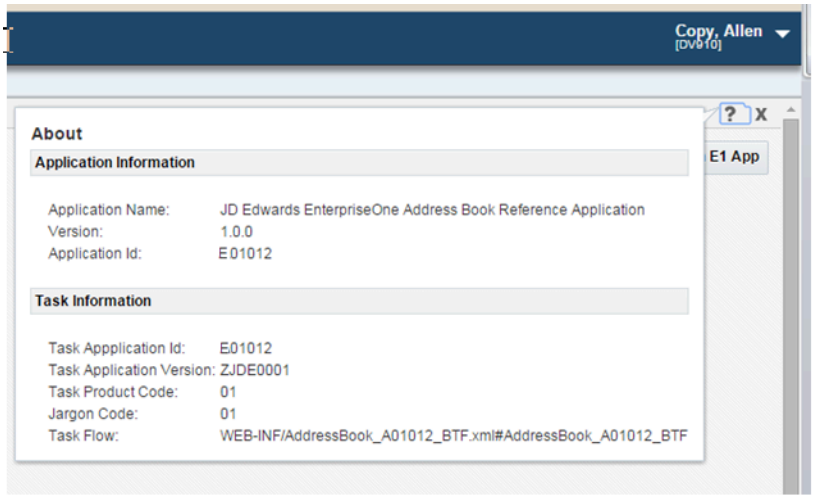
Example: Display Error Messages When Running Locally on JDeveloper's Integrated Oracle WebLogic Server

```
if (AISClientCapability.isCapabilityAvailable(loginEnv, AISClientCapability.GRID))
{
    // Code to perform Form Service Request and process response.
}
else
{
    String msg = "The grid capability is not supported on AIS Server";
    if (runningInJDEADFContainer)
    {
        E1AdfUtils.addErrorMessage(msg);
        E1AdfUtils.launchMessagePopup();
    }
    else
    {
        FacesContext.getCurrentInstance().addMessage("E1ADFApp", new
 FacesMessage(FacesMessage.SEVERITY_ERROR, "AIS Capability Not Supported ", msg));
    }
}
```

See *Accessing EnterpriseOne Data from ADF Applications* to review the code that first determines whether the EnterpriseOne ADF application is running locally or in the Enterpriseone ADF Container and then sets the runningInJDEADFContainer boolean accordingly.

Supporting About Information

The EnterpriseOne ADF Container provides an About button that enables you to display information about the EnterpriseOne ADF application launched within the container. When you click the About button, a dialog box, like the example below, appears and displays similar information:



The dialog box has two sets of information: Application Information and Task Information. The Application Information is provided by the individual ADF Bounded Task flow.

To set up the About information:

1. Create a new file in your project named about.properties and include the following content:

   [About]

   applicationName=JD Edwards EnterpriseOne Address Book Reference Application

   version=1.0.0

   appID=E01012

   Make sure to place the file in a package that is named uniquely. The standard location would be with the application named package. In this example it is in the com.oracle.e1.E01012 package.
2. During the initialization of your application you must communicate to the container where the about.properties for your application is stored. Make sure you have a handshakeId before calling the E1AdfUtils.initializeAppInstance() method. Having a handshakeId is an indication that your application is running within the container.
3. Pass the package path to your about.properties file as in this example:

   E1AdfUtils.intializeAppInstance("/com/oracle/e1/E01012/");

   > **Note:** The example application shown above is named E01012. "E" is the naming convention Oracle recommends you use for your EnterpriseOne ADF applications. With Release 9.2.1, applications using external forms for EnterpriseOne ADF applications should be named starting with "P."

About Running EnterpriseOne ADF Applications in Composed EnterpriseOne Pages (Release 9.2.0.2)

In addition to running EnterpriseOne ADF applications from a menu, you can also launch them from a Composed EnterpriseOne Page or run them as embedded content within a Composed EnterpriseOne Page. Furthermore, in Release 9.2.1, you can create an external form to link to your EnterpriseOne ADF application and use the EnterpriseOne Application content type to add it to a Composed EnterpriseOne Page. Otherwise, an EnterpriseOne menu task for the application must exist and be placed on a Task View before EnterpriseOne can run the EnterpriseOne ADF application as an ADF Application content type.

When EnterpriseOne ADF applications are launched from either a menu or Composed EnterpriseOne Page, the EnterpriseOne ADF Container displays About and Close icons in the menu bar. When EnterpriseOne ADF applications run as Composed EnterpriseOne Page *content*, the EnterpriseOne ADF Container is embedded within the Composed EnterpriseOne Page and replaces the Close icon with a Refresh icon so the About and Refresh icons are shown. During a refresh, the EnterpriseOne ADF application is restored to its original state by restarting the application's bounded task flow. Any unsaved data or changes to the existing view are lost. Furthermore, the EnterpriseOne ADF Container notifies the EnterpriseOne ADF application when it's running as embedded content by passing a parameter called "pageEmbedded" to the EnterpriseOne ADF application's bounded task flow. When this parameter's value is set to true, the EnterpriseOne ADF application is running inside a Composed EnterpriseOne Page and should use this parameter to modify its appearance or behavior based on where it's running. For example, when launched from a menu, the EnterpriseOne ADF application may contain a Close button on the page. However, when running as embedded Composed EnterpriseOne Page content, the same EnterpriseOne ADF application would hide the Close button, so the application is always displayed in the Composed EnterpriseOne Page and cannot be dismissed by the user.

11 Executing the Build Script from JDeveloper

Executing the Build Script from JDeveloper

In JDeveloper, developers can build an EnterpriseOne ADF application to test the application before actually building and deploying it in a production environment as described in *EnterpriseOne ADF Application Deployment* of this guide.

The JDE Ant build files can be executed from within JDeveloper by adding the Ant targets to the Tools menu as External Tool executables.

To execute Ant Targets from JDeveloper menu:

1. In JDeveloper, click **Tools**, then **External Tools**, and then select **New** from the External Tools dialog box.
2. Click **Next**.
3. Browse to the Ant build file you want to run, and click **Next**.
4. Move the build-and-deploy target to the Selected Targets field, and then click **Next**.
5. Set the Oracle WebLogic Server username and password properties needed for the build-and-deploy, deploy, and undeploy targets only. For any other Ant target in the script, these properties are not used.

   The administrative username - weblogic.username

   The administrative password - weblogic.password
6. Select the **Use Default Ant Version** option, and then click **Next**.
7. Specify the options for the Ant process, and then click **Next**.
8. The Weblogic.jar includes the class definition for the wldeploy Ant task used to deploy/undeploy files to/from an Oracle WebLogic Server. The Weblogic.jar file can be found in the <Weblogic_HOME>/server/lib directory, where Weblogic_HOME is the top-level directory of your Oracle WebLogic Server installation. This step is not necessary for any other Ant target in the build scripts.
9. Enter a caption for the menu item, and then click **Next**.
10. Select the **Tools Menu** option for the Add Items to Menus list, and then click **Next**.
11. Select **Always**, and then click **Finish**.

    Once the Ant target is added, it can be executed from the Tools menu. During execution, the Apache Ant log will display the Ant target's status and display messages if any exceptions are thrown.

12 Appendix A - Creating a Sample EnterpriseOne ADF Application

Creating a Sample EnterpriseOne ADF Application

Before You Begin

To create the sample application discussed in this chapter, you must download the EnterpriseOne ADF Foundation and AIS Client Java API from the JD Edwards Update Center. These downloads contain five jar files required for creating EnterpriseOne ADF applications.

Also, you need to download and install the AIS Client Class Generator.

See *Prerequisites for Developing EnterpriseOne ADF Applications* for information on how to download these components.

You must also have a basic understanding of how to use JDeveloper.

AIS Class Generator

The EnterpriseOne AIS Client Class Generator is a tool you use to create Model foundational classes. These classes are specific to the EnterpriseOne application which will be accessed by the EnterpriseOne ADF application. This tool will be referred to in sections to follow which run you through the steps of creating an EnterpriseOne ADF application, so make sure that you know how to use the AIS Client Class Generator and how to create the JSON file with REST testing tool.

Creating the Sample EnterpriseOne Address Book ADF Application

This section explains how to build the ADFAddressBook application. You will add the functionality to search the EnterpriseOne Address Book and display the results in a list. The purpose of this section is to demonstrate how to make use of the form service request, and how to leverage current EnterpriseOne functionality. This section does not explain how to develop a fully functional user interface Address Book application.

Getting Started

The steps performed for this section are required for all EnterpriseOne ADF applications. In the following sections, you will continue to build out the EnterpriseOne ADF application started here and add functionality to search and review address book data from the EnterpriseOne Address Book (P01012) application.

1. To begin, save the following jar files to the C:\JDeveloper\mywork directory:
    - AIS_Client.jar
    - jackson-annotations-2.x.x.jar
    - jackson-core-2.x.x.jar
    - jackson-databind-2.x.x.jar
    - E1UserSession.jar

Creating a New EnterpriseOne ADF Application

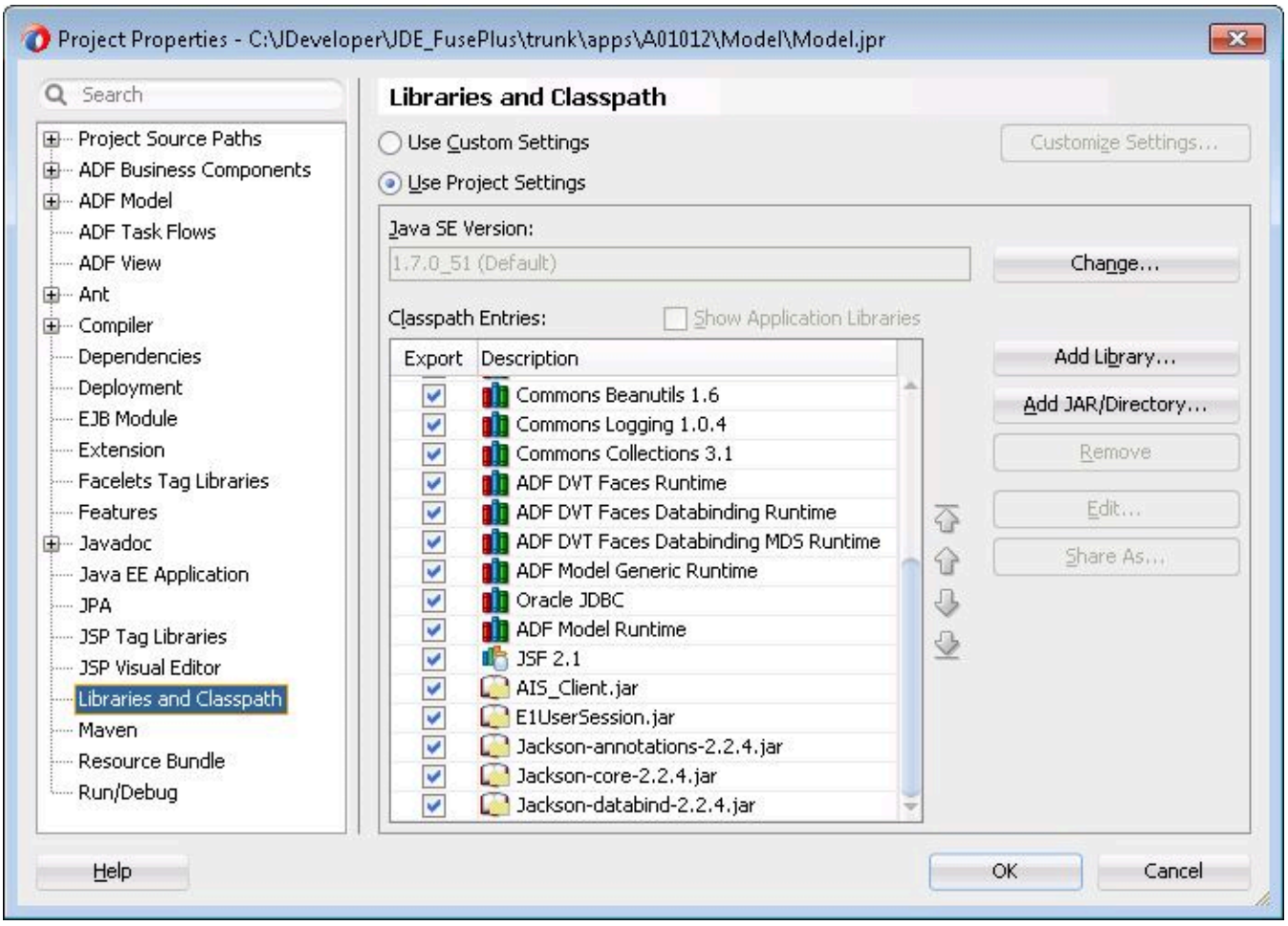
This section describes how to create a new EnterpriseOne ADF application.

1. In JDeveloper, create a new ADF Application.
2. On the New Gallery form, select Applications from the **Categories** section, and then click **ADF Fusion Web Application** from the Items: list.
3. Click **OK**.
4. On the Name Application form, enter the following information:
    - **Application Name**: E01012
    - **Directory**: C:\jdeveloper\mywork\E01012
    - **Application Package Prefix**: com.oracle.e1.E01012

      Ensure that the Application Package Prefix is populated correctly with the following sample name: com.oracle.e1.E01012

      > **Note:** Oracle's standards are to name EnterpriseOne ADF applications beginning with the letter E, for example, E01012.
5. Accept the default settings and click **Finish**.
6. Add the jar files to your Model project by completing the following steps:
    - Right-click **Model** project
    - Select **Project Properties**
    - Select **Libraries and Classpath** (towards the bottom of the left pane)
    - Click the **Add Jar/Directory...** button located on the right side of the screen and add the following jars:
        - AIS_Client.jar
        - jackson-annotations-2.x.x.jar
        - jackson-core-2.x.x.jar
        - jackson-databind-2.x.x.jar
        - E1UserSession.jar

- The following graphic displays the results you should now see:



7. Click the **Add Library** button located on the right side of the screen.
8. Click the **ADF Model Generic Runtime Classpath** entry, and then click OK.
9. Click **OK** to save the changes.
10. In the Projects panel on the left, delete the **WEB-INF folder** that has been added to the Model project.
11. **Save** changes made to the Model project.

Building an EnterpriseOne ADF Application in JDeveloper

This section describes how to use the form service request and leverage existing EnterpriseOne functionality. It describes how to build the E01012 - AddressBook application and describes how to create the functionality used to search for EnterpriseOne Address Book data and display the results in a list. It does not describe how to develop a fully functional Address Book application.

Before continuing through this section, you should have completed the steps in the previous sections. Also, you need to set up the AIS Client Class Generator preferences before you continue. See *"Configuring the AIS Client Class Generator" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for details.

Generating Form Service Foundation Classes for the Data Model

You use the AIS Client Class Generator in JDeveloper to generate the form service foundation classes that are needed to process responses to AIS form service requests.

To run the AIS Client Class Generator

1. Highlight the **Model** project.
2. Run the AIS Client Class Generator by clicking the **Tools** menu, and then clicking **AIS Client Class Generator.**
3. Enter the following inputs to generate classes for W01012B:
    - Application Name: P01012
    - Form Name: W01012B
    - Version: ZJDE0001
    - ReturnControlIDs: 1[19,20,48,49,50,51],54,58,63,62
    - FormInputs: (C, 54)
    - FormServiceAction: R
    - FindOnEntry: <checked>
    - DemoMode: <checked>
    - Generate for Mobile: <unchecked>
    - Preview JSON: <checked>
    - Keep JSON: <checked>
4. Click **Generate** and then **Continue**.
5. Enter the following inputs to generate classes for W01012A:
    - Application Name: P01012
    - Form Name: W01012A
    - Version: ZJDE0001
    - FormServiceAction: R
    - FindOnEntry: <checked>
    - DemoMode: <checked>
    - Preview JSON: <checked>
    - Keep JSON: <checked>
    - Return Control IDs: [blank]
6. Click **Generate** and then **Continue**.
7. Refresh the Model project to see the newly generated classes available for consumption

Creating the Data Control Class

This section describes how to create the AddressBookDC.java class.

