JD Edwards EnterpriseOne Tools

Developer's Guide for EnterpriseOne Application Development Framework (ADF) Applications

Tools Release 9.1.x

E60057-02

September 2015

Describes the Oracle Application Developer Framework (ADF) and discusses the JD Edwards EnterpriseOne-specific tools and environment required to support the development of ADF enterprise applications for EnterpriseOne.

ORACLE

JD Edwards EnterpriseOne Tools Developer's Guide for EnterpriseOne Application Development Framework (ADF) Applications, Tools Release 9.1.x

E60057-02

Contents

List of Examples

Preface

Welcome to the *Developer's Guide for EnterpriseOne Application Development Framework (ADF) Applications*.

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related 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
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understanding This Guide

Use this guide as a companion guide to the *Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*, which describes how to develop web applications using Oracle Application Development Framework (ADF). Oracle ADF is the framework used to create web applications for JD Edwards EnterpriseOne. You can access the Oracle ADF documentation here:

`http://docs.oracle.com/middleware/1213/adf/docs.htm`

The JD Edwards EnterpriseOne Tools Developing and Customizing EnterpriseOne Simplified User Applications Guide includes:

- An overview of the environment required for developing EnterpriseOne ADF enterprise applications, which includes Oracle ADF and JD Edwards EnterpriseOne-specific extensions and tools, referred to as JDE ADF Helpers.

  See Chapter 2, "Introduction to Simplified Application Framework Development"for more information.

- How to use the JDE ADF Helpers with Oracle ADF to create custom EnterpriseOne ADF applications.

  See the following chapters for more information:

  - Chapter 3, "Getting Started"
  - Chapter 4, "Connecting to AIS"
  - Chapter 5, "Building the Data Model"
  - Chapter 6, "Executing AIS Calls for Retrieving Data"
  - Chapter 7, "EnterpriseOne ADF Container"

- Step-by-step instructions on how to create a sample application, including how to add features with JDE ADF Helpers.

  See Appendix A, "Creating a Sample ADF Application."

2

Introduction to Simplified Application Framework Development

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

This chapter contains the following topics:

2.1 About the Runtime Architecture for EnterpriseOne ADF Enterprise Applications

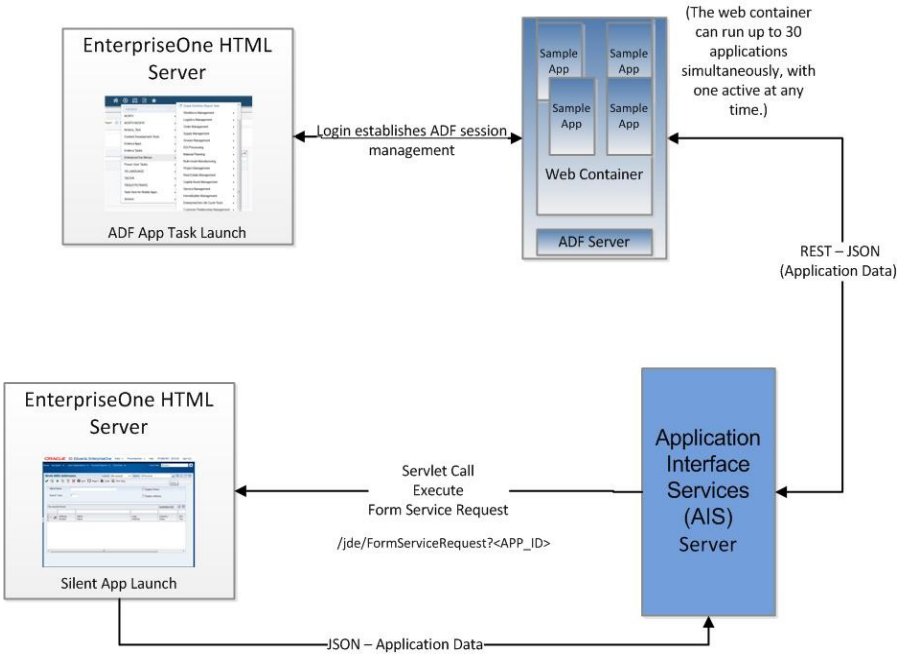
EnterpriseOne ADF applications expose a light interface to manage EnterpriseOne data. The Application Interface Services (AIS) Server provides a JSON over REST interface to EnterpriseOne applications and forms through the EnterpriseOne HTML Server. The AIS Server exposes this interface to enable communication between ADF applications and EnterpriseOne. The data displayed in an ADF application is retrieved by making calls to AIS.

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

The AIS Server maintains sessions when ADF applications make requests. You can configure the session timeouts for the AIS Server through Server Manager.

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

Figure 2–1 Runtime Architecture for EnterpriseOne ADF Enterprise Applications



The following list describes the above graphic:

1. When user logs into EnterpriseOne, a connection is established between the AIS, JAS, and ADF Servers, identified by a handshake id.
2. When an ADF application task is launched, JAS sends a request to the JDEADFContainer web app to start the ADF application, passing the handshake id.
3. The ADF application uses the handshake id to access the AIS token created for the current session and executes AIS service calls.
4. AIS validates the token for each request, forwards the requests to JAS to fulfill, and returns the responses to the ADF application.

2.2 About Oracle ADF

Oracle ADF is a solution that enables you to create web applications that run using JD Edwards data. EnterpriseOne web enterprise applications are built using Oracle ADF.

You should gain a thorough understanding of Oracle ADF before reading further. See the *Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

2.3 Developing Custom Simplified Applications with ADF for EnterpriseOne

If you have customized EnterpriseOne applications to meet your specific business requirements, you can extend the functionality of the ADF applications to interact with the customized EnterpriseOne applications. ADF enables you to retrieve data from custom EnterpriseOne applications by using existing business logic within your EnterpriseOne applications.

Before you develop custom ADF applications, you should plan all aspects of the process. The "Introduction to Building Fusion Web Applications with Oracle ADF" chapter in the *Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* describes the architecture and key functionality of Oracle Application Development Framework (Oracle ADF) when used to build a web application that uses ADF Business Components, ADF Model, ADF Controller, and ADF Faces rich client, along with high-level development practices.

EnterpriseOne is utilizing ADF Task Flows to support Simplified EnterpriseOne applications.

These activities include gathering requirements, designing, developing, deploying, testing and debugging, securing, enabling access to the server-side data, redeploying, retesting and debugging, and publishing. As you perform these activities, make sure to consider the following tasks when building EnterpriseOne ADF applications with ADF:

- **Designing**. Determine which forms in EnterpriseOne the ADF application will access data from. If there are multiple forms, consider creating a batch request to gather data from multiple forms.
- **Developing**. Use the JD Edwards EnterpriseOne AIS Client API, AIS Client Class, and JDE ADF Helpers to help expedite the development of your ADF application. Refer to Appendix A, "Creating a Sample ADF Application" in this guide for an example of how to develop a custom ADF application for EnterpriseOne using these utilities.

2.3.1 JDE ADF Helpers

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

- AIS_Client.jar

  The AIS_Client.jar contains the AIS Client Java API, which provides Java methods to support communication with the JD Edwards EnterpriseOne Application Interface Services (AIS) Server.

- 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 E1 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's required for all AIS service requests within the ADF application.

2.3.2 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 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.

2.3.3 Form Service Requests (FSR)

AIS Server calls are used to retrieve data from forms in the EnterpriseOne web client. These calls are referred to as form service requests. 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.

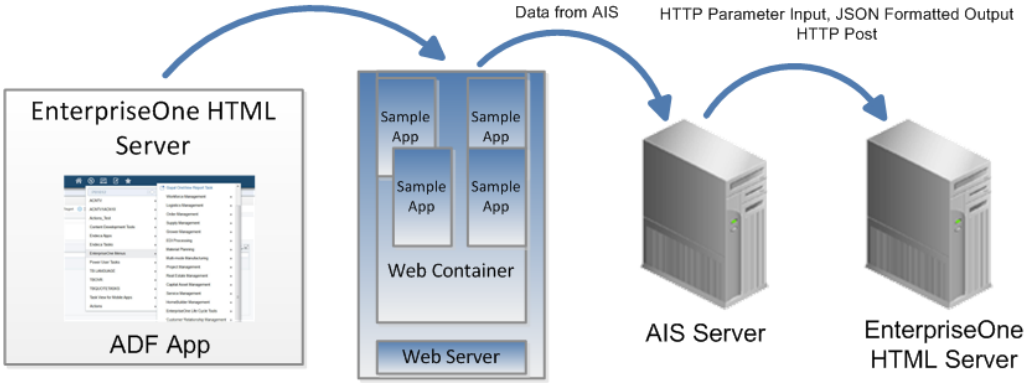
By sending an ordered list of commands, a form service request can replicate the actions taken by an EnterpriseOne web client user, including populating fields, pressing buttons, and other actions. The form service request enables you to perform various operations on a single form. The URL for the form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

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

2.3.4 EnterpriseOne Rest Services Interface

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

Figure 2–2 AIS Client and Server Communication



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

You can make REST calls directly to the AIS Server without using the JDE ADF Framework APIs. Use a REST service testing tool to call AIS services directly by sending JSON text to the URL for the service. All services are accessed using URLs with this format: `http://<aisserver>:port/jderest/<uri>`, where uri is the path to the various types of services such as formservice, file, defaultconfig, and poservice.

2.4 Sample Application

This guide includes an appendix that provides 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 this guide, you can refer to the steps in this appendix to see an example of how the features are used in the development of an ADF application.