1. In the Projects panel located on the left of the screen, right-click the **Model** project, select **New**, and then select **From Gallery**.
2. On the New Gallery form, select **Java** from the Categories panel, and then select **Class** in the Items panel.
3. On the Create Java Class form, enter the following information:
   - **Name**: AddressBookRecord
   - **Package**: com.oracle.e1.E01012.formservicetypes
4. Click the **OK** button to create the Java class.
5. Add the following fields to AddressBookRecord.java:

```
private String addressNumber;
private String alphaName;
private String mailingName;
private String address;
private String city;
private String state;
private String zipCode;
private String taxId;
private String businessUnit;
private String searchType;
```

6. Add the following method to AddressBookRecord.java:

```
public void clear()
{
    addressNumber = "";
    alphaName = "";
    mailingName = "";
    address = "";
    city = "";
    state = "";
    zipCode = "";
    taxId = "";
    businessUnit = "";
    searchType = "";
}
```

7. Right-click within the cass and select **Generate Accessors...** from context menu.
8. Select all methods and click **OK** button.
9. **Save** changes to AddressBookRecord.java.
10. In the **Projects** panel located on the left of the screen, right-click the **Model** project, select **New**, and then select Java Class.
11. On the Create Java Class form, enter the following information:

    Name: AddressBookDC

    Package: com.oracle.e1.E01012.model
12. Click the **OK** button to create the Java class.
13. Add the following fields to AddressBookDC.java:

```
    private P01012_W01012B_FormParent addressBook = new
P01012_W01012B_FormParent();
    private P01012_W01012B_GridRow selectedAddressBook = new
P01012_W01012B_GridRow();
    private AddressBookRecord addressBookDetail = new AddressBookRecord();
    private String selectedAddressNumber = "";

    // Processing Options
    private String p01012Version = "";
    private String hideTax = "";
    private String displaySupplierMaster = "";
    private String displayCustomerMaster = "";
    private String searchTypeDefault = "";
    private String displayCSMS = "";
    private String p0100041Version = "";
    private String selfServiceMode = "";
    private String typeCode = "";
    private String postalCodeValidate = "";
    private String postalAddressRetrieval = "";
    private String p03013Version = "";
    private String p04012Version = "";
```

14. Right-click within the class and select **Generate Accessors...** from context menu.
15. Select all methods and click **OK** button
16. Save changes to AddressBookDC.java.

Populating Address Book List and Creating the Data Control

In this section you will populate the address book list and create the data control.

1. Add the following connection fields and constants to AddressBookDC.java

```
    // Session
    private E1UserSessionBean userBean;
    private LoginEnvironment  loginEnv;
    private boolean runningInJDEADFContainer = true;

    // Hard coded connection values.
    private static final String AIS_SERVER = "http://host:port";
    private static final String USER_NAME = "username";
    private static final String PASSWORD = "password";
    private static final String DEVICE = "E1ADFApps";
    private static final String ROLE = "role";
    private static final String ENVIRONMENT = "environment";
```

2. Add the code shown below to AddressBookDC.java:

```
public AddressBookDC()
{
     ADFContext.getCurrent().getPageFlowScope().put("showDetail", "false");
     String handshakeId = (String) ADFContext.getCurrent().getPageFlowScope().get("handshakeId");
     if (handshakeId == null || (handshakeId != null && handshakeId.length() == 0))
     {
          runningInJDEADFContainer = false;
          userBean = new E1UserSessionBean(AIS_SERVER, USER_NAME, PASSWORD, ENVIRONMENT, ROLE, DEVICE);
     }
     else
     {
          // Initialize application for about properties, only do this when running in the container
          E1AdfUtils.intializeAppInstance("/com/oracle/e1/E01012/");
     }
```

```
        loginEnv = E1AdfUtils.getLoginEnvironment(userBean);

        if (loginEnv != null)
        {
             List<String> reqCapabilities = loginEnv.getRequiredCapabilities();
   reqCapabilities.add("processingOption");

             retrievePOValues();
        }
        else
        {
             processConnectionException();
        }
}

private void retrievePOValues()
{
        try
        {
             // Retrieve E01012 proxy application's processing options
             String appName = (String) ADFContext.getCurrent().getPageFlowScope().get("appName");
             String appVersion = (String) ADFContext.getCurrent().getPageFlowScope().get("appVersion");

             ProcessingOptionRequest e01012PORequest = new ProcessingOptionRequest(loginEnv);
             if ((appName == null) || (appName.length() == 0))
             {
                  appName = "E01012";
             }
             e01012PORequest.setApplicationName(appName);

             if ((appVersion == null) || (appVersion.length() == 0))
             {

                  appVersion = "ZJDE0001";
             }
             e01012PORequest.setVersion(appVersion);

             String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, e01012PORequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.PO_SERVICE);
             ProcessingOptionsSet e01012POSet = loginEnv.getObjectMapper().readValue(response,
 ProcessingOptionsSet.class);

             if (e01012POSet != null)
             {
                  p01012Version = (String) e01012POSet.getOptionValue("szP01012Version_1");
                  if (p01012Version == null)
                  {
                       p01012Version = "ZJDE0001";
                  }
             }
             else
             {
                  p01012Version = "ZJDE0001";
             }

             // Retrieve Address Book (P01012) processing options.
    ProcessingOptionRequest p01012PORequest = new ProcessingOptionRequest(loginEnv);
             p01012PORequest.setApplicationName("P01012");

             p01012PORequest.setVersion(p01012Version);
           String response1 = JDERestServiceProvider.jdeRestServiceCall(loginEnv, p01012PORequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.PO_SERVICE);
             ProcessingOptionsSet p01012POSet = loginEnv.getObjectMapper().readValue(response1,
 ProcessingOptionsSet.class);

             if (p01012POSet != null)
             {
                  hideTax = (String) p01012POSet.getOptionValue("HideTax_4");
                displaySupplierMaster = (String) p01012POSet.getOptionValue("GoToSupplierMaster_5");
                  displayCustomerMaster = (String) p01012POSet.getOptionValue("GoToCustomerMaster_6");
                  searchTypeDefault = (String) p01012POSet.getOptionValue("SearchTypeDefault_7");
                  displayCSMS = (String) p01012POSet.getOptionValue("GoToCSMS_8");
                  p0100041Version = (String) p01012POSet.getOptionValue("Version_9");
                  selfServiceMode = (String) p01012POSet.getOptionValue("cSelfServiceMode_10");
```

```
                typeCode = (String) p01012POSet.getOptionValue("cTypeCode_11");
                postalCodeValidate = (String) p01012POSet.getOptionValue("cPostalCodeValidate_12");
                postalAddressRetrieval = (String)
 p01012POSet.getOptionValue("cPostalAddressRetrieval_13");
                p03013Version = (String) p01012POSet.getOptionValue("szCustMasterVersion_VER_14");
                p04012Version = (String) p01012POSet.getOptionValue("szSuppMasterVersion_VER_15");
           }
      }
      catch (CapabilityException e)
      {
           // Processing Option cability is not supported on AIS.  Use default search type.
           searchTypeDefault = "E";
      }
      catch (JDERestServiceException e)
      {
           processErrorException(JDERestServiceProvider.handleServiceException(e));
      }
      catch (Exception e)
      {
           processErrorException(e);
      }
}

public void retrieveAddressBookList()
{
     if (loginEnv != null)
     {
          FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P01012_W01012B");
          formRequest.setVersion(p01012Version);
          formRequest.setMaxPageSize("600");
          formRequest.setFormServiceAction("R");

          FSREvent w01012BFSREvent = new FSREvent();

          // Set search type based on the processing option of P01012.
  w01012BFSREvent.setFieldValue("54", searchTypeDefault);

          String alphaNameFilter = (String)
 ADFContext.getCurrent().getPageFlowScope().get("alphaNameFilter");
          if (alphaNameFilter != null && alphaNameFilter.trim().length() > 0)
          {
               w01012BFSREvent.setFieldValue("58", "*" + alphaNameFilter + "*");
          }
          else
          {
               w01012BFSREvent.setFieldValue("58", "**");
          }

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

          try
          {
               String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);

               P01012_W01012B_FormParent temp_p01012_W01012B_FormParent =
 loginEnv.getObjectMapper().readValue(response, P01012_W01012B_FormParent.class);
   addressBook.getFs_P01012_W01012B().getData().getGridData().setRowset
   (temp_p01012_W01012B_FormParent.getFs_P01012_W01012B().getData().getGridData().getRowset());

               if (addressBook.getFs_P01012_W01012B().getData().getGridData().getRowset().size() > 0)
               {
                    ADFContext.getCurrent().getPageFlowScope().put("selectedRowKey", "0");

                    // Display Address Book detail for first record in list.
    processSelectedAddress(0);
               }
               else
               {
                    // Clear detail fields.
                    addressBookDetail.clear();
                    ADFContext.getCurrent().getPageFlowScope().put("showDetail", "false");
               }
```

```
            }
            catch (JDERestServiceException e)
            {
              processErrorException(JDERestServiceProvider.handleServiceException(e));
            }
            catch (IOException e)
            {
                 processErrorException(e);
            }
       }
       else
       {
            processConnectionException();
       }
  }

  public void processSelectedAddress(int rowIndex)
  {
       if (rowIndex >= 0)
       {
            selectedAddressBook =
   addressBook.getFs_P01012_W01012B().getData().getGridData().getRowset().get(rowIndex);
            setSelectedAddressNumber(selectedAddressBook.getMnAddressNumber_19().getValue());

            ADFContext.getCurrent().getPageFlowScope().put("showDetail", "true");
       }
       else
       {
            ADFContext.getCurrent().getPageFlowScope().put("showDetail", "false");
       }

       retrieveAddressBookDetail(selectedAddressNumber);
  }

  private void retrieveAddressBookDetail(String addressNumber)
  {
       if (loginEnv != null)
       {
            FormRequest formRequest = new FormRequest(loginEnv);
            formRequest.setFormName("P01012_W01012A");
            formRequest.setVersion("ZJDE0001");
            formRequest.setFormServiceAction("R");
            formRequest.addToFISet("12", addressNumber);

            try
            {
                 String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
   JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);

                 P01012_W01012A_FormParent tempDetail = loginEnv.getObjectMapper().readValue(response,
   P01012_W01012A_FormParent.class);
                 if (tempDetail != null)
                 {
                      P01012_W01012A_FormData form = tempDetail.getFs_P01012_W01012A().getData();
                      setSelectedAddressNumber(form.getTxtAddressNumber_21().getValue());
                      addressBookDetail.setAddressNumber(form.getTxtAddressNumber_21().getValue());
                      addressBookDetail.setAlphaName(form.getTxtAlphaName_28().getValue());
                      addressBookDetail.setMailingName(form.getTxtMailingName_38().getValue());
                      addressBookDetail.setAddress(form.getTxtAddressLine1_40().getValue());
                      addressBookDetail.setCity(form.getTxtCity_52().getValue());
                      addressBookDetail.setState(form.getTxtState_54().getValue());
                      addressBookDetail.setZipCode(form.getTxtPostalCode_50().getValue());
                      addressBookDetail.setTaxId(form.getTxtTaxID_34().getValue());
                      addressBookDetail.setBusinessUnit(form.getTxtBusinessUnit_62().getValue());
                      addressBookDetail.setSearchType(form.getTxtSearchType_36().getValue());
                 }
                 else
                 {
                      addressBookDetail.clear();
                 }
            }
            catch (JDERestServiceException e)
           {
                 processErrorException(JDERestServiceProvider.handleServiceException(e));
            }
```

```
            catch (IOException e)
            {
                processErrorException(e);
            }
        }
        else
        {
            processConnectionException();
        }
    }

    public void clearFields()
    {
        addressBookDetail.clear();
    }

    public void launchE1App()
    {
        E1FormInterconnect form = new E1FormInterconnect();
        form.setApplication("P01012");
        form.setForm("W01012B");
        form.setVersion("ZJDE0001");
        form.addFormInterconnectValue("2", "C");
        E1AdfUtils.launchE1Application(form);
    }

    private void processErrorException(String msg)
    {
        displayMessage("Error", msg);
    }

    private void processErrorException(Exception e)
    {
        displayMessage("Error", e.getMessage());
    }

    private void processConnectionException()
    {
        displayMessage("Connection Error", "Connection failed. Please contact your system administrator");
    }

    private void displayMessage(String title, String msg)
    {
        if (runningInJDEADFContainer)
        {
            E1AdfUtils.addErrorMessage(msg);
            E1AdfUtils.launchMessagePopup();
        }
        else
        {
           ELContext elCtxt = ADFContext.getCurrent().getELContext();
           ExpressionFactory expFactory = ADFContext.getCurrent().getExpressionFactory();
           ValueExpression ve = expFactory.createValueExpression(elCtxt, "#{addressBookListenerBean}",
 Object.class);
           if (ve.getValue(elCtxt) != null)
           {
               MethodExpression me = expFactory.createMethodExpression(elCtxt,
 "#{addressBookListenerBean.displayLocalMessage}", Void.class, new Class[] {String.class,
 String.class});
                    me.invoke(elCtxt, new Object[] {title, msg});
           }
        }
    }
```

3. Right-click the AddressBookDC in the Projects Panel and select **Create Data Control**.
4. Accept the defaults and click **Next**, **Next**, and **Finish**.

Creating a Task Flow

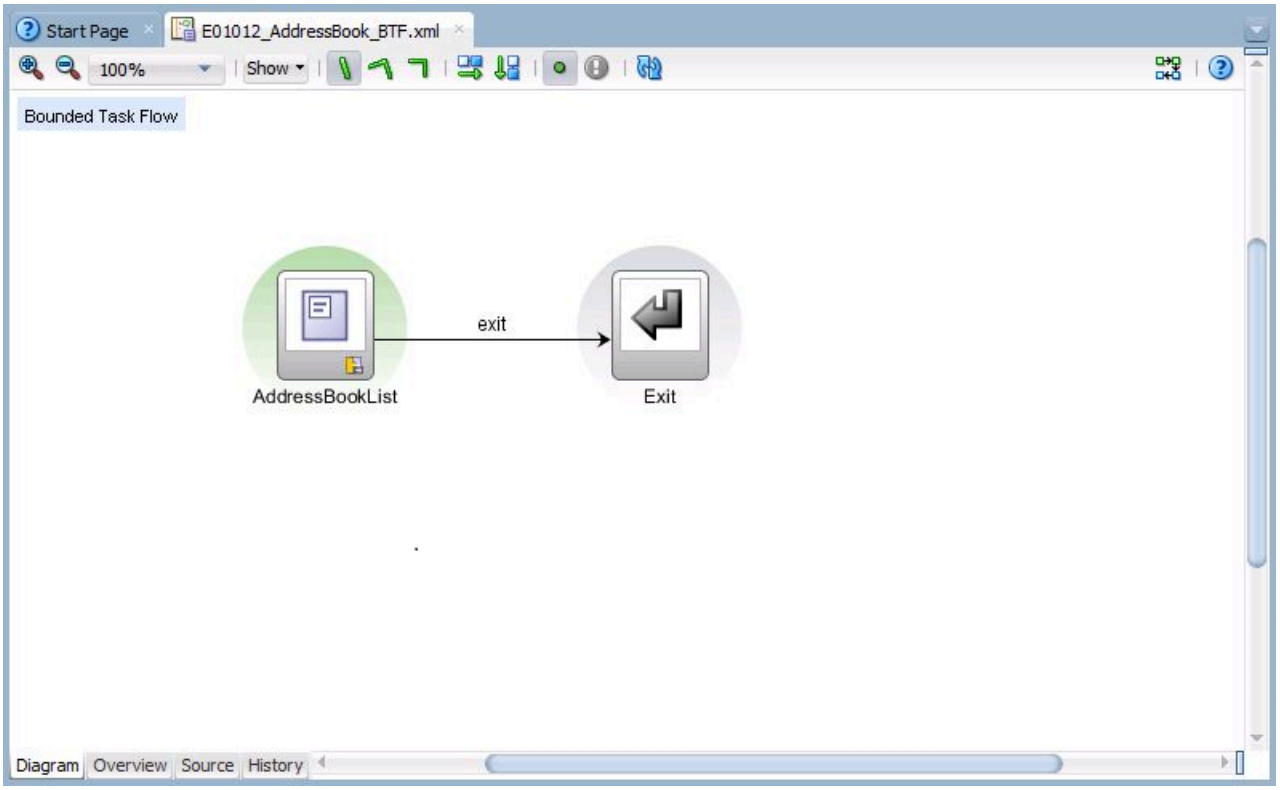
This section describes how to create a task flow.

1. In the Projects panel on the left, expand both the **ViewController** project and **Web Content** folder, and right-click the **Page Flows** folder.
2. Select **New** and then choose **ADF Task Flow**.
3. On the Create Task Flow form, enter `E01012_AddressBook_BTF` in the Name field.
4. Verify that Create as Bounded Task Flow and Create with Page Fragments options are both selected.

> **Note:**
>
> As previously mentioned in Chapter 7, an EnterpriseOne Solution Explorer task must be created for an EnterpriseOne ADF application before it can be launched. On this Solution Explorer task, you will need to specify the task flow ID of the EnterpriseOne ADF application (see *Finding the ADF Bounded Task Flow ID*), which includes the task flow name you specify in this step. For the sample EnterpriseOne ADF application, the task flow ID will be:
>
> /WEB-INF/E01012_AddressBook_BTF.xml#E01012_AddressBook_BTF
>
> Due to size restrictions of the URL field on the Solution Explorer task, a task flow ID cannot exceed 80 characters. This includes the starting "/WEB-INF/" directory name and any subfolders you include in the task flow path.
>
> With Release 9.2.1, the task flow ID is specified in the External Application form property for the external form type.

5. From the **Components** palette, drag a **View** component onto the E01012_AddressBook_BTF.xml diagram.
6. Name the view `AddressBookList`.
7. Drag a **Task Flow Return** component onto the E01012_AddressBook_BTF.xml diagram.
8. From the Components palette, select the **Control Flow Case** component, click on the **AddressBookList** view, and then click on the return activity. This creates a flow arrow from the view to the activity.
9. Set the action string for the flow arrow to exit.

10. Use the following graphic as a guide for setting up the bounded task flow diagram:



11. Double-click the **AddressBookList** view to create a facelet page fragment called AddressBookList.jsff.

> **Note:**
>
> Make sure you save the jsff file to the following location:
>
> C:\jdeveloper\mywork\E01012\ViewController\public_html\fragments

12. Select the **Overview** tab at the bottom of the E01012_AddressBook_BTF.xml diagram, select **Behavior** tab from the available list on the left, and change the Share data controls with calling task flow option to Isolated.
13. Select the **Parameters** tab and add the following Input Parameter Definitions:

Name	Class	Required
appName	java.lang.String	false
appVersion	java.lang.String	false
handshakeId	java.lang.String	false
jargonCode	java.lang.String	false
pageEmbedded	java.lang.String	false

Name	Class	Required
		See *EnterpriseOne ADF Container*for additional details on how this parameter relates to Composed EnterpriseOne Pages.
formDataStructure	com.oracle.e1.E1FormDataStructure	false

Note: In addition to the above list of input parameters, bounded tasks flows for EnterpriseOne ADF applications must satisfy the following requirements:

- The bounded task flow must contain a Task Flow Return activity with an "exit" action string for the navigation rule.
- The bounded task flow must have a wild card navigation rule to transition to the exit activity using the "exit" action string if you are using multiple views in your bounded task flow.

Creating Address Book Search Panel

1. In the **Projects** panel located on the left, expand the **ViewController**, **Web Content**, and **fragments** nodes and double-click the **AddressBookList.jsff** page fragment.
   This will display the page fragment in the design panel in the middle.
2. Select the **source** tab at the bottom of the panel. Insert a **Panel Stretch Layou**t between the ui:composition tags from the Components palette and set the following attributes:
   - startWidth: 350px
   - inlineStyle: margin:10px;
   - dimensionsFrom: parent
3. Insert a second **Panel Stretch Layout** component in the start facet of the first Panel Stretch layout and set the following attribute:
   topHeight: 40px
4. Insert a **Panel Group Layout component** inside the top facet of the inner Panel Stretch Layout. Set the layout attribute to horizontal.
5. Insert an **Input Text** Component inside the Panel Group Layout and set the following attributes:
   - label: Name
   - value: #{pageFlowScope.alphaNameFilter}
6. Insert a **Spacer** component after the Input Text component and set its width attribute to 12px.
7. Insert a Button component after the Spacer component and change the text attribute to Search.
8. Set focus on the Button component and then select the **Edit** option next to the ActionListener property in the Properties panel.
9. On the Edit Property: ActionListener form, click the **New** button next to the Managed Bean drop-down list, and set the following fields:
   - Bean Name: addressBookListenerBean

- Class Name: AddressBookListenerBean
- Scope: request (Use default)

10. Click **OK** to create the class and the managed bean declaration in the E01012_AddressBook_BTF.xml.
11. Select the **New** button next to the Method drop-down list and set the following field:

    Method Name: searchAddressBook
12. Click **OK** to create the method and click **OK** again to close the Edit Property form.
13. Insert a **Spacer** component after the **ADF Button** and set its width attribute to 8px.
14. Insert a **Button** component after the **Spacer**, and set the following attributes:
    - action: exit
    - text: Close
    - rendered: #{!pageFlowScope.pageEmbedded}
15. From the **Data Controls** panel, drag the **rowset** member in addressBook > fs_P01012_W01012B > data > gridData into the center facet of the inner Panel Stretch Layout.

    You will be prompted to specify the type of component to add.
16. Select **Table/List View > ADF List View** and then click **Next and Finish** to add the component.
17. Delete the **separator** facet inside of the Panel Group Layout.
18. Select the **Bindings** tab at the bottom of the panel.
19. In the **Bindings** list on the left, highlight the **rowset** binding and click the pencil icon to edit the binding.
20. On the **Edit Tree Binding** form, click the green **'+'** icon and select **mnAddressNumber_19**.
21. Select the parent folder, click the **green icon** again, and then select **SAlphaName_20**. You can ignore the "Rule Already Exists for the selected Accessor" error.
22. Click the **OK** button to save the binding changes.
23. In the **Bindings** list, click the green '**+**' icon to add a new binding.
24. On the Insert Item form, select **action** from the list and click the **OK** button.
25. On the **Create Action Binding** form, expand the **AddressBookDC** node and select the **Execute** operation under addressBookDetail.
26. Click the **OK** button to add the binding.
27. Click the green '**+**' icon again to add a second binding.
28. On the Insert Item form, select **methodAction** from the list and click the **OK** button.
29. On the Create Action Binding form, expand the **AddressBookDC** node and select the **retrieveAddressBookList()** method. Click the **OK** button to add the binding.
30. Click the green '**+**' icon again to add a third binding.
31. On the Insert Item form, select **methodAction** from the list and click **OK**.
32. On the Create Action Binding form, expand the **AddressBookDC** node and select the **processSelectedAddress(int)** method. Click **OK** to add the binding.
33. Click **Save All** from the toolbar.
34. Select the **Source** tab at the bottom of the panel.
35. Insert an **Output Text Component** inside the Panel Group Layout of the List View and set the value attribute to #{item.mnAddressNumber_19.bindings.value.inputValue} - #{item.SAlphaName_20.bindings.value.inputValue}.
36. Set focus on the **List View** component and then select the **Edit** option next to the SelectionListener property in the Properties panel.
37. On the Edit Property: SelectionListener form, select **addressBookListenerBean** from Managed Bean drop-down list.
38. Select the **New** button next to the Method drop-down list and set the following field:

    Method Name: addressRecordSelected

39. Click the **OK** button to create the method and click **OK** again to close the Edit Property form.
40. Set the **List View binding** attribute to #{addressBookListenerBean.listView} and the selection attribute to single.
41. Set focus on the **List View component** and then select the **Edit** option next to the PartialTriggers property in the Properties panel.
42. On the Edit Property: PartialTriggers form, expand the top facet and panel group layout components and double-click button-b1 from the Available panel to move it to the Selected panel.
43. Click the **OK** button to set the attribute value.
44. In the Projects panel, expand the **ViewController and Application Sources** nodes and open the com.oracle.e1.E01012.view.AddressBookListenerBean.java class.
45. Add the code shown below to AddressBookListenerBean.java.

```
private ComponentReference listView;
private ComponentReference form;

public AddressBookListenerBean()
{
}

public void searchAddressBook(ActionEvent actionEvent)
{
     // Reset index of list view iterator.
     BindingContainer bc = BindingContext.getCurrent().getCurrentBindingsEntry();
     DCIteratorBinding rowsetItr = ((DCBindingContainer)
 bc).findIteratorBinding("rowsetIterator");
     if (rowsetItr != null)
     {
          rowsetItr.clearForRecreate();

          rowsetItr.setCurrentRowIndexInRange(0);
     }

     // Reset list view's selected row keys so no list item is initially
 highlighted.
     resetListViewSelectedItem();

     // Execute data conrol method to retrieve list of Adress Book records.
     OperationBinding methodBinding =
 bc.getOperationBinding("retrieveAddressBookList");
     if (methodBinding != null)
     {
          methodBinding.execute();
          refreshAddressBookDetailIterator(bc);
     }
}

public void addressRecordSelected(SelectionEvent event)
{
     RichListView listView = getListView();

     // Retrieve index of selected list item row.
     RowKeySetTreeImpl selectedRowKeys = (RowKeySetTreeImpl)
 listView.getSelectedRowKeys();
     List rowKeyList = (List) selectedRowKeys.iterator().next();
     Key rowKey = (oracle.jbo.Key) rowKeyList.get(0);
     int rowIndex = ((Integer) rowKey.getKeyValues()[0]).intValue();

     AdfFacesContext.getCurrentInstance().addPartialTarget(listView);
```

```
        // Launch data control method to retrieve Address Book detail based on
 selected address number.
        BindingContainer bc = BindingContext.getCurrent().getCurrentBindingsEntry();
        OperationBinding methodBinding =
 bc.getOperationBinding("processSelectedAddress");
        if (methodBinding != null)
        {
             Map args = methodBinding.getParamsMap();
             args.put("rowIndex", rowIndex);
             methodBinding.execute();

             // Refresh detail panel.

 AdfFacesContext.getCurrentInstance().addPartialTarget(form.getComponent());
             refreshAddressBookDetailIterator(bc);
        }
   }

   private void refreshAddressBookDetailIterator(BindingContainer bc)
   {
        // Execute iterator binding to refresh Addrss Book detail panel.
        OperationBinding abDetailIterator = bc.getOperationBinding("Execute");
        if (abDetailIterator != null)
        {
             abDetailIterator.execute();
        }
   }

   public void resetListViewSelectedItem()
   {
        getListView().setSelectedRowKeys(new RowKeySetTreeImpl());
   }

   public void setListView(RichListView listView)
   {
        this.listView = ComponentReference.newUIComponentReference(listView);
   }

   public RichListView getListView()
   {
        if (listView != null)
        {
             return (RichListView) listView.getComponent();
        }
        return null;
   }

   public void setForm(RichPanelFormLayout form)
   {
        this.form = ComponentReference.newUIComponentReference(form);
   }

   public RichPanelFormLayout getForm()
   {
        if (listView != null)
        {
             return (RichPanelFormLayout) form.getComponent();
        }
        return null;
```

```
}
```

46. Save changes to AddressBookListenerBean and build the class.

Creating Address Book Detail Panel

1. From the Components palette, insert a Panel Tabbed component into the center facet of the outer Panel Stretch Layout.
2. On the Create Panel Tabbed form, set the **Text column** to Address Book and select the **Selected** radio button in the grid row.
3. Click the **OK** button to create the component.
4. From the Data Controls panel, drag the **addressBookDetail** member into the Show Detail Item of the Panel Tabbed component.
5. When prompted to specify the type of component to add, select **ADF Form**.
6. On the Create Form window, arrange the order of the fields and set the display labels as follows:
    - - addressNumber: Address Number
    - - alphaName: Name
    - - searchType: Search Type
    - - taxId: Tax Id
    - - businessUnit: Business Unit
    - - mailingName: Mailing Name
    - - address: Address
    - - city: City
    - - state: State
    - - zipCode: Zip Code
7. Click the **OK** button to add the component.
8. Set focus on the **Panel Form Layout component** and then select the **Edit** option next to the PartialTriggers property in the Properties panel.
9. On the Edit Property: PartialTriggers form, expand the start facet of the outer Panel Stretch Layout, expand the top facet and panel group layout components of the inner Panel Stretch Layout, and double-click button - **b1** from the Available panel to move it to the Selected panel. Click the **OK** button to set the attribute value.
10. Set the following attribute:

```
binding='#{addressBookListenerBean.form}'
```

11. Set the Alta skin following the instructions in *Skinning*.
12. The AddressBookList.jsff page fragment should look like the following:

```
<?xml version='1.0' encoding='UTF-8'?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:af="http://
xmlns.oracle.com/adf/faces/rich" xmlns:f="http://java.sun.com/jsf/core">
    <af:panelStretchLayout id="psl1" startWidth="350px" dimensionsFrom="parent"
 inlineStyle="margin:10px;">
        <f:facet name="bottom"/>
        <f:facet name="center">
            <af:panelTabbed position="above" id="pt1">
                <af:showDetailItem id="tab1" text="Address Book" disclosed="true">
                    <af:panelFormLayout id="pfl1" partialTriggers="b1"
 binding="#{addressBookListenerBean.form}">
```

```
                        <af:inputText value="#{bindings.addressNumber.inputValue}"
label="Address Number"

required="#{bindings.addressNumber.hints.mandatory}"
columns="#{bindings.addressNumber.hints.displayWidth}"

maximumLength="#{bindings.addressNumber.hints.precision}"
shortDesc="#{bindings.addressNumber.hints.tooltip}"
                                    id="it2">
                            <f:validator
binding="#{bindings.addressNumber.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.alphaName.inputValue}"
label="Name" required="#{bindings.alphaName.hints.mandatory}"

columns="#{bindings.alphaName.hints.displayWidth}"
maximumLength="#{bindings.alphaName.hints.precision}"

shortDesc="#{bindings.alphaName.hints.tooltip}" id="it3">
                            <f:validator binding="#{bindings.alphaName.validator}"/
>
                        </af:inputText>
                        <af:inputText value="#{bindings.searchType.inputValue}"
label="Search Type" required="#{bindings.searchType.hints.mandatory}"

columns="#{bindings.searchType.hints.displayWidth}"
maximumLength="#{bindings.searchType.hints.precision}"

shortDesc="#{bindings.searchType.hints.tooltip}" id="it4">
                            <f:validator
binding="#{bindings.searchType.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.taxId.inputValue}"
label="Tax Id" required="#{bindings.taxId.hints.mandatory}"

columns="#{bindings.taxId.hints.displayWidth}"
maximumLength="#{bindings.taxId.hints.precision}"
                                    shortDesc="#{bindings.taxId.hints.tooltip}"
id="it5">
                            <f:validator binding="#{bindings.taxId.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.businessUnit.inputValue}"
label="Business Unit"

required="#{bindings.businessUnit.hints.mandatory}"
columns="#{bindings.businessUnit.hints.displayWidth}"

maximumLength="#{bindings.businessUnit.hints.precision}"
shortDesc="#{bindings.businessUnit.hints.tooltip}"

                                    id="it6">
                            <f:validator
binding="#{bindings.businessUnit.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.mailingName.inputValue}"
label="Mailing Name"

required="#{bindings.mailingName.hints.mandatory}"
columns="#{bindings.mailingName.hints.displayWidth}"
```

```
maximumLength="#{bindings.mailingName.hints.precision}"
shortDesc="#{bindings.mailingName.hints.tooltip}"
                                    id="it7">
                            <f:validator
binding="#{bindings.mailingName.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.address.inputValue}"
label="Address" required="#{bindings.address.hints.mandatory}"

columns="#{bindings.address.hints.displayWidth}"
maximumLength="#{bindings.address.hints.precision}"
                                    shortDesc="#{bindings.address.hints.tooltip}"
id="it8">
                            <f:validator binding="#{bindings.address.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.city.inputValue}"
label="City" required="#{bindings.city.hints.mandatory}"
                                    columns="#{bindings.city.hints.displayWidth}"
maximumLength="#{bindings.city.hints.precision}"
                                    shortDesc="#{bindings.city.hints.tooltip}"
id="it9">
                            <f:validator binding="#{bindings.city.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.state.inputValue}"
label="State" required="#{bindings.state.hints.mandatory}"

columns="#{bindings.state.hints.displayWidth}"
maximumLength="#{bindings.state.hints.precision}"
                                    shortDesc="#{bindings.state.hints.tooltip}"
id="it10">
                            <f:validator binding="#{bindings.state.validator}"/>
                        </af:inputText>
                        <af:inputText value="#{bindings.zipCode.inputValue}"
label="Zip Code" required="#{bindings.zipCode.hints.mandatory}"

columns="#{bindings.zipCode.hints.displayWidth}"
maximumLength="#{bindings.zipCode.hints.precision}"
                                    shortDesc="#{bindings.zipCode.hints.tooltip}"
id="it11">
                            <f:validator binding="#{bindings.zipCode.validator}"/>
                        </af:inputText>
                    </af:panelFormLayout>
                </af:showDetailItem>
            </af:panelTabbed>
        </f:facet>
        <f:facet name="start">
            <af:panelStretchLayout id="psl2" topHeight="40px">
                <f:facet name="bottom"/>
                <f:facet name="center">
                    <af:listView value="#{bindings.rowset.collectionModel}"
var="item"
                                 emptyText="#{bindings.rowset.viewable ? 'No data
to display.' : 'Access Denied.'}"
                                 fetchSize="#{bindings.rowset.rangeSize}" id="lv1"

selectionListener="#{addressBookListenerBean.addressRecordSelected}"
                                 selection="single"
                                 binding="#{addressBookListenerBean.listView}"
partialTriggers="::b1">
```

```
                            <af:listItem id="li1">
                                <af:panelGroupLayout layout="horizontal" id="pgl2">
                                    <af:outputText
 value="#{item.mnAddressNumber_19.bindings.value.inputValue} -
 #{item.SAlphaName_20.bindings.value.inputValue}" id="ot1"/>
                                </af:panelGroupLayout>
                            </af:listItem>
                        </af:listView>
                    </f:facet>
                    <f:facet name="start"/>
                    <f:facet name="end"/>
                    <f:facet name="top">
                        <af:panelGroupLayout id="pgl1" layout="horizontal">
                            <af:inputText label="Name" id="it1"
                                          value="#{pageFlowScope.alphaNameFilter}"/>
                            <af:spacer width="12px" height="10" id="s1"/>
                            <af:button
 actionListener="#{addressBookListenerBean.searchAddressBook}" text="Search"
 id="b1"/>
                            <af:spacer width="8px" height="10" id="s2"/>
                            <af:button text="Close" action="exit" id="b2" rendered="#{!
pageFlowScope.pageEmbedded}"/>
                        </af:panelGroupLayout>
                    </f:facet>
                </af:panelStretchLayout>
            </f:facet>
            <f:facet name="end"/>
            <f:facet name="top"/>
        </af:panelStretchLayout>
</ui:composition>
```