3

Getting Started

This chapter refers to certifications (MTRs) and prerequisites for developing and running EnterpriseOne ADF enterprise applications. It also includes information about the following topics:

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

3.1 Certifications (Formerly Known as Minimum Technical Requirements)

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

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

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

3.2 Prerequisites

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

- You have basic knowledge for creating and testing web applications using JDeveloper.
- You must be running a minimum of EnterpriseOne Tools release 9.1.5.3.
- To run the EnterpriseOne ADF Container (JDEADFContainer), the ADF server must have WebLogic 12.1.3 with ADF libraries deployed.
- Complete the prerequisites listed in the Oracle ADF guide, which include:
  - Oracle JDeveloper 12.1.3

    See "Prerequisites for Developing ADF Applications" in the *Oracle Fusion Middleware Developing ADF Applications with Oracle ADF Application Framework Guide*.
- Download the Application Interface Services Client Java API 1.2.1 from the **JD Edwards Update Center** on My Oracle Support https://support.oracle.com/

To locate the download on the **JD Edwards Update Center**, use the Type field to search on "EnterpriseOne ADF." The download is available in a package titled AIS_Client_Java_API_1.2.1.

The download contains the following files:

- AIS Client JARS (Required for developing EnterpriseOne ADF applications)
  - AIS_Client.jar
  - jackson-annotations-2.2.4.jar
  - jackson-core-2.2.4.jar
  - jackson-databind-2.2.4.jar
- AISCGE 12c_v1.6.2.zip (AIS Client Class Generator extension for JDeveloper)

- Download the EnterpriseOne ADF Foundation 1.1.0 from JD Edwards Update Center on My Oracle Support (`https://support.oracle.com/`).

To locate the download on the **JD Edwards Update Center**, use the Type field to search on EntepriseOne ADF. The download is available in a package titled E1_ADF_Foundation_1.1.0. The download contains the following files:

- Ant Build Scripts (Ant build/deploy scripts for EnterpriseOne ADF applications and JDEADFContainer.ear).
- JDEADFContainer.ear (EnterpriseOne ADF Container web application)
- E1UserSession.jar (Helper classes required for developing EnterpriseOne ADF applications)

- Configure the EnterpriseOne HTML(JAS) Server and JDEADFContainer web application to enable communication between the AIS, JAS, and ADF Servers.

3.3 Installing the AIS Client Class Generator Extension for JDeveloper

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

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

To install the extension:

1. In JDeveloper, select the **Help** menu, **Check for Updates**.
2. Select **Install From Local File**, and then enter the location of the zip file.
3. Select **Install From Local File** , and then enter the location of the AISCGE 12c zip file.

   JDeveloper closes automatically.

4

Connecting to AIS

This chapter describes the method used by ADF applications to connect to the AIS Server and contains the following topics:

4.1 Creating a Connection to AIS Server

An ADF application's bounded task flow includes handshakeId as one of its four required input parameters. The application data control uses this handshakeId parameter to determine whether the ADF application is running locally on JDeveloper's Integrated WebLogic Server (WLS) or executing inside the JDEADFContainer.

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

Example 4–1 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 ADF application is running locally, the handshakeId input parameter will be null. As a result, the 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 will be 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 WLS. Before deploying the 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 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 JAS, AIS, and JDEADFContainer.

5 Building the Data Model

This chapter contains the following topics:

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

5.1 Understanding the Data Model

The data model for EnterpriseOne ADF applications includes foundation classes that hold the data returned from AIS services. These classes are specific to the EnterpriseOne application forms accessed by an ADF application's data control. You use the AIS Client Class Generator to generate these classes in JDeveloper and typically add them to the Model project along with the data control class.

5.2 Configuring the AIS Client Class Generator

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

To configure the AIS Client Class Generator:

1. In JDeveloper, access Preferences:

   On Microsoft Windows, select the **Tools** menu, **Preferences**.

   On Mac, select the **JDeveloper** menu, **Preferences**.

2. Select **AIS Client Class Generator**.
3. Press the **Load Extension** button, if visible, to display the AIS Client Class Generator form.
4. On Preferences, complete the following fields to specify the AIS Server location and AIS Server information:
   - **AIS Server URL**. This is a fully qualified URL to the AIS Server. Make sure it ends with the port number of the AIS Server.
   - **JAS Server URL**. (Optional) Only enter a value if you want to override the value configured on the AIS Server.
   - **Username**. Enter a JD Edwards EnterpriseOne user name.

- **Password**. Enter a JD Edwards EnterpriseOne user password.
- **Environment**. (Optional) Enter a value only if you want to override the value configured on the AIS Server.
- **Role**. (Optional) Enter a value only if you want to override the value configured on the AIS Server.
- **JSON Files Folder**. The directory where JSON data files are saved, if you elect to retain the generated JSON.
- **Default Java Classes Folder**. The directory where the generated form classes are stored, if an active project is not selected.
- **Java Package**. Enter the name of the Java package assigned to the generated form classes. This will also determine the folder structure for the Java classes in the project src folder.

5. Click **OK**.

5.3 Generating Data Classes Based on a Form

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

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

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

1. In JDeveloper, select the **Model** project in your ADF application workspace.

   JDeveloper will save the classes generated by the AIS Client Class Generator in this location.

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

- **FormServiceAction**. Enter the action to be performed. Valid values include: Create, Read, Update, Delete.
- **FindOnEntry**. (Optional)
- **DemoMode**. (Optional, but recommended) This ensures at least one grid row is present, so grid classes are generated even if there is no data in the database.
- **Generate for Mobile Application**. Ensure that this check box is not checked

4. Select the **Preview JSON Data** and **Keep JSON Files** check boxes, if you want to preview and keep the generated JSON files.
5. Click the **Generate** button to generate the JSON, and then verify that it has the fields and records you need.
6. Click **Continue** to generate the Java files.

   If successful, a confirmation message appears that shows the locations of the JSON files and the form service classes.
7. Highlight the Model project and then click the **refresh** button to display the new files.

6

Executing AIS Calls for Retrieving Data

This chapter contains the following topics:

6.1 Understanding JD Edwards EnterpriseOne ADF Framework Container

The JD Edwards EnterpriseOne ADF Container (JDEADFContainer) is a web application that executes ADF applications created as bounded task flows and packaged in ADF Library JAR files. These EnterpriseOne ADF applications must use the AIS Client API to perform REST services on the AIS Server to interact with EnterpriseOne. The AIS Client 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 the *JD Edwards EnterpriseOne Application Interface Services (AIS) Client API Reference*

6.2 Understanding AIS Server Capabilities

The AIS Server exposes various capabilities on which client applications may or may not depend. If your 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 AIS Server capabilities using the following URL:

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

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

Example 6–1 Capability List

Each AIS capability used by your 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.

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

```
            {
         "name": "thumbnailSize",
         "shortDescription": "Specify desired thumbnail size for MO List",
         "longDescription": "Ability to request a specific sized thumbnail images
in a Media Object List Request",
         "asOfRelease": "9.1.5"
      },
            {
         "name": "gridCellClick",
         "shortDescription": "Click Grid Cell Hyperlink",
         "longDescription": "Ability to use GridCellClick event, to execute
hyperlink in grid",
         "asOfRelease": "9.1.5.2"
      },
            {
         "name": "query",
         "shortDescription": "Query",
         "longDescription": "Ability to use Query on forms that support it",
         "asOfRelease": "9.1.5.2"
      },
            {
         "name": "taskAuthorization",
         "shortDescription": "Task Authorization",
         "longDescription": "Ability to receive a list of authorized tasks based
on a task view id, or task id and parent id with in a task view",
         "asOfRelease": "9.1.5.2"
      },
            {
         "name": "urlMediaObjects",
         "shortDescription": "URL Media Objects",
         "longDescription": "Ability to view, add or delete url type media
objects",
         "asOfRelease": "9.1.5.2"
      },
            {
         "name": "jargon",
         "shortDescription": "Data Item Jargon Service",
         "longDescription": "Ability to request data item descriptions based on
users language and jargon (system) code",
         "asOfRelease": "9.1.5.3"
      },
            {
         "name": "aliasNaming",
         "shortDescription": "Alias Naming",
         "longDescription": "Ability receive form service responses with fields
named by Data Dictionary alias ",
         "asOfRelease": "9.1.5.3"
      }
   ]
```

Example 6–2

```
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. The below ADF example shows both of these methods.

Example 6–3

```
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);
    }
}
```

6.3 Understanding Form Service Requests (FSRs)

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

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

To send an FSR to the AIS Server, send a POST to the following URL and send JSON in the body:

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

If testing with a REST testing tool, you can send JSON directly. If using the JD Edwards ADF helpers in an ADF application, you must specify only the URI when calling the jdeRestServiceCall. The URI is "formservice" and you can use the static variable, for example:

```
JDERestServiceProvider.FORM_SERVICE_URI
```

6.3.1 Events Used in an FSR

The following table lists the events that you can include in an FSR, and describes the action that each event performs.

Event	Description	Parameters
Set Control value	Sets the value of a control on a form, like filter fields or any other form control.	controlID ("25") value ("Bob" or "01/01/2015")
Set QBE Value	Sets the value of a QBE column.	controlID ("1[42]" or "1_2[25]") value ("Jill" or "55")
Set Checkbox Value	Sets the value of a check box.	controlID ("77") value ("on" or "off")
Set Radio Button	Sets the value of the radio button.	controllID ("87") value ("87")
Set Combo Value	Sets the value of a combo box entry.	contolID ("125") value (2) - Index of the entry.
Do Action	Presses a button or Hyper Item.	controlID ("156")
Select Row	Selects the specified row in a grid.	controlID ("1.30") - ID of the grid dot then row index (zero based).

Event	Description	Parameters
Select All Rows	Select all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - ID of the grid.
Un Select All Rows	Unselects all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - ID of the grid.
Un Select Row	Unselects the specified row in a grid.	controlID ("1.30") - ID of the grid dot then row index (zero based).
Click Grid Cell	Click the hyperlink in a grid cell.	controlID("1.5.22") - ID of the grid dot then the row index dot column ID.

The last three events listed in the table are available only with the selectAllGridRows capability which is available starting with EnterpriseOne Tools 9.1 Update 5 release. The AIS Client API requires these events to be in a try block because they throw a Capability Exception.

6.3.2 Using the Form Service Request Event in Form Design Aid (FDA)

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

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

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

6.3.3 Placing Events in the Proper Order

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

6.3.4 Considering Hidden Filters and Hidden QBE

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

6.3.5 Example of Events on Power Forms

The Form Service Request event and other events that perform actions in a form are also available on power forms so that you can populate a subform value or press a button on a subform. Indicate a subform by adding an extra prefix on the control, for example:

```
3_1[25]
```

In this example, 3 is the subform ID, 1 is the subform grid, and 25 is the grid column ID.

Another example is `3_43`, in which 3 is the subform ID and 43 is the field ID. Getting a JSON representation of the form should help you find the exact IDs that you need.

6.3.6 Example of JSON Code in a FSR

The sample code in Example 6–4 is an example of JSON code in a FSR that executes EnterpriseOne actions in the following order:

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

Example 6–4 Executing EnterpriseOne Actions - JSON Request

```
 {
    "token":
"044BlLYkCUcjQGRxvR3r+LH27liC6l6psFHOTp9whmkPxE=MDE4MDA2MTYxMDIwOTQ1MDU2NTc0NDY0U2
9hcFVJMTM4NDQ0NjU2NTUwNQ==",
    "version": "ZJDE0001",
    "formActions": [
        {

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

Example 6–5 Executing EnterpriseOne Actions - JSON Response


```
{
  "fs_P01012_W01012B" : {
    "title" : "Work With Addresses",
    "data" : {
      "z_AT1_53" : {
        "id" : 53,
        "title" : "Search Type",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblSearchType_53",
        "value" : "Search Type",
        "editable" : false
      },
      "z_AT1_54" : {
        "id" : 54,
        "title" : "Search Type",
        "dataType" : 2,
        "staticText" : "Search Type",
        "visible" : true,
        "bsvw" : true,
        "longName" : "txtSearchType_54",
        "value" : "*",
        "editable" : true,
        "assocDesc" : ""
      },
      "z_ALPH_57" : {
        "id" : 57,
        "title" : "Alpha Name",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblAlphaName_57",
        "value" : "Alpha Name",
        "editable" : false
      },
      "z_ALPH_58" : {
        "id" : 58,
        "internalValue" : "Allen*",
        "title" : "Alpha Name",
        "dataType" : 2,
        "staticText" : "Alpha Name",
        "visible" : true,
        "bsvw" : false,
        "longName" : "txtAlphaName_58",
        "value" : "Allen*",
        "editable" : true
      },
      "z_DL01_66" : {
        "id" : 66,
        "title" : "",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblDL01_66",
        "value" : "",
        "editable" : false
      },
      "z_EV01_63" : {
```

```
  "id" : 63,
  "internalValue" : "1",
  "title" : "Display Address",
  "dataType" : 1,
  "visible" : true,
  "bsvw" : false,
  "longName" : "chkDisplayAddress_63",
  "value" : "on",
  "editable" : true
},
"z_EV01_62" : {
  "id" : 62,
  "internalValue" : "1",
  "title" : "Display Phone",
  "dataType" : 1,
  "visible" : true,
  "bsvw" : false,
  "longName" : "chkDisplayPhone_62",
  "value" : "on",
  "editable" : true
},
"gridData" : {
  "titles" : {
    "col_19" : "Address Number",
    "col_20" : "Alpha Name",
    "col_40" : "Address Line 1",
    "col_44" : "City",
    "col_81" : "Prefix",
    "col_46" : "Phone Number",
    "col_47" : "Phone Type",
    "col_48" : "Long Address",
    "col_49" : "Industry Class",
    "col_50" : "Sch Typ",
    "col_51" : "Tax ID"
  },
  "rowset" : [ {
    "rowIndex" : 0,
    "MOExist" : true,
    "z_AN8_19" : {
      "id" : 19,
      "internalValue" : 6001,
      "title" : "Address Number",
      "dataType" : 9,
      "visible" : true,
      "mathValue" : {
        "currencyDecimals" : 0,
        "zero" : false,
        "negative" : false,
        "intValue" : 6001,
        "decimalPosition" : 0
      },
      "bsvw" : true,
      "longName" : "mnAddressNumber_19",
      "value" : "6001",
      "editable" : false
    },
    "z_ALPH_20" : {
      "id" : 20,
      "internalValue" : "Allen, Ray",
      "title" : "Alpha Name",
```

```
    "dataType" : 2,
    "visible" : true,
    "bsvw" : true,
    "longName" : "sAlphaName_20",
    "value" : "Allen, Ray",
    "editable" : false
  },
  "z_ADD1_40" : {
    "id" : 40,
    "internalValue" : "410 17th Avenue",
    "title" : "Address Line 1",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "sAddressLine1_40",
    "value" : "410 17th Avenue",
    "editable" : false
  },
  "z_CTY1_44" : {
    "id" : 44,
    "internalValue" : "Denver",
    "title" : "City",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "sCity_44",
    "value" : "Denver",
    "editable" : false
  },
  "z_AR1_81" : {
    "id" : 81,
    "internalValue" : "",
    "title" : "Prefix",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "sPrefix_81",
    "value" : "",
    "editable" : false
  },
  "z_PH3_46" : {
    "id" : 46,
    "internalValue" : "",
    "title" : "Phone Number",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "sPhoneNumber_46",
    "value" : "",
    "editable" : false
  },
  "z_PHTP_47" : {
    "id" : 47,
    "internalValue" : "",
    "title" : "Phone Type",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "sPhoneType_47",
    "value" : "",
```

```
          "editable" : false
        },
        "z_ALKY_48" : {
          "id" : 48,
          "internalValue" : " ",
          "title" : "Long Address",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sLongAddress_48",
          "value" : " ",
          "editable" : false
        },
        "z_SIC_49" : {
          "id" : 49,
          "internalValue" : " ",
          "title" : "Industry Class",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sIndustryClass_49",
          "value" : " ",
          "editable" : false
        },
        "z_AT1_50" : {
          "id" : 50,
          "internalValue" : "E",
          "title" : "Sch Typ",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sSchTyp_50",
          "value" : "E",
          "editable" : false
        },
        "z_TAX_51" : {
          "id" : 51,
          "internalValue" : "798525841",
          "title" : "Tax ID",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sTaxID_51",
          "value" : "798525841",
          "editable" : false
        }
      } ],
      "summary" : {
        "records" : 1,
        "moreRecords" : false
      }
    }
  },
  "errors" : [ ],
  "warnings" : [ ]
},
"stackId" : 1,
"stateId" : 1,
"rid" : "785dbb271b6068a6",
"currentApp" : "P01012_W01012B_ZJDE0001",
```

```
  "sysErrors" : [ ]
}
```

6.3.7 Example of API Methods for an FSR

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

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

Example 6–6 API Methods for Setting Commands

```
        FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P01012_W01012B");
        formRequest.setVersion("ZJDE0001");
        formRequest.setFormServiceAction(FormRequest.ACTION_READ);

        // Create event holder and add actions in order
        FSREvent formEvents = new FSREvent();
        formEvents.setQBEValue("1[50]", searchType);
        formEvents.setFieldValue("58", name);
        formEvents.checkBoxChecked("62");
        formEvents.checkBoxChecked("63");
        formEvents.doControlAction("15");

        // Add event holder to the form request
        formRequest.addFSREvent(formEvents);

        // Execute form service request and process response
        try
        {
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
formRequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_
SERVICE_URI);

            P01012_W01012B_FormParent newFormParent =
loginEnv.getObjectMapper().readValue(response, P01012_W01012B_FormParent.class);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
```

6.3.8 Grid Action Events

In addition to interacting with fields on the form, you can interact with grids using grid action events. If you use a grid action event, you must identify "grid" as a required capability before executing the form request.

See Section 6.2, "Understanding AIS Server Capabilities" for more information.