Creating the ADF Test Page

1. In the Projects panel, right-click the **ViewController project**, select **New**, and then select **Page**.
2. On the Create JSF Page form, set the file name to **E01012TestPage.jsf**, verify the **Facelets** radio button is selected, and click the **OK** button. The new page will display in the main panel.
3. Change to source view.
4. Change the title **attribute of the af:document tag** to "E01012 Address Book."
5. From the Projects panel, select the **E01012_AddressBook_BTF** from the ViewController > Web Content > Page Flow folder and insert between the af:form tags. From the Create context menu that displays next, select **Region**.
6. The **E01012TestPage.jsf** should look like the following code:

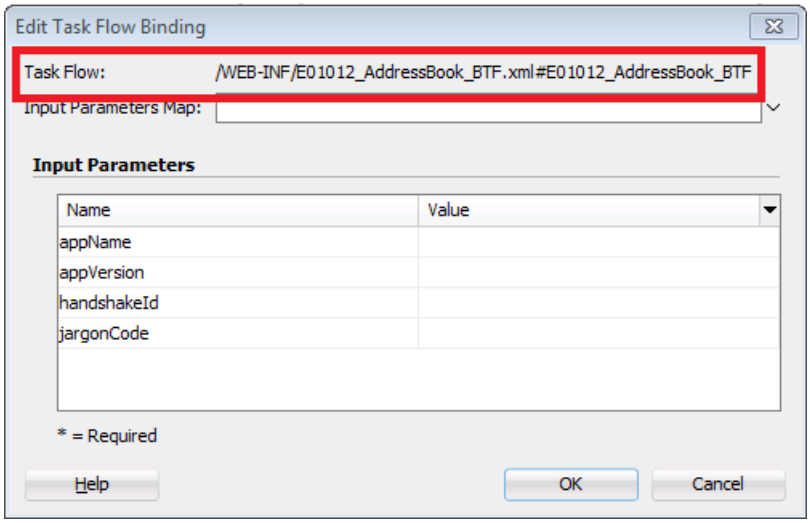
```
<f:view xmlns:f="http://java.sun.com/jsf/core" xmlns:af="http://xmlns.oracle.com/
adf/faces/rich">
    <af:document title="E01012 Address Book" id="d1">
        <af:form id="f1">
            <af:region value="#{bindings.E01012_AddressBook_BTF1.regionModel}"
 id="r1"/>
        </af:form>
    </af:document>
</f:view>
```

Finding the ADF Bounded Task Flow ID

Each ADF bounded task flow has a unique ID composed of the task flow definition ID and the document name. This composite ID is used when creating EnterpriseOne menu tasks for proxy applications or for assigning the External Application property on external forms (Release 9.2.1), both used to launch EnterpriseOne ADF applications.

To find the ADF bounded task flow ID

1. Open the test page created in the previous section and select the **Bindings** tab in the Editor window.
2. On the Binding Definition page, select the taskflow binding from the Executables list and click the **Edit** button.
3. The task flow ID is located at the top of the Edit Task Flow Binding form, as shown in the following graphic:



   Record the task flow ID value and enter it in the Path field for EnterpriseOne ADF application menu tasks in the Solution Explorer Interactive Application (P9000).

Alternatively, the task flow ID can be copied directly from the page bindings definition file:

1. Open the test page created in the previous section and select the **Bindings** tab in the Editor window.
2. On the Bindings Definition page, select the link to the Page Definition File, which will open the xml file in the Editor window.
3. Select and copy the taskFlowId attribute value from the taskFlow element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" version="12.1.3.10.8"
 id="E01012TestPagePageDef"
                 Package="com.oracle.e1.E01012.view.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <taskFlow id="E01012_AddressBook_BTF1" taskFlowId="/WEB-INF/
E01012_AddressBook_BTF.xml#E01012_AddressBook_BTF" activation="deferred"
              xmlns="http://xmlns.oracle.com/adf/controller/binding">
      <parameters>
        <parameter id="appName" value="E01012"/>
      </parameters>
    </taskFlow>
  </executables>
```

```
    <bindings/>
</pageDefinition>
```

Running the Test Page

1. In the Projects panel, expand the **ViewController and Web Content nodes**.
2. Right-click the **E01012TestPage.jsf** and select **Run**.

   JDeveloper will start the Integrated WebLogic Server, deploy the web application, and display the test page in the default browser. If you have not yet created the default Oracle WebLogic Server domain, JDeveloper will prompt you for an admin ID and password before starting the Integrated WebLogic Server.

Creating ADF Library JAR File

Once your EnterpriseOne ADF application is complete, it must be packaged in an ADF Library JAR file before it can be launched from an EnterpriseOne menu and run in the JDEADFContainer.

1. In the Projects panel, right-click the **ViewController** project, select **Deployment**, and then select **New Deployment Profile**.
2. On the Create Deployment Profile form, select **ADF Library JAR File** for the Profile Type and specify a **Deployment Profile Name**, like E01012, which will also be the name of the generated JAR file.
3. Click the **OK** button.
4. Right-click the **ViewController** project again, select **Deployment**, and select the new deployment profile.
5. Click **Finish**.

   The new JAR file will be stored in the deploy folder in the ViewController directory. Review *Building and Deploying EnterpriseOne ADF Applications* for instructions on building and deploying this JAR file within the EnterpriseOne ADF library WAR file.

Other Considerations

This section list additional considerations when creating and building EnterpriseOne ADF applications.

Translations

EnterpriseOne ADF applications are not translated so all labels must come from EnterpriseOne so that they can be translated. The following list describes two ways to accomplish this task:

1. Use Jargon service to get the values from the Data Dictionary. This is the recommended approach when you are using an existing EnterpriseOne application.

   See *"AIS Services (Endpoints)" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for a description of the Jargon service.
2. The second approach is recommended if you are creating a new EnterpriseOne application to support your EnterpriseOne ADF application. In your new EnterpriseOne application create a form (that will not be used by customers) and add Form Controls to it with the text values that need to be translated.

Skinning

If Skyros is the skin family for an EnterpriseOne ADF application, you need to modify the configuration file. This is because when the EnterpriseOne ADF application launches from an EnterpriseOne menu and runs inside the JDEADFContainer, the ADF application is rendered using the Alta skin instead of Skyros. To use the Alta skin when running the test page locally on JDeveloper's Integrated WebLogic Server, you need to modify the trinidad-config.xml file under ViewController/Web Content/WEB-INF to match the following:

<?xml version="1.0" encoding="windows-1252"?>

<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">

<skin-family>alta</skin-family>

<skin-version>v1</skin-version>

</trinidad-config>

It is important to note that the JDEADFContainer has extended the Alta skin, so only the Calendar component renders using Skyros styles. If your EnterpriseOne ADF application uses this component and you have changed the trinidad-config.xml file to use Alta, then the Calendar component will render differently when run locally versus running inside the JDEADFContainer.

13 Appendix B - Creating a Sample EnterpriseOne ADF Application for Data Service Aggregation Request

Creating a Sample EnterpriseOne ADF Application for Data Service Aggregation Request (Release 9.2.0.2)

Before You Begin

This appendix utilizes many of the same steps presented in Appendix A for creating an EnterpriseOne ADF application. The following sections primarily include new steps and code for executing data service aggregation requests and generating bar charts based on data included in the AIS Server responses. When appropriate, you will be directed back to sections in Appendix A to complete parts of the EnterpriseOne ADF application. The code presented in this appendix is also available in the E137001 JDeveloper workspace packaged with the OverstatedWorkOrderHours download from the Update Center.

The AIS Client Class Generator extension for JDeveloper should already be installed. See *Setting Up the Environment for Deploying EnterpriseOne ADF Applications*.

You must also have a basic understanding of how to use JDeveloper.

Creating the Overstated Work Order Hours ADF Application

You will need to follow the steps in section A.2 to create a new EnterpriseOne ADF application, specifying a unique name and package prefix.

See *Creating the Sample EnterpriseOne Address Book ADF Application*.

Generating Data Service Foundation Classes

Similar to section A.3.1, use the AIS Client Class Generator to create the classes needed to process responses to AIS data service requests.

Before continuing through this section, you should have completed the steps in the previous sections. You should also be able to use the AIS Client Class Generator.

Creating the Data Service Class

To create the data service class:

1. In the Projects panel located on the left of the screen, highlight the **Model** project,
2. Select **AIS Client Class Generator** from the Tools menu.
3. Enter the following information:

   Service Type: Data Service

   Target Type: Table

   Target Name: F4801

   Return Control Ids: F4801.NUMB|F4801.ANPA|F4801.ANP|F4801.HRSO|F4801.HRSA|F4801.PC

   FindOnEntry: <checked>

   DemoMode: <checked>

   Generate for Mobile: <unchecked>

   Preview JSON Data: <checked>

   Keep JSON Files: <unchecked>
4. Click the **Generate** button to execute the data service request and then click **Continue** to create the data service classes.
5. Enter the following information:

   - **Service Type:** Form Service
   - **Application Name:** E137001
   - **Form Name:** W137001A
   - **Version:** <blank>
   - **FormInputs:** <blank>
   - **FormServiceAction:** R
   - **ReturnControlIDs:** <blank>
   - **FindOnEntry:** <checked>
   - **DemoMode:** <checked>
   - **Generate for Mobile:** <unchecked>
   - **Preview JSON Data:** <checked>
   - **Keep JSON Files:** <unchecked>
6. Click the **Generate** button to execute the form service request and then click **Continue** to create the form service classes.

7. Refresh the **Model** project to view the newly generated classes.
   These classes will be used to deserialize the data service response from AIS, so the application data control can access the aggregated values returned from EnterpriseOne.

Creating ChartDataItem Helper Class

The ChartDataItem class is used to create a list of objects that contain the data required to render ADF DVT chart components. For this example, each object will contain a single, aggregated data value for display in a stacked bar chart. For other DVT chart components, this class can provide up to three data points to represent the object in the data visualization. For example, the ChartDataItem object would provide two data points (x and y) for a Scatter Chart component and three data points (x, y, and z) for a Bubble Chart component. However, other DVT components, like Gauges, Maps, and Gantt Charts, have different data model requirements to structure the data values specific to the component. Review the *Oracle® Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces* guide to find data model requirements for each DVT component.

To create the ChartDataItem class:

1. In the Projects panel located on the left of the screen, right-click the Model project, select **New**, and then select **From Gallery**.
2. On the New Gallery form, select **Java** from the Categories panel, and then select **Class** in the Items panel.
3. On the Create Java Class form, enter 'ChartDataItem' in the Name field.
4. Click **OK** to create the Java class.
5. Add the following fields to ChartDataItem.java:

```
private String series;
 private String group;
 private Double x;
 private Double y;
 private Double z;
```

6. Add the following constructors to ChartDataItem.java:

```
public ChartDataItem(String group, String series, Double x)
 {
       this.series = series;
    this.group = group;
      this.x = x;
 }

public ChartDataItem(String group, String series, Double x, Double y)
 {
      this(group, series, x);
      this.y = y;
 }

 public ChartDataItem(String group, String series, Double x, Double y, Double z)
 {
      this(group, series, x, y);
      this.z = z;
}
```

7. Right-click within the class and select **Generate Accessors...** from the context menu.
8. Select all methods and click **OK**.
9. Save changes to ChartDataItem.java.

Creating the Application Data Control

This section contains these topics:

- Creating the Data Control Class
- Adding Data Control Class Fields and Constants
- Updating Data Control Constructor
- Creating Accessor Methods
- Adding Error Handling Methods
- Adding Method to Retrieve Label Text
- Adding Accessor Methods for Labels
- Adding Method to Retrieve Data
- Adding Chart Refresh Method
- Generating Data Control

Creating the Data Control Class

To create the data control class:

1. In the Projects panel located on the left of the screen, right-click the **Model** project, select **New**, and then select **From Gallery**.
2. On the New Gallery form, select **Java** from the Categories panel, and then select **Class** in the Items panel.
3. On the Create Java Class form, enter **`EquipWorkOrderDC`** in the **Name** field.
4. Click **OK** to create the Java class.

Adding Data Control Class Fields and Constants

Add the following fields to EquipWorkOrderDC.java:

```
// Chart filter.
private String groupByFilter = GROUP_BY_EQUIP_NUM;

// Chart data.
private List<ChartDataItem> understatedChartData = new ArrayList<ChartDataItem>();
private List<ChartDataItem> overstatedChartData = new ArrayList<ChartDataItem>();

// Session
private E1UserSessionBean userBean;
private LoginEnvironment loginEnv;
private boolean runningInJDEADFContainer = true;


// Hard coded AIS connection values.
private static final String AIS_SERVER = "host:port";
private static final String USER_NAME = "user";
private static final String PASSWORD = "pwd";
private static final String DEVICE = "E1ADFApps";
```

```
    private static final String ROLE = "role";
    private static final String ENVIRONMENT = "env";

    // F4801 columns.
    private static final String ACTUAL_HOURS_ALIAS = "HRSA";
    private static final String EST_HOURS_ALIAS = "HRSO";
    private static final String EQUIP_NUMBER_ALIAS = "NUMB";
    private static final String ASSIGNED_TO_ALIAS = "ANP";
    private static final String SUPERVISOR_ALIAS = "ANPA";

    // Chart Type.
    private static final String GROUP_BY_EQUIP_NUM = "0";
    private static final String GROUP_BY_ASSIGNED_TO = "1";
    private static final String GROUP_BY_SUPERVISOR = "2";

    // Label Keys.
    private static final String HOURS_KEY = "Hours";
    private static final String SUPERVISOR_KEY = "ANPA";
    private static final String ASSIGNED_TO_KEY = "ANP";
    private static final String EQUIP_NUMBER_KEY = "EQPNUM";
    private static final String COMP_WO_TITLE_KEY = "CompWOChartTitle";
    private static final String IN_PROC_WO_TITLE_KEY = "InProcWOChartTitle";

    // Default label text.
    private static final String DEFAULT_JARGON = "13";

   // Capital Asset Management
    private static final String HOURS = "Hours";
    private static final String SUPERVISOR = "Supervisor";
    private static final String ASSIGNED_TO = "Assigned To";
    private static final String EQUIPMENT_NUMBER = "Equipment Number";
    private static final String COMP_WO_CHART_TITLE = "Completed Work Orders Actual Hours
Over Estimated";
    private static final String IN_PROC_WO_CHART_TITLE = "In Process Work Orders Actual
Hours Over Estimated";

    // Chart labels.
    private HashMap<String, String> labels = new HashMap<String, String>();
    {
        labels.put(HOURS_KEY, HOURS);
        labels.put(SUPERVISOR_KEY, SUPERVISOR);
        labels.put(ASSIGNED_TO_KEY, ASSIGNED_TO);
        labels.put(EQUIP_NUMBER_KEY, EQUIPMENT_NUMBER);
        labels.put(COMP_WO_TITLE_KEY, COMP_WO_CHART_TITLE );
        labels.put(IN_PROC_WO_TITLE_KEY, IN_PROC_WO_CHART_TITLE);
    }

  private static ADFLogger logger = ADFLogger.createADFLogger(EquipWorkOrderDC.class);
```

Updating Data Control Constructor

Add the following code to the Data Control class constructor, EquipWorkOrderDC():

```
   String jargonCode = (String)
ADFContext.getCurrent().getPageFlowScope().get("jargonCode");
   String handshakeId = (String)
ADFContext.getCurrent().getPageFlowScope().get("handshakeId");
```

```
    if (handshakeId == null || (handshakeId != null && handshakeId.length() == 0))
    {
        logger.finest("***** Handshake Id not passed to task flow.  Creating manual
connection. *****");
        runningInJDEADFContainer = false;
        userBean = new E1UserSessionBean(AIS_SERVER, USER_NAME, PASSWORD, ENVIRONMENT,
ROLE, DEVICE);
    }
    else
    {
        // Initialize application's about properties when running in the ADF container.
        E1AdfUtils.intializeAppInstance("/com/oracle/e1/E137001/");
    }

    loginEnv = E1AdfUtils.getLoginEnvironment(userBean);

    if (loginEnv != null)
    {
        logger.finest("***** Acquired valid LoginEnvironment object. *****");
        List<String> reqCapabilities = loginEnv.getRequiredCapabilities();
        reqCapabilities.add(AISClientCapability.JARGON);
        reqCapabilities.add(AISClientCapability.DATA_SERVICE);
        reqCapabilities.add(AISClientCapability.DATA_SERVICE_AGGREGATION);

        // Retrieve translated labels.
        retrieveLabelText(jargonCode);

        // Retrieve initial data set.
        retrieveData(groupByFilter);
    }
    else
    {
        // When a connection to E1 is not established, add a blank data item to each
chart's data list,
        // so empty charts display on the page .
        logger.finest("***** No chart data.  Adding empty data values to chart data lists.
*****");
        understatedChartData.add(new ChartDataItem(".", "", 0.0));
        overstatedChartData.add(new ChartDataItem(".", "", 0.0));

        logger.severe("***** Valid LoginEnvironment object is missing. *****");
        processErrorException("Connection failed. Please contact your system
administrator");
    }
```

The constructor will establish a connection with AIS and add AIS services used by the application to the required capabilities list.

Creating Accessor Methods

To create accessor methods:

1. Right-click within the class file in the Editor window and select **Generate Accessors...** from the context menu.
2. Select the **groupByFilter**, **understatedChartData**, and **overstatedChartData** fields from the Methods list.
3. Click **OK**.
4. Save changes to EquipWorkOrderDC.java.

Adding Error Handling Methods

Add the below methods to EquipWorkOrderDC.java. These methods display error messages to the user at runtime, regardless of whether the application is running in the JDEADFContainer web application or in a test page on the local Integrated WebLogic Server.

```
private void processErrorException(String msg)
{
    logger.severe(msg);
    displayMessage("Error", msg);
}

private void processErrorException(Exception e)
{
    logger.severe(e);
    displayMessage("Error", e.getMessage());
}

private void displayMessage(String title, String msg)
{
    if (runningInJDEADFContainer)
    {
        E1AdfUtils.addErrorMessage(msg);
        E1AdfUtils.launchMessagePopup();
    }
    else
    {
        ELContext elCtxt = ADFContext.getCurrent().getELContext();
         ExpressionFactory expFactory = ADFContext.getCurrent().getExpressionFactory();
         ValueExpression ve = expFactory.createValueExpression(elCtxt,
 "#{workOrderListenerBean}", Object.class);
         if (ve.getValue(elCtxt) != null)
         {
              MethodExpression me = expFactory.createMethodExpression(elCtxt,
 "#{workOrderListenerBean.displayLocalMessage}", Void.class, new Class[] {String.class,
 String.class});
              me.invoke(elCtxt, new Object[] {title, msg});
         }
    }
}
```

Adding a Method to Retrieve Label Text

Add the below method to EquipWorkOrderDC.java. This method retrieves translated text from EnterpriseOne using two methods and inserts the text into a label map accessed by the bar chart components at runtime. Some of the desired text is available in the EnterpriseOne Data Dictionary, so the AIS jargon service is executed to retrieve labels from the necessary data items. However, the chart titles and y-axis label cannot be found in Data Dictionary, so a form service request is also executed to retrieve this text from form controls added to the EnterpriseOne proxy app, E137001. If either AIS service request fails, the label map is pre-populated with hard-coded labels in English, so the bar charts always have labels.

```
    private void retrieveLabelText(String jargonCode)
    {
```

```
        // Perform FSR to get label text from E137001 proxy app.
        FormRequest proxyAppRequest = new FormRequest(loginEnv);
        proxyAppRequest.setFormName("E137001_W137001A");
        proxyAppRequest.setFormServiceAction("R");

        FSREvent formEvent = new FSREvent();
        formEvent.doControlAction("12");        // Press Cancel button.
        proxyAppRequest.addFSREvent(formEvent);

        try
        {
            String formResponse = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
proxyAppRequest, JDERestServiceProvider.POST_METHOD,
JDERestServiceProvider.FORM_SERVICE_URI);
            E137001_W137001A_FormParent formParent =
loginEnv.getObjectMapper().readValue(formResponse, E137001_W137001A_FormParent.class);

            if (formParent != null)
            {
                // Add form text to label map.
                E137001_W137001A_FormData formData =
formParent.getFs_E137001_W137001A().getData();
                labels.put(HOURS_KEY, formData.getTxtHours_15().getTitle());
                labels.put(COMP_WO_TITLE_KEY,
formData.getTxtCompletedWorkOrdersChartTitle_17().getTitle());
                labels.put(IN_PROC_WO_TITLE_KEY,
formData.getTxtInProcessWorkOrdersChartTitle_19().getTitle());
            }

            // Perform jargon request to retrieve remaining labels.
            if (jargonCode == null || jargonCode.trim().isEmpty())
            {
                jargonCode = DEFAULT_JARGON;
            }

            // Retrieve label text for Equipment Number, Assigned To, and Supervisor.
            JargonRequest jargonReq = new JargonRequest(loginEnv, jargonCode);
            jargonReq.addDataItem(EQUIP_NUMBER_KEY);
            jargonReq.addDataItem(ASSIGNED_TO_KEY);
            jargonReq.addDataItem(SUPERVISOR_KEY);

            String jargonResponse = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
jargonReq, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.JARGON_SERVICE);
            JargonResponse jargonResp =
loginEnv.getObjectMapper().readValue(jargonResponse, JargonResponse.class);

            if (jargonResp != null)
            {
                // Add jargon text to label map.
                List<JargonResponseItem> jargonItems = jargonResp.getRequestedItems();
                if (jargonItems != null && jargonItems.size() > 0)
                {
                    for (JargonResponseItem item: jargonItems)
                    {
                        labels.put(item.getSzDict().trim().toUpperCase(),
item.getRowDescription());
                    }
                }
            }
        }
```

```
        catch (JDERestServiceException e)
        {
            processErrorException(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            processErrorException(e);
        }
    }
```

Adding Accessor Methods for Labels

Add the below methods to EquipWorkOrderDC.java. These methods access the label map at runtime to retrieve the text that will be displayed on the page.

```
    public String getChartXAxisTitle()
    {
        if (groupByFilter.equals(GROUP_BY_EQUIP_NUM))
        {
            return labels.get(EQUIP_NUMBER_KEY);
        }
        else if (groupByFilter.equals(GROUP_BY_ASSIGNED_TO))
        {
            return labels.get(ASSIGNED_TO_KEY);
        }
        else
        {
            return labels.get(SUPERVISOR_KEY);
        }
    }

    public String getChartYAxisTitle()
    {
        return labels.get(HOURS_KEY);
    }

    public String getEquipmentNumberItemText()
    {
        return labels.get(EQUIP_NUMBER_KEY);
    }

    public String getAssignedToItemText()
    {
        return labels.get(ASSIGNED_TO_KEY);
    }

    public String getSupervisorItemText()
    {
        return labels.get(SUPERVISOR_KEY);
    }

    public String getOverstatedChartTitle()
    {
        return labels.get(IN_PROC_WO_TITLE_KEY);
    }

    public String getUnderStatedChartTitle()
```

```
    {
    }
        return labels.get(COMP_WO_TITLE_KEY);
```

Adding Method to Retrieve Data

Add the below method to EquipWorkOrderDC.java. This method retrieves aggregated work order data from the F4801 in EnterpriseOne and formats the results. The first part creates two data service aggregation requests to summarize estimated and actual hours for in-process and completed work orders. Once the responses are received from the AIS Server, they are processed separately to add the aggregated sum amounts to the two lists bound to the DVT bar chart components. See *Executing AIS Calls for Retrieving Data* for additional details on data service aggregation requests.

```
    private void retrieveData(String chartType)
    {
        // For demo purposes, only display the first ten data entries in either chart.
        int understatedDataCount = 0;
        int overstatedDataCount = 0;

        // Clear chart data lists before adding new data items.
        understatedChartData.clear();
        overstatedChartData.clear();

        try
        {
            // Create F4801 aggregation data request to sum actual and estimated hours
for
            // completed work orders, ordered by equipment, assigned to, or supervisor.
            DataRequest f4801DataReq1 = new DataRequest(loginEnv);
            f4801DataReq1.setDataServiceType(DataRequest.TYPE_AGGREGATION);
            f4801DataReq1.setTargetName("F4801");
            f4801DataReq1.setTargetType(DataRequest.TARGET_TABLE);

            // Sum estimated and actual hours.
            AggregationInfo sumHoursAgg1 = new AggregationInfo(loginEnv);
            sumHoursAgg1.addAggregationColumn(ACTUAL_HOURS_ALIAS,
AggregationType.AGG_TYPE_SUM());
            sumHoursAgg1.addAggregationColumn(EST_HOURS_ALIAS,
AggregationType.AGG_TYPE_SUM());

            // Group data according to user selection.
            if (chartType.equals(GROUP_BY_ASSIGNED_TO))
            {
                sumHoursAgg1.addAggregationGroupBy(ASSIGNED_TO_ALIAS);
                sumHoursAgg1.addAggregationOrderBy(ASSIGNED_TO_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
            }
            else if (chartType.equals(GROUP_BY_SUPERVISOR))
            {
                sumHoursAgg1.addAggregationGroupBy(SUPERVISOR_ALIAS);
                sumHoursAgg1.addAggregationOrderBy(SUPERVISOR_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
            }
            else
            {
                sumHoursAgg1.addAggregationGroupBy(EQUIP_NUMBER_ALIAS);
                sumHoursAgg1.addAggregationOrderBy(EQUIP_NUMBER_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
```

```
            }

            // Add aggregation to data request.
            f4801DataReq1.setAggregation(sumHoursAgg1);

            // Create query to retrieve F4801 data where actual hours are greater than
zero and
            // completion percentage is 100.00.
            Query completedWOQuery = new Query(loginEnv);
            completedWOQuery.setAutoFind(true);
            completedWOQuery.setMatchType(Query.MATCH_ALL);

            // Add query conditions.
            completedWOQuery.addNumberCondition("F4801.HRSA", NumericOperator.GREATER(),
new BigDecimal(0.00));
            completedWOQuery.addNumberCondition("F4801.PC", NumericOperator.EQUAL(), new
BigDecimal(100.00));

            // Add Query to data request.
            f4801DataReq1.setQuery(completedWOQuery);

            // Create F4801 aggregation data request to sum actual and estimated hours
for
            // in process work orders, ordered by equipment, assigned to, or supervisor.
            DataRequest f4801DataReq2 = new DataRequest(loginEnv);
            f4801DataReq2.setDataServiceType(DataRequest.TYPE_AGGREGATION);
            f4801DataReq2.setTargetName("F4801");
            f4801DataReq2.setTargetType(DataRequest.TARGET_TABLE);

            // Sum estimated and actual hours.
            AggregationInfo sumHoursAgg2 = new AggregationInfo(loginEnv);
            sumHoursAgg2.addAggregationColumn(ACTUAL_HOURS_ALIAS,
AggregationType.AGG_TYPE_SUM());
            sumHoursAgg2.addAggregationColumn(EST_HOURS_ALIAS,
AggregationType.AGG_TYPE_SUM());

            // Group data according to user selection.
            if (chartType.equals(GROUP_BY_ASSIGNED_TO))
            {
                logger.finest("**** Grouping data by Assigned To (ANP) field *****");
                sumHoursAgg2.addAggregationGroupBy(ASSIGNED_TO_ALIAS);
                sumHoursAgg2.addAggregationOrderBy(ASSIGNED_TO_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
            }
            else if (chartType.equals(GROUP_BY_SUPERVISOR))
            {
                logger.finest("**** Grouping data by Supervisor (ANPA) field *****");
                sumHoursAgg2.addAggregationGroupBy(SUPERVISOR_ALIAS);
                sumHoursAgg2.addAggregationOrderBy(SUPERVISOR_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
            }
            else
            {
                logger.finest("**** Grouping data by Equipment Number (NUMB) field
*****");
                sumHoursAgg2.addAggregationGroupBy(EQUIP_NUMBER_ALIAS);
                sumHoursAgg2.addAggregationOrderBy(EQUIP_NUMBER_ALIAS,
OrderByDirection.ORDER_DIRECT_ASCENDING());
            }
```

```
            // Add aggregation to second data request.
            f4801DataReq2.setAggregation(sumHoursAgg2);

            // Create query to retrieve F4801 data where actual hours are greater than
zero and
            // completion percentage is less than 100.00.
            Query inProcessWOQuery = new Query(loginEnv);
            inProcessWOQuery.setAutoFind(true);

            inProcessWOQuery.setMatchType(Query.MATCH_ALL);

            // Add query conditions.
            inProcessWOQuery.addNumberCondition("F4801.HRSA", NumericOperator.GREATER(),
new BigDecimal(0.00));
            inProcessWOQuery.addNumberCondition("F4801.PC", NumericOperator.LESS(), new
BigDecimal(100.00));

            // Add query to second data request.
            f4801DataReq2.setQuery(inProcessWOQuery);
            // Batch data requests.
            BatchDataRequest batchDataRequest = new BatchDataRequest(loginEnv);
            batchDataRequest.getDataRequests().add(f4801DataReq1);
            batchDataRequest.getDataRequests().add(f4801DataReq2);

            // Execute data service request
            logger.finest("***** Executing Data Aggregation Service - Start *****");
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
batchDataRequest, JDERestServiceProvider.POST_METHOD,
JDERestServiceProvider.DATA_SERVICE_URI);
            logger.finest("***** Executing Data Aggregation Service - Complete *****");

            if (response != null)
            {
                // Process data service responses.
                ArrayNode array1 =
AggregationResponseHelper.getAggregateValuesArrayBatch(response, "F4801", 0);
                for (JsonNode node : array1)
                {
                    JsonNode groupByInfo = node.get("HRSA_SUM");
                    Double actualHours =  groupByInfo.asDouble();
                    JsonNode groupBy2 = node.get("HRSO_SUM");
                    Double estHours = groupBy2.asDouble();