The types of grid action events include:

- Selecting grid rows

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

- Inserting grid rows

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

- Updating grid rows

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

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

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

6.3.9 Example of Grid Action Events

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

The sample code in Example 6–7 is an example of JSON that deletes a phone number in the third row of a grid. It is important to note:

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

Example 6–7 Selecting Grid Rows - JSON Request

```
{
    "token":
"0443HC90ZH4pq9CScdvJ+nkecflSJI9q+YGbc7lXrGZ7So=MDE5MDA2ODQ4MjcyMDk2MTUwMjg0NDkyOF
NvYXBVSTEzOTIwNzE5NzY4NzE=",

    "formActions": [
```

```
            {
             "command": "SelectRow",
             "controlID": "1.3"
        }
        ,
        {
            "command": "DoAction",
            "controlID": "59"
        },
        {
            "command": "DoAction",
            "controlID": "4"
        }
    ],
    "formInputs": [
        {
            "value": "6001",
            "id": "4"
        },
        {
            "value": "0",
            "id": "5"
        }
    ],

    "formServiceAction": "U",
    "deviceName": "RESTclient",
    "formName": "P0115_W0115A"
}
```

Example 6–8 Selecting Grid Rows - JSON Response

```
{
  "fs_P0115_W0115A" : {
    "title" : "Phone Numbers",
    "data" : {
      "z_DL01_71" : {
        "id" : 71,
        "title" : "Ray Allen",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblDL01_71",
        "value" : "Ray Allen",
        "editable" : false
      },
      "z_ALPH_52" : {
        "id" : 52,
        "title" : "Who's Who Line",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblWhosWhoLine_52",
        "value" : "Who's Who Line",
        "editable" : false
      },
      "z_AN8_7" : {
        "id" : 7,
        "internalValue" : 6001,
        "title" : "Address Number",
```

```
    "dataType" : 9,
    "staticText" : "Address Number",
    "visible" : true,
    "mathValue" : {
      "currencyCode" : "",
      "currencyDecimals" : 0,
      "zero" : false,
      "negative" : false,
      "intValue" : 6001,
      "decimalPosition" : 0
    },
    "bsvw" : true,
    "longName" : "txtAddressNumber_7",
    "value" : "6001",
    "editable" : false,
    "assocDesc" : "Allen, Ray"
  },
  "z_DL01_54" : {
    "id" : 54,
    "title" : "Allen, Ray",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblDL01_54",
    "value" : "Allen, Ray",
    "editable" : false
  },
  "gridData" : {
    "titles" : {
      "col_28" : "Prefix",
      "col_29" : "Phone Number",
      "col_27" : "Phone Type",
      "col_66" : "Phone Type Description",
      "col_26" : "Line Number"
    },
    "rowset" : [ {
      "rowIndex" : 0,
      "MOExist" : false,
      "z_AR1_28" : {
        "id" : 28,
        "internalValue" : "303",
        "title" : "Prefix",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPrefix_28",
        "value" : "303",
        "editable" : true
      },
      "z_PH1_29" : {
        "id" : 29,
        "internalValue" : "555-1212",
        "title" : "Phone Number",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneNumber_29",
        "value" : "555-1212",
        "editable" : true
      },
```

```
      "z_PHTP_27" : {
        "id" : 27,
        "internalValue" : "CAR",
        "title" : "Phone Type",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneType_27",
        "value" : "CAR",
        "editable" : true
      },
      "z_DL01_66" : {
        "id" : 66,
        "internalValue" : "Car or Mobile",
        "title" : "Phone Type Description",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "sPhoneTypeDescription_66",
        "value" : "Car or Mobile",
        "editable" : false
      },
      "z_RCK7_26" : {
        "id" : 26,
        "internalValue" : 2,
        "title" : "Line Number",
        "dataType" : 9,
        "visible" : true,
        "mathValue" : {
          "currencyDecimals" : 0,
          "zero" : false,
          "negative" : false,
          "intValue" : 2,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "mnLineNumber_26",
        "value" : "2",
        "editable" : false
      }
    } ],
    "summary" : {
      "records" : 1,
      "moreRecords" : false
    }
  },
  "z_AN8_6" : {
    "id" : 6,
    "title" : "Address Number",
    "dataType" : 9,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblAddressNumber_6",
    "value" : "Address Number",
    "editable" : false
  },
  "z_IDLN_32" : {
    "id" : 32,
    "internalValue" : 0,
    "title" : "Who&#39;s Who Line",
```

```
        "dataType" : 9,
        "visible" : true,
        "mathValue" : {
          "currencyDecimals" : 0,
          "zero" : true,
          "negative" : false,
          "intValue" : 0,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "txtWhosWhoLine_32",
        "value" : "0",
        "editable" : false,
        "assocDesc" : "Ray Allen"
      }
    },
    "errors" : [ ],
    "warnings" : [ ]
  },
  "stackId" : 1,
  "stateId" : 1,
  "rid" : "5b45563764cddbc4",
  "currentApp" : "P0115_W0115A",
  "sysErrors" : [ ]
}
```

Example 6–9 Selecting Grid Rows - ADF Application Code

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

```
    private void deletePhone(int row)
    {
        FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P0115_W0115A");
        formRequest.setFormServiceAction(FormRequest.ACTION_UPDATE);
        formRequest.addToFISet("4", addressNumber);
        formRequest.addToFISet("5", "0");

        FSREvent formEvents = new FSREvent();

        // Select row to delete
        formEvents.selectRow("1", row);

        // Press Delete button.
        formEvents.doControlAction("59");

        // Press OK button
        formEvents.doControlAction("4");

        // Add event holder to the request
        formRequest.addFSREvent(formEvents);

        // Execute form service request and process response
        try
        {
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
formRequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_
SERVICE_URI);

            P0115_W0115A_FormParent newFormParent =
```

```
loginEnv.getObjectMapper().readValue(response, P0115_W0115A_FormParent.class);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

Example 6–10 Request Insert Rows - JSON

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

```
{
    "token":
"0449WQ44KE69+ahEmpJPStYG/iqyjTCMpmygYwMkveitD0=MDE5MDEwNTI4ODYyMjg4MDczNzU4OTE4OW
phdmFjbGllbnQxNDI0NjcyNTU4ODAy",
    "deviceName": "javaclient",
    "formName": "P0115_W0115A",
    "formInputs": [
        {
            "id": "4",
            "value": "6001"
        },
        {
            "id": "5",
            "value": "0"
        }
    ],
    "formServiceAction": "U",
    "formActions": [
        {
            "gridAction": {
                "gridID": "1",
                "gridRowInsertEvents": [
                    {
                        "gridColumnEvents": [
                            {
                                "command": "SetGridCellValue",
                                "columnID": "27",
                                "value": "HOM"
                            },
                            {
                                "command": "SetGridCellValue",
                                "columnID": "28",
                                "value": "303"
                            },
                            {
                                "command": "SetGridCellValue",
                                "columnID": "29",
                                "value": "334-4000"
                            }
                        ]
```

```
                    }
                ]
            }
        },
        {
            "command": "DoAction",
            "controlID": "4"
        }
    ]
}
```

Example 6–11 Insert Rows - JSON Response

```
{
  "fs_P0115_W0115A" : {
    "title" : "Phone Numbers",
    "data" : {
      "z_DL01_71" : {
        "id" : 71,
        "title" : "Ray Allen",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblDL01_71",
        "value" : "Ray Allen",
        "editable" : false
      },
      "z_ALPH_52" : {
        "id" : 52,
        "title" : "Who's Who Line",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblWhosWhoLine_52",
        "value" : "Who's Who Line",
        "editable" : false
      },
      "z_AN8_7" : {
        "id" : 7,
        "internalValue" : 6001,
        "title" : "Address Number",
        "dataType" : 9,
        "staticText" : "Address Number",
        "visible" : true,
        "mathValue" : {
          "currencyCode" : "",
          "currencyDecimals" : 0,
          "zero" : false,
          "negative" : false,
          "intValue" : 6001,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "txtAddressNumber_7",
        "value" : "6001",
        "editable" : false,
        "assocDesc" : "Allen, Ray"
      },
      "z_DL01_54" : {
        "id" : 54,
```

```
    "title" : "Allen, Ray",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblDL01_54",
    "value" : "Allen, Ray",
    "editable" : false
  },
  "gridData" : {
    "titles" : {
      "col_28" : "Prefix",
      "col_29" : "Phone Number",
      "col_27" : "Phone Type",
      "col_66" : "Phone Type Description",
      "col_26" : "Line Number"
    },
    "rowset" : [ {
      "rowIndex" : 0,
      "MOExist" : false,
      "z_AR1_28" : {
        "id" : 28,
        "internalValue" : "303",
        "title" : "Prefix",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPrefix_28",
        "value" : "303",
        "editable" : true
      },
      "z_PH1_29" : {
        "id" : 29,
        "internalValue" : "555-1212",
        "title" : "Phone Number",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneNumber_29",
        "value" : "555-1212",
        "editable" : true
      },
      "z_PHTP_27" : {
        "id" : 27,
        "internalValue" : "CAR",
        "title" : "Phone Type",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneType_27",
        "value" : "CAR",
        "editable" : true
      },
      "z_DL01_66" : {
        "id" : 66,
        "internalValue" : "Car or Mobile",
        "title" : "Phone Type Description",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "sPhoneTypeDescription_66",
```

```
      "value" : "Car or Mobile",
      "editable" : false
    },
    "z_RCK7_26" : {
      "id" : 26,
      "internalValue" : 1,
      "title" : "Line Number",
      "dataType" : 9,
      "visible" : true,
      "mathValue" : {
        "currencyCode" : "",
        "currencyDecimals" : 0,
        "zero" : false,
        "negative" : false,
        "intValue" : 1,
        "decimalPosition" : 0
      },
      "bsvw" : true,
      "longName" : "mnLineNumber_26",
      "value" : "1",
      "editable" : false
    }
  }, {
    "rowIndex" : 1,
    "MOExist" : false,
    "z_AR1_28" : {
      "id" : 28,
      "internalValue" : "303",
      "title" : "Prefix",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPrefix_28",
      "value" : "303",
      "editable" : true
    },
    "z_PH1_29" : {
      "id" : 29,
      "internalValue" : "334-4000",
      "title" : "Phone Number",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneNumber_29",
      "value" : "334-4000",
      "editable" : true
    },
    "z_PHTP_27" : {
      "id" : 27,
      "internalValue" : "HOM",
      "title" : "Phone Type",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneType_27",
      "value" : "HOM",
      "editable" : true
    },
    "z_DL01_66" : {
      "id" : 66,
```