                    // Build list of ChartDataItem objects for complete work order chart.
                    // The group by value is assigned as both the group and series value.
                    // The difference in actual and estimated sum amounts is x value.
                    Double result = actualHours - estHours;
                    if (result > 0.0)
                    {
                        if (understatedDataCount < 10)
                        {
                            // Create entry for the bar chart.
                            understatedDataCount++;
                            ChartDataItem dataItem;
                            if (chartType.equals(GROUP_BY_ASSIGNED_TO))
                            {
                                // Get Assigned To value.
                                String assignedTo =
node.get(AggregationResponseHelper.GROUP_BY).get(ASSIGNED_TO_ALIAS).asText().trim();
```

```
                                dataItem = new ChartDataItem(assignedTo, assignedTo,
result);
                            }
                            else if (chartType.equals(GROUP_BY_SUPERVISOR))
                            {
                                // Get Supervisor value.
                                String supervisor =
node.get(AggregationResponseHelper.GROUP_BY).get(SUPERVISOR_ALIAS).asText().trim();
                                dataItem = new ChartDataItem(supervisor, supervisor,
result);
                            }
                            else
                            {
                                // Get equipment number value.
                                String equipment =
node.get(AggregationResponseHelper.GROUP_BY).get(EQUIP_NUMBER_ALIAS).asText().trim();
                                dataItem = new ChartDataItem(equipment, equipment,
result);
                            }
                            understatedChartData.add(dataItem);
                        }
                        else
                        {
                            // Stop processing the result set.
                            break;
                        }
                    }
                }

                // Process sum results for work orders that are not yet complete.
                ArrayNode array2 =
AggregationResponseHelper.getAggregateValuesArrayBatch(response, "F4801", 1);
                for (JsonNode node : array2)
                {
                    JsonNode groupByInfo = node.get("HRSA_SUM");
                    Double actualHours =  groupByInfo.asDouble();
                    JsonNode groupBy2 = node.get("HRSO_SUM");
                    Double estHours = groupBy2.asDouble();

                    // Build list of ChartDataItem objects for in process Work order
chart.
                    // The group by value is assigned as both the group and series value.
                    // The difference in actual and estimated sum amounts is x value.
                    Double result = actualHours - estHours;
                    if (result > 0.0)
                    {
                        if (overstatedDataCount < 10)
                        {
                            // Create entry for the bar chart.
                            overstatedDataCount++;
                            ChartDataItem dataItem;
                            if (chartType.equals(GROUP_BY_ASSIGNED_TO))
                            {
                                // Get Assigned To value.
                                String assignedTo =
node.get(AggregationResponseHelper.GROUP_BY).get(ASSIGNED_TO_ALIAS).asText().trim();
                                dataItem = new ChartDataItem(assignedTo, assignedTo,
result);
                            }
                            else if (chartType.equals(GROUP_BY_SUPERVISOR))
```

```
                                {
                                    // Get Supervisor value.
                                    String supervisor =
node.get(AggregationResponseHelper.GROUP_BY).get(SUPERVISOR_ALIAS).asText().trim();
                                    dataItem = new ChartDataItem(supervisor, supervisor,
result);

                                }
                                else
                                {
                                    //  Get equipment number value.
                                    String equipment =
node.get(AggregationResponseHelper.GROUP_BY).get(EQUIP_NUMBER_ALIAS).asText().trim();
                                    dataItem = new ChartDataItem(equipment, equipment,
result);
                                }
                                overstatedChartData.add(dataItem);
                            }
                            else
                            {
                                // Stop processing the result set.
                                break;
                            }
                        }
                    }

                    // If either chart's data set is empty, add a blank data item to the list,
so blank charts
                    // continue to display on the page.
                    if (understatedChartData.isEmpty())
                    {
                        logger.finest("***** No chart data.  Adding empty value to chart data
list. *****");
                        understatedChartData.add(new ChartDataItem(".", "", 0.0));
                    }

                    if (overstatedChartData.isEmpty())
                    {
                        logger.finest("***** No chart data.  Adding empty value to chart data
list. *****");
                        overstatedChartData.add(new ChartDataItem(".", "", 0.0));
                    }
                }
                else
                {
                    logger.finest("***** Data Service Response is null.  No data to process.
*****");
                }
            }
            catch (CapabilityException e)
            {
                processErrorException(e);
            }
            catch (JDERestServiceException e)
            {
                processErrorException(JDERestServiceProvider.handleServiceException(e));
            }
            catch (Exception e)
            {
                processErrorException(e);
```

```
        }
    }
```

Adding Chart Refresh Method

Add the following method to EquipWorkOrderDC.java. This method calls the retrieveData() method to execute a data service aggregation request that retrieves a fresh set of data from EnterpriseOne. This method is normally called when the user selects a different radio button to summarize by a different F4801 column.

```
    public void refreshChartData(String chartType)
    {
        if (loginEnv != null)
        {
            logger.finest("***** Retrieving new chart data. *****");
            retrieveData(chartType);
        }
        else
        {
            logger.severe("***** LoginEnvironment is null.  Unable to retrieve new chart
data. *****");
            processErrorException("Connection failed. Please contact your system
administrator");
        }
    }
```

Generating Data Control

To generate a data control:

1. Right-Click the **EquipWorkOrderDC** in the Projects Panel and select **Create Data Control** from the context menu.
2. Accept the defaults and click **Next, Next, Finish**.
3. Refresh the Data Controls list to see the EquipWorkOrderDC data control.

Creating the Task Flow

To create the bounded task flow for the EnterpriseOne ADF application, see *Creating a Task Flow*. When creating the bounded task flow, specify unique names for the task flow and the View component added to the new task flow. Make sure a JSF fragment is created for the View component added to the bounded task flow.

Adding ADF Face Components to the View Page Fragment

To add ADF face components to the View Page fragment:

1. In the Projects panel located on the left, expand the **ViewController, Web Content**, and **Fragments** nodes and double-click the page fragment generated for the View object you added to the bounded task flow in section B.6.

   This will display the page fragment in the design panel in the middle.
2. Select the **Source** tab at the bottom of the panel. Insert a **Panel Stretch Layout** between the ui:composition tags from the **Components** palette and set the following attribute:
   - **dimensionsFrom**: parent
3. Insert a **Panel Group Layout** component inside the top facet of the Panel Stretch Layout. Set the following attributes:
   - **layout**: scroll
   - **halign**: end
4. Insert a **Panel Group Layout** component inside the first Panel Group layout. Set the following attribute:
   - **layout**: horizontal
5. From the Data Controls panel, drag the **groupByFilter** in EquipWorkOrderDC into the inner panelGroupLayout component.

   You will be prompted to specify the type of component to add.
6. Select **Text > ADF> ADF Output Text**. This will add an **outputText** component to the page fragment and create a page binding for the **groupByFilter** field.
7. From the Components panel, drag a **Radio Group** component inside the inner panelGroupLayout component after the outputText component.

   On the Insert Radio Group form, select the **Create List** radio button and add the following entries to the list using the green '+' button:

Item Label	Item Value
Equipment	0
Assigned To	1
Supervisor	2

   Click **Finish**. A selectOneRadio component is added to the fragment with a selectItem component for each valid radio button value.

8. Delete the label attribute from the **selectOneRadio** component.
9. Copy the value attribute from the **outputText** component to the **selectOneRadio** component to bind the radio button group to the **groupByFilter** field in the data control.
10. Set the following attributes on the **selectOneRadio** component:
    - **layout:** horizontal
    - **autoSubmit:** true
    - **simple:** true
    - **inlineStyle: margin:** 10px 10px 0 10px;
11. Set focus on the **selectOneRadio** component and then select the **Edit** option next to the **ValueChangeListener** property in the Properties panel.
12. On the Edit Property: ValueChangeListener form, click the **New** button next to the Managed Bean drop-down list, and set the following fields:
    - **Bean Name:** workOrderListenerBean
    - **Class Name:** WorkOrderListenerBean
    - **Scope:** request (Use default)
13. Click **OK** to create the class and the managed bean declaration in the page flow XML file.
14. Select the **New** button next to the Method drop-down list and set the following field:
    - **Method Name:** radioGroupValueChanged
15. Click **OK** to create the method and click **OK** again to close the Edit Property form.
16. In the Projects panel, expand the **ViewController** and **Application Sources** nodes, and open the WorkOrderListenerBean.java class in the Editor window.
17. Add the code shown below to WorkOrderListenerBean.java:

```
public void radioGroupValueChanged(ValueChangeEvent event)
 {
     // Execute data control method to refresh data.
     DCBindingContainer bindings = (DCBindingContainer)
 BindingContext.getCurrent().getCurrentBindingsEntry();
     DCIteratorBinding iter =
 bindings.findIteratorBinding("EquipWorkOrderDCIterator");
     if (iter != null)
     {
         DCDataControl dc = iter.getDataControl();
         EquipWorkOrderDC equipWorkOrderDC = (EquipWorkOrderDC)
 dc.getDataProvider();

         if (equipWorkOrderDC != null)
         {
             equipWorkOrderDC.refreshChartData(event.getNewValue().toString());

             OperationBinding refreshOverstatedChart =
 bindings.getOperationBinding("Execute");
             OperationBinding refreshUnderstatedChart =
 bindings.getOperationBinding("Execute1");
             if (refreshOverstatedChart != null && refreshUnderstatedChart != null)
             {
                 refreshOverstatedChart.execute();
                 refreshUnderstatedChart.execute();
             }
```

```
            }
        }
}

public void displayLocalMessage(String title, String msg)
{
    FacesContext.getCurrentInstance().addMessage("E1ADFApp", new
 FacesMessage(FacesMessage.SEVERITY_ERROR, title, msg));
}
```

Save the changes to the class.

18. From the Data Controls panel, drag the **equipmentNumberItemText, assignedToItemText,** and **supervisorItemText** fields in EquipWorkOrderDC into the inner panelGroupLayout component below the selectOneRadio component and create an Output Text component for each field. Again, this will create a page binding for each field.
19. Copy the value of the value attribute for the **equipmentNumberItemText outputText** component and paste the value into the label attribute of the **Equipment selectItem** component in the **selectOneRadio** component.
20. Copy the value of the value attribute for the **assignedToItemText outputText component** and paste the value into the label attribute of the **Assigned To selectItem** component in the **selectOneRadio** component.
21. Copy the value of the value attribute for the **supervisorItemText outputText** component and paste the value into the label attribute of the **Supervisor selectItem** component in the **selectOneRadio** component.
22. From the Components palette, insert a **Panel Grid Layout** component inside the center facet of the Panel Stretch Layout.
23. On the Create Panel Grid Layout form, set the following values:
    - **Columns:** 2
    - **Rows:** 1
24. Click **Finish**.
25. Set the following attribute of the panelGridLayout component:
    - **dimensionsFrom**: parent
26. Set focus on the panelGridLayout component and then select the **Edit** option next to the PartialTriggers property in the Properties panel.
27. On the Edit Property: PartialTriggers form, expand the top facet and both panelGroupLayout components, and double-click **selectOneRadio-sor1** from the Available panel to move it to the Selected Panel.
28. Click **OK** to set the attribute value.
29. Set the following attribute of the gridRow component:
    - **height**: 100%
30. Set the following attributes of the first gridCell component:
    - **marginEnd**: 15px
    - **width:** 50%
    - **halign**: stretch
    - **valign**: stretch
31. Set the following attributes of the second gridCell component:
    - **marginStart:** 15px
    - **width:** 50%

- **halign:** stretch
- **valign:** stretch

32. From the Data Controls panel, drag the **overstatedChartTitle, underStatedChartTitle, chartXAxisTitle,** and **chartYAxisTitle** fields into the first gridCell component and create Output Text components for each field. Again, this will create a page binding for each field.
33. From the Data Controls panel, drag the **overstatedChartData** field into the first gridCell component.

    You are prompted to specify the type of component to add.
34. Select **Chart**.

    On the Component Gallery form, select **Bar** from the Categories list on the left, choose **Stacked Bar** from the Chart Types section at the top, pick the second layout in the Quick Start Layouts section at the bottom, and click **OK**.
35. On the Create Stacked Bar Chart form, drag the **x** field from the Available list on the left to the **Bars** field on the right. Then drag the **group** field from the Available list to the **X Axis** field on the right.

    Click **OK**.
36. Set the following attributes for the barChart component:
    - **titleHalign:** center
    - **inlineStyle:** "font-weight:bold;"
37. Copy the value of the value attribute for the overstatedChartTitle outputText component and paste it into the title attribute of the barChart component.
38. Set the following attribute for the chartDataItem component:
    - **series:** #{item.series}
39. From the Components palette, add the **chartLegend** component to the barChart component and set the rendered attribute to false.
40. From the Components palette, add the **chartXAxis** and **chartYAxis** components to the barChart component.
41. Copy the value of the value attribute for the chartXAxisTitle outputText component and paste it into the title attribute of the chartXAxis component.
42. Copy the value of the value attribute for the chartYAxisTitle outputText component and paste it into the title attribute of the chartYAxis component.
43. From the Data Controls panel, drag the **understatedChartData** field into the second gridCell component.

    You will be prompted to specify the type of component to add.
44. Select **Chart**. On the Component Gallery form, select **Bar** from the Categories list on the left, choose **Stacked Bar** from Chart Types section at the top, pick the second layout in the Quick Start Layouts section at the bottom, and click **OK**.
45. On the Create Stacked Bar Chart form, drag the **x** field from the Available list on the left to the **Bars** field on the right. Then drag the **group** field from the Available list to the **X Axis** field on the right. Click **OK.**
46. Set the following attributes for the barChart component:
    - **titleHalign**: center
    - **inlineStyle:** "font-weight:bold;"
47. Copy the value of the value attribute for the underStatedChartTitle outputText component and paste it into the title attribute of the barChart component.
48. Set the following attribute for the chartDataItem component:
    - **series**: #{item.series}

49. From the Components palette, add the **chartLegend** component to the barChart component and set the rendered attribute to false.
50. From the Components palette, add the **chartXAxis** and **chartYAxis** components to the barChart component.
51. Copy the value of the value attribute for the chartXAxisTitle outputText component and paste it into the title attribute of the chartXAxis component.
52. Copy the value of the value attribute for the chartYAxisTitle outputText component and paste it into the title attribute of the chartYAxis component.
53. Delete all outputText components from the page fragment.
54. Select the **Bindings** tab at the bottom of the panel.
55. In the Bindings list on the left, click the green '**+**' icon to add a new binding
56. On the Insert Item form, select **action** from the list and click **OK**.
57. On the Create Action Binding form, expand the EquipWorkOrderDC, overstatedChartData, and Operations nodes and select the **Execute** operation.
58. Click **OK** to add the action binding.
59. Click the green '**+**' icon again to add a second binding.
60. On the Insert Item form, select **action** from the list and click **OK**.
61. On the Create Action Binding form, expand the EquipWorkOrderDC, understatedChartData, and Operations nodes and select the **Execute** operation.
62. Click **OK** to add the action binding.
63. In the Bindings list, select **overstatedChartData** and press the pencil icon to edit the binding.
64. On the Edit Tree Binding form, move series from the Available Attributes list to the Display Attributes list. Click **OK** to save changes.
65. In the Bindings list, select **understatedChartData** and press the pencil icon to edit the binding.
66. On the Edit Tree Binding form, move series from the Available Attributes list to the Display Attributes list. Click **OK** to save changes.
67. Perform a final **Save** for the page fragment.

    The page fragment should look like this:

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:af="http://
xmlns.oracle.com/adf/faces/rich" xmlns:f="http://java.sun.com/jsf/core"
                xmlns:dvt="http://xmlns.oracle.com/dss/adf/faces">
    <af:panelStretchLayout id="psl1" dimensionsFrom="parent">
        <f:facet name="bottom"/>
        <f:facet name="center">
            <af:panelGridLayout id="pgl3" dimensionsFrom="parent"
 partialTriggers="sor1">
                <af:gridRow marginTop="5px" height="100%" marginBottom="5px"
 id="gr1">
                    <af:gridCell marginStart="5px" width="50%" id="gc1"
 marginEnd="15px" halign="stretch" valign="stretch">
                        <dvt:barChart stack="on" id="barChart1" var="row"
 value="#{bindings.overstatedChartData.collectionModel}"
                                      titleHalign="center" inlineStyle="font-
weight:bold;" title="#{bindings.overstatedChartTitle.inputValue}">
                            <dvt:chartLegend id="cl1" rendered="false"/>
                            <dvt:chartXAxis id="cxa1"
 title="#{bindings.chartXAxisTitle.inputValue}"/>
                            <dvt:chartYAxis id="cya1"
 title="#{bindings.chartYAxisTitle.inputValue}"/>
                            <f:facet name="dataStamp">
                                <dvt:chartDataItem id="di1" value="#{row.x}"
 group="#{row.group}" series="#{row.series}"/>
```

```
                                </f:facet>
                            </dvt:barChart>
                        </af:gridCell>
                        <af:gridCell marginStart="15px" width="50%" marginEnd="5px"
id="gc2" halign="stretch" valign="stretch">
                            <dvt:barChart stack="on" id="barChart2" var="row"
value="#{bindings.understatedChartData.collectionModel}"
                                          titleHalign="center" inlineStyle="font-weight:
bold;" title="#{bindings.underStatedChartTitle.inputValue}">
                                <dvt:chartLegend id="cl2" rendered="false"/>
                                <dvt:chartXAxis id="cxa2"
title="#{bindings.chartXAxisTitle.inputValue}"/>
                                <dvt:chartYAxis id="cya2"
title="#{bindings.chartYAxisTitle.inputValue}"/>
                                <f:facet name="dataStamp">
                                    <dvt:chartDataItem id="di2" value="#{row.x}"
group="#{row.group}" series="#{row.series}"/>
                                </f:facet>
                            </dvt:barChart>
                        </af:gridCell>
                    </af:gridRow>
                </af:panelGridLayout>
            </f:facet>
            <f:facet name="start"/>
            <f:facet name="end"/>
            <f:facet name="top">
                <af:panelGroupLayout id="pgl1" layout="scroll" halign="end">
                    <af:panelGroupLayout id="pgl2" layout="horizontal">
                        <af:selectOneRadio id="sor1"
value="#{bindings.groupByFilter.inputValue}"
                                           layout="horizontal" autoSubmit="true"
simple="true" inlineStyle="margin: 10px 10px 0 10px;"

valueChangeListener="#{workOrderListenerBean.radioGroupValueChanged}">
                            <af:selectItem
label="#{bindings.equipmentNumberItemText.inputValue}" value="0" id="si1"/>
                            <af:selectItem
label="#{bindings.assignedToItemText.inputValue}" value="1" id="si2"/>
                            <af:selectItem
label="#{bindings.supervisorItemText.inputValue}" value="2" id="si3"/>
                        </af:selectOneRadio>
                    </af:panelGroupLayout>
                </af:panelGroupLayout>
            </f:facet>
        </af:panelStretchLayout>
    </ui:composition>
```

Changing the Assigned Skin to Alta

To set the application skin to Alta, see *Skinning*.

14 Appendix C - Manually Building and Deploying EnterpriseOne ADF Applications

Manually Building and Deploying EnterpriseOne ADF Applications

Note: Starting with Tools Release 9.2.1.2, you have the option of deploying the ADF applications through Server Manager instead of performing the manual steps listed here. For more information, see "Install an Application Development Framework Server Instance" in the *JD Edwards EnterpriseOne Tools Server Manager Guide* .

Understanding the Manual Process

EnterpriseOne ADF applications cannot be deployed and run as a standalone application. You must deploy them with the EnterpriseOne ADF Container application, which provides a login feature for accessing deployed EnterpriseOne ADF applications, as well as About and Help features for EnterpriseOne ADF applications running inside the container.

When EnterpriseOne ADF applications are created, they are packaged into JAR files. Oracle provides a build script (Ant script) that you can use to generate a WAR file from the application JAR file. You can then use this same build script to deploy the EnterpriseOne ADF application WAR file to an Oracle WebLogic Server with ADF runtime.

Oracle also provides a build script for configuring and deploying the EnterpriseOne ADF Container application.

You must deploy EnterpriseOne ADF applications and the EnterpriseOne ADF Container on the same domain.

Prerequisites

Complete the following prerequisites before deploying EnterpriseOne ADF applications:

- *Set Up an Environment for Deploying EnterpriseOne ADF Applications*
- *Set Up Oracle WebLogic Server to Use wldeploy*
- *Download EnterpriseOne ADF Foundation*
- *Download a Prebuilt EnterpriseOne ADF Application (Optional)*

Set Up an Environment for Deploying EnterpriseOne ADF Applications

If you have not already done so, set up the environment as described in *Setting Up the Environment for Deploying EnterpriseOne ADF Applications* of this guide.

Set Up Oracle WebLogic Server to Use wldeploy

The build scripts use the wldeploy task to deploy artifacts to server instances or clusters on Oracle WebLogic Server. Therefore, you need to set up the environment to use the wldeploy task according to instructions in the Basic Steps for Using wldeploy section of the "wldeploy Ant Task Reference" chapter in the *Oracle® Fusion Middleware: Developing Applications for Oracle WebLogic Server12c Release 1(12.1.1)* guide.

When following the instructions in this chapter, remember these details about the build scripts:

- The scripts must execute on a machine with an Oracle WebLogic Server installation to access the wldeploy task definition class. The server can be a standalone Oracle WebLogic Server or an integrated Oracle WebLogic Server instance included with a JDeveloper install.
- You execute the deploy targets from the Ant scripts using the command line.

Download EnterpriseOne ADF Foundation

Download the latest E1_ADF_Foundation_1.x.x from the JD Edwards Update Center (`https://updatecenter.oracle.com`). To locate the download, enter `EnterpriseOne ADF` in the Type field and select `All 9.x Releases` in the Release field and click **Search**.

E1_ADF_Foundation_1.x.x contains the following files:

- Ant Build Scripts (folder). This folder contains the following files that you use to build EnterpriseOne ADF applications for deployment in the EnterpriseOne ADF Container:
  - E1 ADF Container/build.xml
  - E1 ADF Container/build.properties
  - E1 ADF Container/UpdateURLConnections.xsl
  - E1 ADF Application/build.xml
  - E1 ADF Application/build.properites
- ADF 12.1.3 Container and ADF 12.2.1 Container. Both of these folders contain files for deploying ADF applications. The files that you use depend on the version of Oracle WebLogic Server (with ADF runtime) that you set up to deploy your ADF applications: Oracle WebLogic Server 12.1.3 or Oracle WebLogic Server 12.2.1. Both folders contain the following files:
  - JDEADFContainer.ear file. This file contains the EnterpriseOne ADF Container application.
  - E1UserSession.jar file. This file is not used for deployment, but contains helper classes required for developing EnterpriseOne ADF applications. If you plan to develop EnterpriseOne ADF applications, see *EnterpriseOne ADF Application Development Resources* for more information.

Download a Prebuilt EnterpriseOne ADF Application (Optional)

If you want to use a prebuilt EnterpriseOne ADF application, download the version of the prebuilt EnterpriseOne ADF application available for your JD Edwards EnterpriseOne Tools release. A prebuilt EnterpriseOne ADF application download contains ADF Library JAR file artifacts that are used to build and deploy the prebuilt application.

Building an EnterpriseOne ADF Application

Use the EnterpriseOne ADF application build script to build the EnterpriseOne ADF applications library. The following instructions describe how to execute the build script from the command line.

Type the following environment setup commands at the command line:

```
.(WLS_HOME)/server/bin/setWLSenv.sh
export PATH=$PATH:/anthome/bin
export ANT_PATH=/anthome
export JAVA_PATH=/jdk_home
```

Alternatively, if you are a developer and want to test building a new EnterpriseOne ADF application, you can execute the build script from within JDeveloper. See *Alternative Method for Manually Deploying EnterpriseOne ADF Applications* in this appendix for more information.

Set Up the Build Directories

Prior to executing the build script to generate the library WAR file, the build location needs to include the directory and files as described in this table.

Name	Type
E1 ADF Application/ADFLibraryJARs	The ADFLibraryJARs folder will contain the ADF library JAR files to include in the library WAR file. If this directory does not exist, you need to create it.
E1 ADF Application/ build.properties	PROPERTIES File
E1 ADF Application/ build.xml	XML Ant Script File

Define the Properties in the build.properties File

This table describes the properties in the build.properties file. These properties should be populated prior to running the script to deploy so that the EnterpriseOne ADF applications library manifest file is updated with the correct settings.

Ant Script Property	Description	Purpose
implementation.version	Identifies the implementation version of the library WAR file	build
specification.version	Identifies the specification version of the library WAR file	build
created.by	Identifies the entity that created the library WAR file	build
weblogic.server.host	The host on which the Administration Server is running.	deploy

Ant Script Property	Description	Purpose
weblogic.server.port	The port on which the Administration Sever is listening.	deploy
weblogic.deploy.targets	A comma-separated list of the target servers, clusters, or virtual hosts to which the JDE task flow library is deployed.	deploy
weblogic.debug	Enables wldeploy debugging messages.	deploy
weblogic.verbose	Specifies whether wldeploy displays verbose output messages.	deploy
weblogic.remote.deploy	Specifies if the server is located on a different machine. This affects how filenames are transmitted. The default value is true for this property.	deploy
weblogic.upload.war	Specifies if the JDETaskFlowLibrary.war file is copied to the Administration server's upload directory prior to deployment. The default value is true for this property.	deploy

Run the Build Script

After the properties are set in the build.properties file, you can execute the script at the command line using one of the following commands:

```
$ant
```

or

```
$ant build
```

These commands will execute the build target in the Ant script, which packages the bounded task flow library JAR files from the ADFLibraryJARs directory into a deployable library WAR file created in the deploy directory. You should receive a build successful message after executing.

You can proceed with the deployment in one of two ways: use the commands described in the next section to automatically deploy the JDETaskFlowLibrary.war to the specified Oracle WebLogic Server and/or clusters, or manually deploy the application as described in *Alternative Method for Manually Deploying EnterpriseOne ADF Applications*.

Deploying an EnterpriseOne ADF Application to Oracle WebLogic Server

Use the EnterpriseOne ADF build script to deploy the EnterpriseOne ADF artifacts. Prior to running the script:

- Make sure the build location includes the files as described in *Build Directory and Files for an EnterpriseOne ADF Application Build*.
- Make sure the deploy properties are defined in the build.properties file as described in *Properties in the build.properties File*.

> **Note:** The Ant script can also be used to immediately deploy the JDETaskFlowLibrary.war file to an Oracle WebLogic Server instance or cluster once the build process has completed.

After the properties are set in the build.properties file, you can execute the script at the command line in the directory that contains the build.xml script file using one of the following commands:

```
$ant deploy
```

or

```
$ant build-and-deploy
```

The first command executes the script's deploy target only and attempts to deploy the JDETaskFlowLibrary.war to the specified Oracle WebLogic Server and/or clusters. If the library war file does not exist, an error message is displayed, instructing you to run the build target first.

The second command launches the script's build target first, and then it launches the deploy target if the JDETaskFlowLibrary.war file was successfully created. You will be prompted to enter the Oracle WebLogic Server username and password during deployment. If successful, you will see a build successful message.

Configuring the EnterpriseOne ADF Container

Perform these tasks to build the EnterpriseOne ADF Container:

- *Set Up the Build Directories for the EnterpriseOne ADF Container Build*
- *Define the Properties in the EnterpriseOne ADF Container build.properties File*
- *Run the Build Script*

> **Note:** You must have completed the steps to build and deploy the JDETaskFlowLibrary.war file to the ADF server as outlined in *Set Up the Build Directories for the EnterpriseOne ADF Container Build* and *Deploying an EnterpriseOne ADF Application to Oracle WebLogic Server* in this appendix.

Set Up the Build Directories for the EnterpriseOne ADF Container Build

Before you execute the Ant script to update the JDEADFContainer.ear file, set up the following build directories:

Name	Type
AntBuildScripts/E1 ADF Container/ build.xml	XML Ant Script File
AntBuildScripts/E1 ADF Container/ build.properties	PROPERTIES File
AntBuildScripts/E1 ADF Container/ JDEADFContainer.ear	EAR File
AntBuildScripts/E1 ADF Container/ UpdateURLConnections.xsl	XSLT File

Define the Properties in the EnterpriseOne ADF Container build.properties File

Before deploying the ADF Container, use the EnterpriseOne ADF Container build script to update the e1adf.ini settings and connections.xml URL values. The build script uses the wldeploy task to deploy artifacts to server instances or clusters on Oracle WebLogic Server.

See *Building an EnterpriseOne ADF Application* and *Deploying an EnterpriseOne ADF Application to Oracle WebLogic Server* for more information.

Before you run the build script to update the JDEADFContainer.ear file, you have to define the properties in the build.properties file that are listed in the Ant Script Property column in *Set Up the Build Directories for the EnterpriseOne ADF Container Build*. When you run the build script, the script uses these values to update the associated settings in the e1adf.ini file. These required settings enable the communication between the EnterpriseOne ADF application and the EnterpriseOne HTML Server.

Additionally, you can assign values to the Ant properties described in *Table* to update the associated URL values in the connections.xml file. These settings are used to identify the map services used by ADF DVT map components. If these Ant properties are left blank, the EnterpriseOne ADF Container will use the default URL values delivered in the JDEADFContainer.ear file to access the below hosted services:

- Service: MapViewer

  URL: http://elocation.oracle.com/mapviewer
- Service: Geocoder

  URL: http://elocation.oracle.com/geocoder/gcserver

Note: You may want to install your own MapViewer instance, since the hosted MapViewer and Geocoder services by Oracle are intended for testing purposes only.

e1adf.ini Property	Ant Script Property	Description
aisServer	ais.server	The URL of the AIS Server used to populate data in the EnterpriseOne ADF applications. Usually this AIS Server is configured to point to the same HTML Server where Simplified UI is configured. For information on how to obtain this URL, see "Deploying and Managing the AIS Server through Server Manager" in the *JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* .
deviceName	device.name	The device name sent with every request to the AIS Server (AIS requires it).
jasWhitelist	jas.whitelist	Comma-delimited list of JAS URLs that are allowed to imbed this EnterpriseOne ADF Container when the Simplified User Interface setting for supporting ADF applications is configured for the EnterpriseOne HTML Server. The whitelist needs a comma-delimited list of server URLs for the EnterpriseOne HTML Server. The server URL should be of the form **<Protocol>://<IP Address/Fully Qualified Domain Name>:<Port>**. The IP Address/Fully Qualified Domain Name value depends on how the server has been set up and what URL the client (browser) sees as the server's origin. There is no penalty for an incorrect value being part of the whitelist, as long as the correct value is also present.
injectIframeBustingHeaders	inject.framebusting.headers	When set to true Content Security Policy (CSP) headers are included on every response from the ADF server, false indicates no CSP headers are added. The recommended setting is True.
sessionTimeout	session.timeout	The amount of time the EnterpriseOne ADF Container session stays alive while the user is not interacting with it, expected in milliseconds - recommended to be longer than EnterpriseOne HTML Server. Be sure to define the EnterpriseOne ADF Container session time out following this guideline: EnterpriseOne HTML server session time out < EnterpriseOne ADF Container session time out < AIS server session time out
sessionTimeToLive	session.timeToLive	The maximum amount of time a user's EnterpriseOne ADF Container session is allowed to be retained. Even if the user is continually interacting with the application,

e1adf.ini Property	Ant Script Property	Description
		the session will time out after this amount of time. The default value is 24 hours. The maximum value is 24 hours. Be sure to define the EnterpriseOne ADF Container session time to live following this guideline: EnterpriseOne ADF Container token time to live = AIS server session time to live(Release 9.2.0.5)
accessManagerEnabled	access.manager.enabled	Setting to enable/disable OAM use for ADF container access. Value should be true or false. (Release 9.2.0.2)
accessManagerServer	access.manager.server	The URL of the OHS gateway to access EnterpriseOne ADF applications in the container when OAM is enabled. Requests to ADF container via any other URLs will be blocked. If OAM is disabled, this value will be disregarded. (Release 9.2.0.2)

Connections.xml URLs and Ant script properties for E1 ADF Container

Connections.XML	Ant Script Property	Description
MapViewer URL	map.viewer.url	The URL of the Oracle Application Server MapViewer service. This service is required to run the base map that provides background for the layers in the ADF geographic map component. MapViewer is a programmable tool for rendering maps using spatial data managed by Oracle Spatial.
Geocoder URL	geocoder.url	The URL of the Geocoder service that converts street addresses into longitude and latitude coordinates for mapping.