```
        "internalValue" : "Home",
        "title" : "Phone Type Description",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "sPhoneTypeDescription_66",
        "value" : "Home",
        "editable" : false
      },
      "z_RCK7_26" : {
        "id" : 26,
        "internalValue" : 2,
        "title" : "Line Number",
        "dataType" : 9,
        "visible" : true,
        "mathValue" : {
          "currencyCode" : "",
          "currencyDecimals" : 0,
          "zero" : false,
          "negative" : false,
          "intValue" : 2,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "mnLineNumber_26",
        "value" : "2",
        "editable" : false
      }
    } ],
    "summary" : {
      "records" : 2,
      "moreRecords" : false
    }
  },
  "z_AN8_6" : {
    "id" : 6,
    "title" : "Address Number",
    "dataType" : 9,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblAddressNumber_6",
    "value" : "Address Number",
    "editable" : false
  },
  "z_IDLN_32" : {
    "id" : 32,
    "internalValue" : 0,
    "title" : "Who&#39;s Who Line",
    "dataType" : 9,
    "visible" : true,
    "mathValue" : {
      "currencyCode" : "",
      "currencyDecimals" : 0,
      "zero" : true,
      "negative" : false,
      "intValue" : 0,
      "decimalPosition" : 0
    },
    "bsvw" : true,
    "longName" : "txtWhosWhoLine_32",
```

```
        "value" : "0",
        "editable" : false,
        "assocDesc" : "Ray Allen"
      }
    },
    "errors" : [ ],
    "warnings" : [ ]
  },
  "stackId" : 1,
  "stateId" : 1,
  "rid" : "93f87493f9d19a14",
  "currentApp" : "P0115_W0115A",
  "sysErrors" : [ ]
}
```

Example 6–12 Insert Rows - ADF Application Code

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

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

```
    public void addPhone()
    {
        try
        {
            loginEnv.getRequiredCapabilities().add(AISClientCapability.GRID);
            FormRequest formRequest = new FormRequest(loginEnv);

            formRequest.setFormName("P0115_W0115A");
            formRequest.setFormServiceAction(FormRequest.ACTION_UPDATE);
            formRequest.addToFISet("4", addressNumber);
            formRequest.addToFISet("5", "0");

            FSREvent formEvents = new FSREvent();
            GridAction gridAction = new GridAction(loginEnv);

            // Add new grid row to grid action
            GridRowInsertEvent rowInsert = new GridRowInsertEvent();
            rowInsert.setGridColumnValue("27", phoneType);
            rowInsert.setGridColumnValue("28", phonePrefix);
            rowInsert.setGridColumnValue("29", phoneNumber);
            gridAction.insertGridRow("1", rowInsert);

            // Add grid action to form events
            formEvents.addGridAction(gridAction);

            // Press OK button
            formEvents.doControlAction("4");

            // Add event holder to the request
            formRequest.addFSREvent(formEvents);

            // Execute form service request and process response
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
formRequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_
SERVICE_URI);
```

```
            P0115_W0115A_FormParent newFormParent =
loginEnv.getObjectMapper().readValue(response, P0115_W0115A_FormParent.class);
        }
        catch (CapabilityException e)
        {
            System.out.println("Grid action capability not supported.");
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

Example 6–13 Request Update Rows - JSON

This sample code is an example of updating a phone number. You must specify the index of the row you want to update. The row index is included in the information returned when you query the grid. Therefore, you must perform a query before you update a row. In this example, the JSON code updates row 1 and sets values in each of the three columns for this row.

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

This sample code shows the phones update request:

```
{
    "token":
"04462IGg6KFX/zTq02/bmYxxXl+SPnJsvD9zbTZ+bDAFxA=MDIwMDEwLTM0NzkyOTQwNDM4MzI5MTc1MD
hqYXZhY2xpZW50MTQyNDY3MzM1NjIzNA==",
    "deviceName": "javaclient",
    "formName": "P0115_W0115A",
    "formInputs": [
        {
            "id": "4",
            "value": "6001"
        },
        {
            "id": "5",
            "value": "0"
        }
    ],
    "formServiceAction": "U",
    "formActions": [
        {
            "gridAction": {
                "gridID": "1",
                "gridRowUpdateEvents": [
                    {
                        "rowNumber": 1,
                        "gridColumnEvents": [
                            {
                                "command": "SetGridCellValue",
                                "columnID": "27",
                                "value": "FAX"
                            },
                            {
```

```
                                "command": "SetGridCellValue",
                                "columnID": "28",
                                "value": "720"
                            },
                            {
                                "command": "SetGridCellValue",
                                "columnID": "29",
                                "value": "334-4010"
                            }
                        ]
                    }
                ]
            }
        },
        {
            "command": "DoAction",
            "controlID": "4"
        }
    ]
}
```

Example 6–14 Update Rows - JSON Response

```
{
  "fs_P0115_W0115A" : {
    "title" : "Phone Numbers",
    "data" : {
      "z_DL01_71" : {
        "id" : 71,
        "title" : "Ray Allen",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblDL01_71",
        "value" : "Ray Allen",
        "editable" : false
      },
      "z_ALPH_52" : {
        "id" : 52,
        "title" : "Who's Who Line",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblWhosWhoLine_52",
        "value" : "Who's Who Line",
        "editable" : false
      },
      "z_AN8_7" : {
        "id" : 7,
        "internalValue" : 6001,
        "title" : "Address Number",
        "dataType" : 9,
        "staticText" : "Address Number",
        "visible" : true,
        "mathValue" : {
          "currencyCode" : "",
          "currencyDecimals" : 0,
          "zero" : false,
          "negative" : false,
          "intValue" : 6001,
```

```
      "decimalPosition" : 0
    },
    "bsvw" : true,
    "longName" : "txtAddressNumber_7",
    "value" : "6001",
    "editable" : false,
    "assocDesc" : "Allen, Ray"
  },
  "z_DL01_54" : {
    "id" : 54,
    "title" : "Allen, Ray",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblDL01_54",
    "value" : "Allen, Ray",
    "editable" : false
  },
  "gridData" : {
    "titles" : {
      "col_28" : "Prefix",
      "col_29" : "Phone Number",
      "col_27" : "Phone Type",
      "col_66" : "Phone Type Description",
      "col_26" : "Line Number"
    },
    "rowset" : [ {
      "rowIndex" : 0,
      "MOExist" : false,
      "z_AR1_28" : {
        "id" : 28,
        "internalValue" : "303",
        "title" : "Prefix",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPrefix_28",
        "value" : "303",
        "editable" : true
      },
      "z_PH1_29" : {
        "id" : 29,
        "internalValue" : "555-1212",
        "title" : "Phone Number",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneNumber_29",
        "value" : "555-1212",
        "editable" : true
      },
      "z_PHTP_27" : {
        "id" : 27,
        "internalValue" : "CAR",
        "title" : "Phone Type",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sPhoneType_27",
        "value" : "CAR",
```

```
      "editable" : true
    },
    "z_DL01_66" : {
      "id" : 66,
      "internalValue" : "Car or Mobile",
      "title" : "Phone Type Description",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : false,
      "longName" : "sPhoneTypeDescription_66",
      "value" : "Car or Mobile",
      "editable" : false
    },
    "z_RCK7_26" : {
      "id" : 26,
      "internalValue" : 1,
      "title" : "Line Number",
      "dataType" : 9,
      "visible" : true,
      "mathValue" : {
        "currencyDecimals" : 0,
        "zero" : false,
        "negative" : false,
        "intValue" : 1,
        "decimalPosition" : 0
      },
      "bsvw" : true,
      "longName" : "mnLineNumber_26",
      "value" : "1",
      "editable" : false
    }
  }, {
    "rowIndex" : 1,
    "MOExist" : false,
    "z_AR1_28" : {
      "id" : 28,
      "internalValue" : "720",
      "title" : "Prefix",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPrefix_28",
      "value" : "720",
      "editable" : true
    },
    "z_PH1_29" : {
      "id" : 29,
      "internalValue" : "334-4010",
      "title" : "Phone Number",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneNumber_29",
      "value" : "334-4010",
      "editable" : true
    },
    "z_PHTP_27" : {
      "id" : 27,
      "internalValue" : "FAX",
      "title" : "Phone Type",
```

```
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneType_27",
      "value" : "FAX",
      "editable" : true
    },
    "z_DL01_66" : {
      "id" : 66,
      "internalValue" : "FAX",
      "title" : "Phone Type Description",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : false,
      "longName" : "sPhoneTypeDescription_66",
      "value" : "FAX",
      "editable" : false
    },
    "z_RCK7_26" : {
      "id" : 26,
      "internalValue" : 2,
      "title" : "Line Number",
      "dataType" : 9,
      "visible" : true,
      "mathValue" : {
        "currencyDecimals" : 0,
        "zero" : false,
        "negative" : false,
        "intValue" : 2,
        "decimalPosition" : 0
      },
      "bsvw" : true,
      "longName" : "mnLineNumber_26",
      "value" : "2",
      "editable" : false
    }
  } ],
  "summary" : {
    "records" : 2,
    "moreRecords" : false
  }
},
"z_AN8_6" : {
  "id" : 6,
  "title" : "Address Number",
  "dataType" : 9,
  "visible" : true,
  "bsvw" : false,
  "longName" : "lblAddressNumber_6",
  "value" : "Address Number",
  "editable" : false
},
"z_IDLN_32" : {
  "id" : 32,
  "internalValue" : 0,
  "title" : "Who&#39;s Who Line",
  "dataType" : 9,
  "visible" : true,
  "mathValue" : {
    "currencyCode" : "",
```

```
          "currencyDecimals" : 0,
          "zero" : true,
          "negative" : false,
          "intValue" : 0,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "txtWhosWhoLine_32",
        "value" : "0",
        "editable" : false,
        "assocDesc" : "Ray Allen"
      }
    },
    "errors" : [ ],
    "warnings" : [ ]
  },
  "stackId" : 1,
  "stateId" : 1,
  "rid" : "d483223d39880eb",
  "currentApp" : "P0115_W0115A",
  "sysErrors" : [ ]
}
```