Run the Build Script

You execute the script at the command line from the directory that contains the build.xml script file using either of the following commands:

$ant

or

$ant build

These commands execute the build target in the Ant script, which un-packages the JDEADFContainer.ear file, updates the e1adf.ini file, and repackages the .ear file in the following new "deploy" directory:

```
AntBuildScripts/E1 ADF Container/deploy
```

You should receive a build successful message after executing the command.

The JDEADFContainer.ear file in the deploy directory can then be manually deployed to an Oracle WebLogic Server instance or cluster, or follow the instructions in *Deploying an EnterpriseOne ADF Application to Oracle WebLogic Server* in this appendix.

Deploying the EnterpriseOne ADF Container

After the build process has completed, use the EnterpriseOne ADF Container build script to deploy the EnterpriseOne ADF Container to an Oracle WebLogic Server instance or cluster. Prior to running the script:

- Make sure the build location includes the files as described in *Properties in the build.properties File table*.
- Make sure the deploy properties are defined in the build.properties file as described in *Build Location Files table*.

Run the Build Script to Deploy the EnterpriseOne ADF Container

After the deploy properties are set in the build.properties file, use the build script to deploy the EnterpriseOne ADF Container. In a command line at the directory that contains the build.xml script file, use one of the following commands to execute the script:

$ant deploy

or

$ant build-and-deploy

The first command will execute the script's deploy target only and attempt to deploy the JDEADFContainer.ear file to the specified Oracle WebLogic Server or clusters. If the ear file does not exist, an error message displays instructing the user to run the build target first. The second command will first launch the script's build target and then the deploy target, if the JDEADFContainer.ear file was successfully created. The user will be prompted to enter the Oracle WebLogic Server username and password during deployment.

You should see a build successful message after executing the command. You can also log into the WebLogic Server Administration Console and view the deployed JDEADFContainer application under Deployments.

Alternative Method for Manually Deploying EnterpriseOne ADF Applications

If the build scripts do not work properly, you can manually deploy the EnterpriseOne ADF artifacts to Oracle WebLogic Server. The artifacts include a JDETaskFlowLibrary.war file which is used to deploy the EnterpriseOne ADF Application Library and a JDEADFContainer.ear file which is used to deploy the EnterpriseOne ADF Container application.

The section describes how to:

- *Deploy the EnterpriseOne ADF Application Library*
- *Deploy the EnterpriseOne ADF Container Application*

Deploy the EnterpriseOne ADF Application Library

Use the JDETaskFlowLibrary.war file to deploy the EnterpriseOne ADF application library. To do so:

1. Open the WebLogic Server Administration Console application.
2. In the Domain Structure area located on the left side of the screen, click Deployments.
3. In the Change Center area located on the left side, above the Deployments area, click the Lock & Edit button.
4. In the Summary of Deployments area, click the Install button. The Install Application Assistant displays.
5. In the Locate deployment to install and prepare for deployment area, click the upload your file(s) link.
6. In the Deployment Archive section, click the Choose File button.
7. Locate the JDETaskFlowLibrary.war file, select it, and then click the OK button.
8. Click the Next button.
9. Select the "Install this deployment as a library" option and click the Next button.
10. In the Available targets for oracle.jde.app.library section, select the ADF server(s) to which you want to deploy the JDETaskFlowLibrary.war file, and then click the Next button.
11. Click the Finish button without changing the name of the deployment.
    The JDETaskFlowLibrary.war file deploys to the ADF server.
12. After the JDETaskFlowLibrary.war file deploys, in the Change Center area located on the left of the screen, click the Activate Changes button.
13. Verify that final state of the library is Active and the type is Library.

Deploy the EnterpriseOne ADF Container Application

Use the JDEADFContainer.ear file to deploy the EnterpriseOne ADF Container application. To do so:

1. Open the WebLogic Server Administration Console application.
2. In the Domain Structure area located on the left side of the screen, click Deployments.
3. In the Change Center area located on the left side, above the Deployments area, click the Lock & Edit button.
4. In the Summary of Deployments area, click the Install button.
   The Install Application Assistant displays.
5. In the Locate deployment to install and prepare for deployment area, click the upload your file(s) link.
6. In the Deployment Archive section, click the Choose File button.
7. Locate the JDEADFContainer.ear file, select it, and then click the OK button.
8. Click the Next button.
9. Select the "Install this deployment as an application" option and click the Next button.
10. In the Available targets for JDEADFContainer section, select the ADF server to which you want to deploy the JDEADFContainer.ear file, and then click the Next button.
11. If needed, change the deployment name, so the current deployment can be uniquely identified in the deployments list. This is especially useful when deploying multiple JDEADFContainer instances to different ADF targets on the same WebLogic Server.
    Click the Finish button to complete the deployment.
    The JDEADFContainer.ear file deploys on the ADF server.

12. After the JDEADFContainer.ear file deploys, in the Change Center area located on the left of the screen, click the Activate Changes button.
13. In the Deployments list, select the JDEADFContainer instance you just deployed, and then select the Servicing All Requests option located in the Start drop-down menu.
14. The Start Application Assistant screen displays. Click the Yes button in the Start Deployments section to start the JDEADFContainer web application.
15. Verify that the final state of the JDEADFContainer web application is Active and the type is Enterprise Application.

Configuring EnterpriseOne HTML Server Settings

Server Manager contains EnterpriseOne HTML client settings that you need to configure based on your installation of the EnterpriseOne ADF Container. In Server Manager, complete the settings as described in the following sections.

Security Settings

Server Manager HTML Client Settings Security Category

- SimplifiedUXWhitelist

This option is a comma-delimited list of approved JD Edwards EnterpriseOne ADF containers.

The whitelist needs a comma-delimited list of server URLs for ADF. The server URL syntax should be:

```
<Protocol>://<IP Address/Fully Qualified Domain Name>:<Port>.
```

The IP Address/Fully Qualified Domain Name value depends on how the server has been set up and what URL the client (browser) sees as the server's origin. There is no penalty for an incorrect value being part of the whitelist as long as the correct value is also present.

Runtime Settings

Server Manager HTML Client Settings Web Runtime Category

These settings apply to the ADF server where the EnterpriseOne ADF Container has been installed.

- SimplifiedUXProtocol

  Designate the Simplified UX Framework's host server protocol (http or https).
- SimplifiedUXHost

  Designate the Simplified UX Framework's host server name.
- SimplifiedUXPort

  Designate the Simplified UX Framework's host server's port name (an integer).
- SimplifiedUXContextRoot

  Designate the context root of Simplified UX Framework.

Use the default setting in Server Manager because it matches the setting provided during the EnterpriseOne ADF Container build and deploy.

- SimplifiedUXHandShake

  Designate the handshake name of Simplified UX Framework.

  Use the default setting in Server Manager because it matches the setting provided during the EnterpriseOne ADF Container build and deploy.

- SimplifiedUXWebApp

  Designate the web app name of Simplified UX Framework.

  Use the default setting in Server Manager in order to match the setting updated during the EnterpriseOne ADF Container build and deploy.

Timeout Settings

The AIS Server maintains sessions for EnterpriseOne ADF applications. You can configure the session timeouts for the AIS Server through Server Manager following this guideline:

[CACHE]

UserSession

Be sure to define the UserSession timeout following this guideline:

The EnterpriseOne HTML server session timeout should be less than the EnterpriseOne ADF Container session timeout.

And the EnterpriseOne ADF Container session timeout should be less than the AIS server session timeout.

Note: Make sure that you complete all three of the above sections.

Considerations for an Oracle Access Management (OAM) Configuration when Deploying the EnterpriseOne ADF Container (Release 9.2.0.2)

When you use OAM with an EnterpriseOne HTML Server and OAM with an ADF Server, you will notice that both servers communicate through same domain. In this type of situation, you are likely to have session management issues. If as soon as you log in you receive the error message "Your session has expired", you are probably encountering a session management issue.

To resolve this type of a session management issue, you need to modify the weblogic.xml files in the WEB-INF folder in EnterpriseOne HTML Server and ADF Server deployments.

OAM with JAS Cookie Configuration

Open the Oracle Access Management Console for JAS:

```
<weblogic-web-app xmlns:xsi="http://yourserver" xsi:schemaLocation="http://yourserver"
 xmlns="http://yourserver">
   <session-descriptor>
      <cookie-path>/jde</cookie-path>
      <cookie-http-only> true</cookie-http-only>
   </session-descriptor>
</weblogic-web-app>
```

If you already deployed the EnterpriseOne HTML Server before making these changes, you need to restart the server.

OAM with ADF Cookie Configuration

If you are using one of the latest EnterpriseOne ADF Container versions, you should already have this part included in the code. In that case, you do not need to do anything for the EnterpriseOne ADF Container.

Open the Oracle Access Management Console for ADF:

```
<weblogic-web-app xmlns:xsi="http://yourserver" xsi:schemaLocation="http://yourserver"
 xmlns="http://yourserver">
   <session-descriptor>
      <cookie-path>/jdeadf</cookie-path>
      <cookie-http-only> true</cookie-http-only>
   </session-descriptor>
</weblogic-web-app>
```

If you already deployed the ADF Server before making these changes, you need to restart the server.

OAM with AIS Cooking Configuration

Because the AIS Server has the same cookie as the EnterpriseOne HTML Server and ADF Server and it is not accessed by the browser, the system should not have any conflicts with the EnterpriseOne HTML Server and ADF Server. However, users are recommended to have the same configuration on the AIS Server as well.

Open the Oracle Access Management Console for AIS Server:

```
<weblogic-web-app xmlns:xsi="http://yourserver" xsi:schemaLocation="http://yourserver"
 xmlns="http://yourserver">
   <session-descriptor>
      <cookie-path>/jderest</cookie-path>
      <cookie-http-only> true</cookie-http-only>
   </session-descriptor>
   <context-root>
       jderest
   </context-root>
</weblogic-web-app>
```

If you already deployed the AIS Server before making these changes, you need to restart the server.

Configuring ADF Settings Using Server Manager

In Server Manger, configure the following ADF Server settings, which are located in the Advanced settings, under OAM CONFIG:

- **Oracle Access Manager Enabled**

  Select this check box to let a direct access request to ADF content through.

- **accessManagerServer**

  The EnterpriseOne ADF Container uses this value to decide which requests to let through when the Oracle Access Manager Enabled check box is selected. This value holds the Oracle HTTP Server base URL, which is used by users to access ADF content, for example http://*yourserver:port*

  > **Note:**
  >
  > - *"Install an Oracle Application Development Framework Server Instance" in the JD Edwards EnterpriseOne Tools Server Manager Guide* .

15 Troubleshooting

Build Scripts Additional Information

There are two Ant scripts required to create, update, and deploy ADF libraries and the JDEADFContainer application on Oracle WebLogic Server. Both Ant scripts use the wldeploy task to deploy artifacts to server instances or clusters on Oracle WebLogic Server. Both of the Ant scripts provide these additional targets:

Ant Script Target	Description
clean	Removes the build and deploy directories.
usage	Displays a list of command line options that will provide additional information about Ant or the build scripts.
undeploy	Removes the JDEADFContainer.ear or the JDETaskFlowLibrary.war from Oracle WebLogic Server, depending on which ant script is executed.

Error Messages and Their Meanings

This section describes error messages you may see and provides an explanation of what they mean:

- Error: System issues prevent the application from opening. Please contact your system administrator. The EnterpriseOne HTML server Jas log would contain additional information:

  "The external application launch has failed. The handshake id is null or never completed its initialization. Verify that your system is running and its ini settings are properly configured." As the message notes, the handshake ID is either null or blank. Potential reasons for this situation could be the. jasWhitelist on the ADF Server is not configured correctly or the AIS Server or ADF Server is down.
- Error: Invalid application data. Please contact your system administrator. Ensure that the proxy application associated with the EnterpriseOne ADF application is properly defined and secured. Users could see this error message if the EnterpriseOne task containing the EnterpriseOne ADF application lacks a proxy app and/or task flow path/url. See the *JD Edwards EnterpriseOne Tools Solution Explorer Guide* for additional details on this EnterpriseOne ADF application task.

ADF Charts

The ADF charts continue trying to load when there is not a monitor associated with the server. When this happens, the following error appears in the Oracle WebLogic Server Diagnostic logs:

Can't connect to X11 window server using '10.111.111.150:0.0' as the value of the DISPLAY variable.

To fix this circumstance, configure the server instances with the following Java property set to "true":

```
-Djava.awt.headless=true
```

Setting this property to "true" prevents the Java runtime from attempting to find a monitor to get the DISPLAY variable.

To configure this property in WebLogic Server:

1. On the Server Start tab, add `-Djava.awt.headless=true` in the Arguments field.
2. Click **Save**.
3. Restart the managed instance.

Host Verification Errors, or Failure to Open an EnterpriseOne ADF Application

If you have configured a reverse proxy and enabled Secure Sockets Layer (SSL), ensure that the Host Verification setting is set to None. You can locate this setting on your Web Logic Server, on the SSL tab, in the Advanced section.

16 Appendix E - Extending ADF Applications (Release 9.2.1.1)

Before You Begin

Before using the downloaded ADF application JDeveloper projects, perform the following tasks:

- Review the ADF application development processes found in *EnterpriseOne ADF Application Development Resources*of this guide.
- Download the JD Edwards EnterpriseOne ADF application (.zip) files from the JD Edwards Update Center on My Oracle Support: `https://updatecenter.oracle.com/`
- Ensure that you have the correct version of JDeveloper (12.1.3 or 12.2.1) installed.

> **Note:** JD Edwards EnterpriseOne ADF applications were developed using JDeveloper versions 12.1.3 or 12.2.1. If the application was developed with JDeveloper 12.1.3, then both 12.1.3 and 12.2.1 source .zip files are provided on the JD Edwards Update Center. If the application was developed in 12.2.1, then this is the only source provided. Source written in JDeveloper 12.2.1 cannot be migrated backward to version 12.1.3.

Understanding EnterpriseOne ADF Applications

JD Edwards EnterpriseOne ADF applications are built using JDeveloper. An EnterpriseOne ADF application archive is a specially built JDeveloper project that enables developers to take an existing EnterpriseOne ADF application, customize it to suit their particular business needs, and generate their own .jar file.

Modifying ADF Source Code

The ADF application source is delivered as part of the download from the Update Center. The download contains a .zip file, in which you will find the application source project. See *Creating a Sample EnterpriseOne ADF Application* for references when making any specific, necessary changes to the source code.

In order for the project to compile, you need to attach five .jar files that are provided by Oracle. These steps are referenced in step 7 of *Getting Started* and in *Creating a New EnterpriseOne ADF Application*. The five .jar files are located in the AIS_Client_Java_API_*XXX* and the E1_ADF_Foundation_*XXX* downloads in the Update Center. You should remove any older references to these .jar files before attaching the new ones.

ADF Server Deployed Using Server Manager (Release 9.2.1.2)

If you are deploying the ADF Server using Server Manager and plan to include custom ADF applications on your ADF Server, you need to bundle the scf-manifest.xml file with your custom ADF application so that Server Manager

recognizes it as an ADF application component, which can be used for the creation of an ADF Server Instance. If a valid scf-manifest.xml file is available in the ADF application component JAR file, the provided version, description, and componentName appear in the Server Manager Console.

You can use the existing scf-manifest.xml files as a reference for creating the scf-manifest.xml file for your ADF application. Here is an example of the scf-manifest.xml file from the EnterpriseOne ADF Visual Bill of Material application:

```
<scf-manifest
      targetType="adfAppComponent"
     version="92"
     componentName="E1_ADF_VisualBillOfMaterial_92"
     description="E1 ADF VisualBillOfMaterial v92"
     minimumAgentLevel="131">
</scf-manifest>
```

After acquiring the scf-manifest.xml file from the Update Center, modify the file to identify your application in Server Manager. Update the values for the following fields:

- **version** - Corresponds to the version shown in the Server Manager Console.
- **componentName** - Provides a descriptive name of your application in the Server Manager Console.
- **description** - Provides additional details about the application.

The scf-manifest.xml file should reside in your view controller project path within the \adfmsrc\META-INF\ folder. When you build a new .jar file, this file will get bundled.

Generating Your New ADF Application

After completing your modifications, follow the steps in *Creating ADF Library JAR File* to generate a .jar file from your source code. The new file will then be used instead of the one provided in the download from the Update Center.