Example 6–15 Update Rows - ADF Application Code

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

```
    public void updatePhone(int row)
    {
        try
        {
            loginEnv.getRequiredCapabilities().add(AISClientCapability.GRID);
            FormRequest formRequest = new FormRequest(loginEnv);

            formRequest.setFormName("P0115_W0115A");
            formRequest.setFormServiceAction(FormRequest.ACTION_UPDATE);
            formRequest.addToFISet("4", addressNumber);
            formRequest.addToFISet("5", "0");

            FSREvent formEvents = new FSREvent();
            GridAction gridAction = new GridAction(loginEnv);

            // Add grid row update to grid action
            GridRowUpdateEvent rowUpdate = new GridRowUpdateEvent();
            rowUpdate.setGridColumnValue("27", phoneType);
            rowUpdate.setGridColumnValue("28", phonePrefix);
            rowUpdate.setGridColumnValue("29", phoneNumber);

            // Update row identified by its index (zero-based)
            gridAction.updateGridRow("1", row, rowUpdate);

            // Add grid action to form events
            formEvents.addGridAction(gridAction);

            // Press OK button
            formEvents.doControlAction("4");

            // Add event holder to the request
            formRequest.addFSREvent(formEvents);
```

```
            // Execute form service request and process response
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
formRequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_
SERVICE_URI);
            P0115_W0115A_FormParent newFormParent =
loginEnv.getObjectMapper().readValue(response, P0115_W0115A_FormParent.class);
        }
        catch (CapabilityException e)
        {
            System.out.println("Grid action capability not supported.");
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

6.4 Understanding Batch Form Service

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

6.4.1 Batch Form Service - JSON Input and Output

The Batch Form Service requires JSON input and output.

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

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

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

Example 6–16 JSON Input in a Batch Form Request

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

This example shows a request for two form executions: P01012_W01012B and P0115_W0115A. Each have different form interconnect sets, control sets, and so forth. They are completely independent requests.

```
{
    "token":
"044YQDBopCiD3hGUgEOalPmsWsf4/fPpc9ijQs0WcCp/2I=MDE5MDEwMjYxMzQ2OTM1NzUzNDc3MTEwM2
phdmFjbGllbnQxNDI0NzA3NDUzODM2",
    "deviceName": "javaclient",
    "formRequests": [
        {
            "formName": "P01012_W01012B",
            "version": "ZJDE0001",
```

```
            "formServiceAction": "R",
            "formActions": [
                {
                    "command": "SetQBEValue",
                    "controlID": "1[19]",
                    "value": "6001"
                },
                {
                    "command": "SetControlValue",
                    "controlID": "54",
                    "value": "E"
                },
                {
                    "command": "SetCheckboxValue",
                    "controlID": "62",
                    "value": "on"
                },
                {
                    "command": "SetCheckboxValue",
                    "controlID": "63",
                    "value": "on"
                },
                {
                    "command": "DoAction",
                    "controlID": "15"
                }
            ]
        },
        {
            "formName": "P0115_W0115A",
            "formInputs": [
                {
                    "id": "4",
                    "value": "6001"
                },
                {
                    "id": "5",
                    "value": "0"
                }
            ],
            "formServiceAction": "R"
        }
    ]
}
```

Example 6–17 JSON Output in a Batch Form Service

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

The following sample code is an example of JSON output. For fs_0_P01012_W01012B, the 0 indicates it was the first form requested. For fs_1_P0115_W0115A, the 1 indicates it was the second form requested.

```
{
  "fs_0_P01012_W01012B" : {
    "title" : "Work With Addresses",
    "data" : {
      "z_AT1_53" : {
        "id" : 53,
```

```
    "title" : "Search Type",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblSearchType_53",
    "value" : "Search Type",
    "editable" : false
  },
  "z_AT1_54" : {
    "id" : 54,
    "internalValue" : "E",
    "title" : "Search Type",
    "dataType" : 2,
    "staticText" : "Search Type",
    "visible" : true,
    "bsvw" : true,
    "longName" : "txtSearchType_54",
    "value" : "E",
    "editable" : true,
    "assocDesc" : "Employees"
  },
  "z_ALPH_57" : {
    "id" : 57,
    "title" : "Alpha Name",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblAlphaName_57",
    "value" : "Alpha Name",
    "editable" : false
  },
  "z_ALPH_58" : {
    "id" : 58,
    "internalValue" : " ",
    "title" : "Alpha Name",
    "dataType" : 2,
    "staticText" : "Alpha Name",
    "visible" : true,
    "bsvw" : false,
    "longName" : "txtAlphaName_58",
    "value" : " ",
    "editable" : true
  },
  "z_DL01_66" : {
    "id" : 66,
    "title" : "Employees",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblDL01_66",
    "value" : "Employees",
    "editable" : false
  },
  "z_EV01_63" : {
    "id" : 63,
    "internalValue" : "1",
    "title" : "Display Address",
    "dataType" : 1,
    "visible" : true,
    "bsvw" : false,
```

```
    "longName" : "chkDisplayAddress_63",
    "value" : "on",
    "editable" : true
  },
  "z_EV01_62" : {
    "id" : 62,
    "internalValue" : "1",
    "title" : "Display Phone",
    "dataType" : 1,
    "visible" : true,
    "bsvw" : false,
    "longName" : "chkDisplayPhone_62",
    "value" : "on",
    "editable" : true
  },
  "gridData" : {
    "titles" : {
      "col_19" : "Address Number",
      "col_20" : "Alpha Name",
      "col_40" : "Address Line 1",
      "col_44" : "City",
      "col_81" : "Prefix",
      "col_46" : "Phone Number",
      "col_47" : "Phone Type",
      "col_48" : "Long Address",
      "col_49" : "Industry Class",
      "col_50" : "Sch Typ",
      "col_51" : "Tax ID"
    },
    "rowset" : [ {
      "rowIndex" : 0,
      "MOExist" : true,
      "z_AN8_19" : {
        "id" : 19,
        "internalValue" : 6001,
        "title" : "Address Number",
        "dataType" : 9,
        "visible" : true,
        "mathValue" : {
          "currencyDecimals" : 0,
          "zero" : false,
          "negative" : false,
          "intValue" : 6001,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "mnAddressNumber_19",
        "value" : "6001",
        "editable" : false
      },
      "z_ALPH_20" : {
        "id" : 20,
        "internalValue" : "Allen, Ray",
        "title" : "Alpha Name",
        "dataType" : 2,
        "visible" : true,
        "bsvw" : true,
        "longName" : "sAlphaName_20",
        "value" : "Allen, Ray",
        "editable" : false
```

```
},
"z_ADD1_40" : {
  "id" : 40,
  "internalValue" : "410 17th Avenue",
  "title" : "Address Line 1",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sAddressLine1_40",
  "value" : "410 17th Avenue",
  "editable" : false
},
"z_CTY1_44" : {
  "id" : 44,
  "internalValue" : "Denver",
  "title" : "City",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sCity_44",
  "value" : "Denver",
  "editable" : false
},
"z_AR1_81" : {
  "id" : 81,
  "internalValue" : "303",
  "title" : "Prefix",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sPrefix_81",
  "value" : "303",
  "editable" : false
},
"z_PH3_46" : {
  "id" : 46,
  "internalValue" : "555-1212",
  "title" : "Phone Number",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sPhoneNumber_46",
  "value" : "555-1212",
  "editable" : false
},
"z_PHTP_47" : {
  "id" : 47,
  "internalValue" : "CAR",
  "title" : "Phone Type",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sPhoneType_47",
  "value" : "CAR",
  "editable" : false
},
"z_ALKY_48" : {
  "id" : 48,
  "internalValue" : " ",
  "title" : "Long Address",
```

```
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sLongAddress_48",
          "value" : " ",
          "editable" : false
        },
        "z_SIC_49" : {
          "id" : 49,
          "internalValue" : " ",
          "title" : "Industry Class",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sIndustryClass_49",
          "value" : " ",
          "editable" : false
        },
        "z_AT1_50" : {
          "id" : 50,
          "internalValue" : "E",
          "title" : "Sch Typ",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sSchTyp_50",
          "value" : "E",
          "editable" : false
        },
        "z_TAX_51" : {
          "id" : 51,
          "internalValue" : "798525841",
          "title" : "Tax ID",
          "dataType" : 2,
          "visible" : true,
          "bsvw" : true,
          "longName" : "sTaxID_51",
          "value" : "798525841",
          "editable" : false
        }
      } ],
      "summary" : {
        "records" : 1,
        "moreRecords" : false
      }
    }
  },
  "errors" : [ ],
  "warnings" : [ ]
},
"fs_1_P0115_W0115A" : {
  "title" : "Phone Numbers",
  "data" : {
    "z_DL01_71" : {
      "id" : 71,
      "title" : "Ray Allen",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : false,
      "longName" : "lblDL01_71",
```

```
    "value" : "Ray Allen",
    "editable" : false
  },
  "z_ALPH_52" : {
    "id" : 52,
    "title" : "Who's Who Line",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblWhosWhoLine_52",
    "value" : "Who's Who Line",
    "editable" : false
  },
  "z_AN8_7" : {
    "id" : 7,
    "internalValue" : 6001,
    "title" : "Address Number",
    "dataType" : 9,
    "staticText" : "Address Number",
    "visible" : true,
    "mathValue" : {
      "currencyCode" : "",
      "currencyDecimals" : 0,
      "zero" : false,
      "negative" : false,
      "intValue" : 6001,
      "decimalPosition" : 0
    },
    "bsvw" : true,
    "longName" : "txtAddressNumber_7",
    "value" : "6001",
    "editable" : false,
    "assocDesc" : "Allen, Ray"
  },
  "z_DL01_54" : {
    "id" : 54,
    "title" : "Allen, Ray",
    "dataType" : 2,
    "visible" : true,
    "bsvw" : false,
    "longName" : "lblDL01_54",
    "value" : "Allen, Ray",
    "editable" : false
  },
  "gridData" : {
    "titles" : {
      "col_28" : "Prefix",
      "col_29" : "Phone Number",
      "col_27" : "Phone Type",
      "col_66" : "Phone Type Description",
      "col_26" : "Line Number"
    },
    "rowset" : [ {
      "rowIndex" : 0,
      "MOExist" : false,
      "z_AR1_28" : {
        "id" : 28,
        "internalValue" : "303",
        "title" : "Prefix",
        "dataType" : 2,
```

```
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPrefix_28",
      "value" : "303",
      "editable" : true
    },
    "z_PH1_29" : {
      "id" : 29,
      "internalValue" : "555-1212",
      "title" : "Phone Number",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneNumber_29",
      "value" : "555-1212",
      "editable" : true
    },
    "z_PHTP_27" : {
      "id" : 27,
      "internalValue" : "CAR",
      "title" : "Phone Type",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : true,
      "longName" : "sPhoneType_27",
      "value" : "CAR",
      "editable" : true
    },
    "z_DL01_66" : {
      "id" : 66,
      "internalValue" : "Car or Mobile",
      "title" : "Phone Type Description",
      "dataType" : 2,
      "visible" : true,
      "bsvw" : false,
      "longName" : "sPhoneTypeDescription_66",
      "value" : "Car or Mobile",
      "editable" : false
    },
    "z_RCK7_26" : {
      "id" : 26,
      "internalValue" : 1,
      "title" : "Line Number",
      "dataType" : 9,
      "visible" : true,
      "mathValue" : {
        "currencyDecimals" : 0,
        "zero" : false,
        "negative" : false,
        "intValue" : 1,
        "decimalPosition" : 0
      },
      "bsvw" : true,
      "longName" : "mnLineNumber_26",
      "value" : "1",
      "editable" : false
    }
  }, {
    "rowIndex" : 1,
    "MOExist" : false,
```

```
"z_AR1_28" : {
  "id" : 28,
  "internalValue" : "720",
  "title" : "Prefix",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : true,
  "longName" : "sPrefix_28",
  "value" : "720",
  "editable" : true
},
"z_PH1_29" : {
  "id" : 29,
  "internalValue" : "334-4010",
  "title" : "Phone Number",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : true,
  "longName" : "sPhoneNumber_29",
  "value" : "334-4010",
  "editable" : true
},
"z_PHTP_27" : {
  "id" : 27,
  "internalValue" : "FAX",
  "title" : "Phone Type",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : true,
  "longName" : "sPhoneType_27",
  "value" : "FAX",
  "editable" : true
},
"z_DL01_66" : {
  "id" : 66,
  "internalValue" : "FAX",
  "title" : "Phone Type Description",
  "dataType" : 2,
  "visible" : true,
  "bsvw" : false,
  "longName" : "sPhoneTypeDescription_66",
  "value" : "FAX",
  "editable" : false
},
"z_RCK7_26" : {
  "id" : 26,
  "internalValue" : 2,
  "title" : "Line Number",
  "dataType" : 9,
  "visible" : true,
  "mathValue" : {
    "currencyDecimals" : 0,
    "zero" : false,
    "negative" : false,
    "intValue" : 2,
    "decimalPosition" : 0
  },
  "bsvw" : true,
  "longName" : "mnLineNumber_26",
  "value" : "2",
```

```
            "editable" : false
          }
        } ],
        "summary" : {
          "records" : 2,
          "moreRecords" : false
        }
      },
      "z_AN8_6" : {
        "id" : 6,
        "title" : "Address Number",
        "dataType" : 9,
        "visible" : true,
        "bsvw" : false,
        "longName" : "lblAddressNumber_6",
        "value" : "Address Number",
        "editable" : false
      },
      "z_IDLN_32" : {
        "id" : 32,
        "internalValue" : 0,
        "title" : "Who&#39;s Who Line",
        "dataType" : 9,
        "visible" : true,
        "mathValue" : {
          "currencyDecimals" : 0,
          "zero" : true,
          "negative" : false,
          "intValue" : 0,
          "decimalPosition" : 0
        },
        "bsvw" : true,
        "longName" : "txtWhosWhoLine_32",
        "value" : "0",
        "editable" : false,
        "assocDesc" : "Ray Allen"
      }
    },
    "errors" : [ ],
    "warnings" : [ ]
  }
}
```


6.4.2 Implementing the Batch Form Service

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

6.4.2.1 Batch Request Parent Class

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

Example 6–18 Batch Request Parent Class

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

the most important aspect of the class, to enable the deserialization from JSON to this object.

The form objects, which are P01012_W01012B and P0115_W0115A in the following sample code, are the same ones generated by the AIS Client Class Generator and are used for a single form request.

```
package com.oracle.e1.formservicetypes;


public class BatchResponseParent
{
    private P01012_W01012B fs_0_P01012_W01012B;
    private P0115_W0115A fs_1_P0115_W0115A;

    public BatchResponseParent()
    {
    }

    // Getters and setters for each class variable.

}
```

6.4.2.2 Performing a Batch Form Request

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

To create this method:

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

Example 6–19 Batch Form Request

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

```
public void performBatchFormServiceRequest()
{
    BatchFormRequest batchFormRequest = new BatchFormRequest(loginEnv);

    SingleFormRequest w01012BFormRequest = new SingleFormRequest();
    w01012BFormRequest.setFormName("P01012_W01012B");
    w01012BFormRequest.setVersion("ZJDE0001");
    w01012BFormRequest.setFormServiceAction(FormRequest.ACTION_READ);

    FSREvent formEvents = new FSREvent();
    formEvents.setQBEValue("1[19]", addressNumber);
    formEvents.setFieldValue("54", "E");
    formEvents.checkBoxChecked("62");
    formEvents.checkBoxChecked("63");
    formEvents.doControlAction("15");
    w01012BFormRequest.addFSREvent(formEvents);
```

```
        batchFormRequest.getFormRequests().add(w01012BFormRequest);

        SingleFormRequest w0115AFormRequest = new SingleFormRequest();
        w0115AFormRequest.setFormName("P0115_W0115A");
        w0115AFormRequest.setFormServiceAction(FormRequest.ACTION_READ);
        w0115AFormRequest.addToFISet("4", addressNumber);
        w0115AFormRequest.addToFISet("5", "0");
        batchFormRequest.getFormRequests().add(w0115AFormRequest);

        try
        {
            // Execute form service request and process response
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
batchFormRequest, JDERestServiceProvider.POST_METHOD,
JDERestServiceProvider.BATCH_FORM_SERVICE_URI);

            BatchResponseParent newBatchResponseParent =
loginEnv.getObjectMapper().readValue(response, BatchResponseParent.class);

            if (newBatchResponseParent != null)
            {
                w01012BFormResponse = newBatchResponseParent.getFs_0_P01012_
W01012B();
                if (w01012BFormResponse != null)
                {
                    // Retrieve form response data
                }

                w0115AFormResponse = newBatchRequestParent.getFs_1_P0115_W0115A();
                if (w0115AFormResponse != null)
                {
                    // Retrieve form response data
                }
            }
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

6.5 Working with the EnterpriseOne REST Services Interface

You can make REST calls directly to the AIS Server without using APIs from the AIS_client.jar. 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://<server>:<port>/jderest/<URI>
```

See Section 2.3.4, "EnterpriseOne Rest Services Interface" for an illustration of JSON input and output communication between an ADF device and the AIS Server and between the AIS Server and the EnterpriseOne HTML Server.

The following services are available on the AIS Server, along with the HTTP method used to access them. Unless otherwise stated, the request and response are in the form of JSON strings.

URI	HTTP Method	Description
/defaultconfig	GET	Response includes information about the AIS Server including the release level, JAS configuration and capabilities list. Different versions of AIS have different capabilities, even if the client API you are using is up to date with the latest capabilities, the AIS Server may not be.
/tokenrequest	POST	Based on the input, the response will contain login information including a login token and user details.
/formservice	POST	Based on the input, the response will contain a JSON representation of the form requested.
/batchformservice	POST	Based on the input, the response will contain a JSON representation of all of the forms requested.
/file/gettext	POST	Based on the input, the response will contain the text for the first text media object.
/file/updatetext	POST	Based on the input, the response will contain the status of the text update.
/file/list	POST	Based on the input, the response will contain the list of media objects for the structure and key requested.
/file/upload	POST (Multi-part Form)	The response will contain the details of the uploaded file, including the media object sequence number
/file/download	POST	This response is a Multi-Part form including the data for the attachment.
/file/delete	POST	The response indicates the success or failure to delete the media object for the sequence number passed in.
/poservice	POST	Based on the input, the response will contain the processing option values for the requested application and version.
/log	POST	Based on the input, the AIS Server will write a log entry with the information passed to the log service.
/appstack	POST	This service is available starting with the Tools 9.1 Update 5 release. Based on the input, the response will contain the current form open on the stack and any stack related information.
/jargonservice	POST	This service enables you to retrieve data item descriptions for any EnterpriseOne data dictionary item based on the users language and jargon (system) code. This service depends on language packs applied to the EnterpriseOne system as well as data item description overrides entered with jargon codes. If there is no language pack or no overrides, then the base data item description is returned.

6.5.1 Using a REST Services Client to Interact with AIS

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

Example 6–20 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.domain.com",
   "jasPort": "8412",
   "jasProtocol": "http",
   "defaultEnvironment": "DV910",
   "defaultRole": "*ALL",
   "displayEnvironment": true,
   "displayRole": true,
   "displayJasServer": false,
   "defaultJasServer": "http://jasserver.domain.com:port",
   "ssoAllowed": true,
   "allowSelfSignedSsl": false,
   "sessionTimeout": "30",
   "timeToLive": "720",
   "aisVersion": "EnterpriseOne 9.1.4.6",
   "capabilityList":    [
            {
         "name": "grid",
         "shortDescription": "Grid Actions",
         "longDescription": "Ability to update, insert and delete grid records.",
         "asOfRelease": "9.1.4.4"
      },
            {
         "name": "editable",
         "shortDescription": "Enabled/Disabled",
         "longDescription": "Ability to indicate if form field or grid cell is
editable (enabled) or not (disabled).",
         "asOfRelease": "9.1.4.4"
      },
            {
         "name": "log",
         "shortDescription": "Logging",
         "longDescription": "Endpoint exposed for logging to AIS server log from
client",
         "asOfRelease": "9.1.4.6"
      },
            {
         "name": "processingOption",
         "shortDescription": "Processing Options",
         "longDescription": "Processing Option Service exposed for fetching PO
values from E1",
         "asOfRelease": "9.1.4.6"
      },
            {
         "name": "ignoreFDAFindOnEntry",
         "shortDescription": "Ignore FDA Find On Entry",
         "longDescription": "Ability to use the IgnoreFDAFindOnEntry flag",
         "asOfRelease": "9.1.4.6"
```

```
      }
   ],
   "requiredCapabilityMissing": false,
   "keepJasSessionOpen": true,
   "jasSessionCookieName": "JSESSIONID"
}
```

Example 6–21 Acceptable Input for the tokenrequest Service in JSON

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

Example 6–22 Acceptable Output for the token request Service in JSON Response

```
{
   "username": "jde",
   "environment": "JDV910",
   "role": "*ALL",
   "jasserver": "http://jasserver.domain.com:port",
   "userInfo":    {
      "token":
"044kbwMICIsL8P4jttMj/VtpnEhrSK17i7fa2q8V+hVDlc=MDE4MDEwNjA1NTMwODkwMzQ1ODY0MTkyUk
VTVGNsaWVudDE0MTI2MDg3MDgzODY=",
      "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"
}
```

6.5.1.1 Form Request Attributes

The following list contains a description of the form request attributes. For an example of a form request, see Example 6–23.

- **maxPageSize**. (optional) If you do not specify a value for this attribute, then AIS will only return 100 rows. If you want more rows than that, you must specify a value. However, AIS still will not return more rows than the max record number specified in the JAS ini (DBFetchLimitBeforeWarning).

Note: Consider the value in the DBFetchLimitBeforeWarning setting to be the maximum value that the FSR request can send a value in for max rows with each request. The value sent in will be the one that is used, unless it is greater than the jas setting for DBFetchLimitBeforeWarning. The default shipped value is 2000

- **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 6–23 Form Request

```
{
    "token":
"044RnByWbW3FbLzpxWjSg55QzZWguAGnYqUNMlyB30IgyU=MDE5MDA2ODg0MDE5NjYwNzM1ODcyNDA5Nl
NvYXBVSTEzODc0ODc4OTEzNTc=",
    "maxPageSize": "10",
    "returnControlIDs": "1[19,20]58|62",
    "version": "ZJDE0001",
    "formInputs": [
        {
            "value": "E",
            "id": "2"
        }
    ],
    "formActions": [
        {
            "value": "E",
```

```
            "command": "SetQBEValue",
            "controlID": "1[50]"
        },
        {
            "value": "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"
}
```

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

You have the option to use this technique if you have FDA changes locally and want to test the JSON output while still accessing the AIS Server. This is the closest approximation of how the 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\c
   lasses\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"
}
```

6.6 Understanding Text Media Object Attachments

The AIS Server provides a method to get and update the first text attachment stored for a particular media object structure key. The following two text media object services are available in the AIS_Client API:

- gettext
- updatetext

6.6.1 gettext Service

The URL for getting text media objects is:

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

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

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

Field	Value Description	Example
token (String)	The token you received when you executed a token request.	"044TqjlQVSNHKvokvhWFKa8MYckPgG…"
moStructure (String)	The Media Object Structure.	"ABGT" or "GT0801A"
moKey (Array of Strings)	The key to that media object structure.	["7"] or ["2015","283", "2"]
formName (String)	The form where this media object is used.	"P01012_W01012B"
version (String)	The version of the app where this media object is used.	"ZJDE0001"
deviceName (String)	The device making the call to the service.	"E1ADFApps" or "REST Client"

Example 6–24 gettext Service Input

```
{
    "token":
"044TqjlQVSNHKvokvhWFKa8MYckPgGJdrd3CEDkzz2YjLQ=MDE5MDA2Nzg3NjEzOTA2MTY1MzIwMTkyMF
NvYXBVSTEzOTk1ODA5MzQ0OTg=",
    "moStructure": "ABGT",
```

```
    "moKey": [
        "7"
    ],
    "formName": "P01012_W01012B",
    "version": "ZJDE0001",
    "deviceName": "RESTclient"
}
```

Example 6–25 gettext Service Response

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

6.6.2 Update Text Service

The URL for updating text media objects is:

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

Only plain text is supported.

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

Field	Value Description	Example
token (String)	The token you received when you executed a token request.	"044TqjlQVSNHKvokvhWFKa8MYckPgG…"
moStructure (String)	The media object structure.	"ABGT" or "GT0801A"
moKey (Array of Strings)	The key to the media object structure.	["7"] or ["2015","283", "2"]
formName (String)	The form where this media object is used.	"P01012_W01012B"
version (String)	The version of the application where the media object is used.	"ZJDE0001"
deviceName (String)	The device making the call to the service.	"RESTClient" or "E1ADFApps"
inputText (String)	The text you are updating for the first text media object.	"Update Text"
appendText (Boolean)	A flag to indicate if the text should append to (true) or replace (false) the existing first text attachment.	true or false

Example 6–26 Update Text Service Input

```
{
    "token":
"044X9NgLP5VHqIW0zJcWseFjMtjrnZ35TUFmVV3NW9a9g4=MDE5MDA2NDY5MzQ1NjkzNDM1MjE3NjEyOF
NvYXBVSTE0MDE3MTc4MDkzNTk=",
    "moStructure": "ABGT",
    "moKey": [
        "7"
    ],
    "formName": "P09E2011_W09E2011A",
```

```
    "version": "ZJDE0001",
    "inputText": "\nFirst Line \nSecond Line \nThird Line \nFourth Line \nFifth
line",
    "appendText": false,
    "deviceName": "RESTclient"
}
```

Example 6–27 Update Text Service Response

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

6.6.3 Text ADF Framework API

The Text ADF Framework API provides methods for managing text media objects from ADF applications. The same two methods described in the preceding sections, getText and updateText, are exposed in this API.

MediaObjectOperations Class

The following table describes the methods in the MediaObjectOperations class:

Return	Method	Description
MediaObjectGetTextResponse	getTextMediaObject(LoginEnvironment loginEnv, MediaObjectGetTextRequest MediaObjectTextRequest) throws Exception	Takes a MediaObjectGetTextRequest, calls the get text service, replaces unicode returns and line feeds with system line separator and returns a MediaObjectGetTextResponse with the resulting text attachment value.
MediaObjectUpdateTextResponse	updateTextMediaObject(LoginEnvironment loginEnv, MeidaObjectUpdatedRequest MediaObjectUpdatedTextRequest) throws Exception	Takes a MediaObjectUpdateTextRequest, calls the update text service and returns result in MediaObjectUpdateTextResponse

Example 6–28 Get Example

```
    public void getMediaObjectText()
    {
        MediaObjectGetTextRequest moGetText = new
MediaObjectGetTextRequest(loginEnv);
        moGetText.setFormName(formName);
        moGetText.setVersion(version);
        moGetText.setMoStructure(moStructure);

        // Set media object key value
        moGetText.addMoKeyValue(moKey);

        try
        {
            MediaObjectGetTextResponse moResponse =
MediaObjectOperations.getTextMediaObject(loginEnv, moGetText);

            setFirstText(moResponse.getText());
        }
        catch (JDERestServiceException e)
        {
```

```
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

Example 6–29 Append Example

```
    private void appendMediaObjectText()
    {
        MediaObjectUpdateTextRequest moAppendText = new
MediaObjectUpdateTextRequest(loginEnv);
        moAppendText.setFormName(formName);
        moAppendText.setVersion(version);
        moAppendText.setMoStructure(moStructure);

        // Set media object key
        moAppendText.addMoKeyValue(moKey);
        moAppendText.setAppendText(true);
        moAppendText.setInputText(text);

        try
        {
            MediaObjectUpdateTextResponse moResponse =
MediaObjectOperations.updateTextMediaObject(loginEnv, moAppendText);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

Example 6–30 Update Example

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

        // Set media object key value
        moUpdateText.addMoKeyValue(moKey);
        moUpdateText.setAppendText(false);
        moUpdateText.setInputText(getFirstText());

        try
        {
            MediaObjectUpdateTextResponse moResponse =
MediaObjectOperations.updateTextMediaObject(loginEnv, moUpdateText);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
```

```
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

6.7 Understanding the Media Object API for Photo Media Object Attachments

You can incorporate photo attachments into Oracle ADF applications using the AIS_Client.jar, which contains a Media Object API that provides classes and methods that enable access to media objects from EnterpriseOne. It works in conjunction with the AIS Server and the MediaObjectRequest capability of the EnterpriseOne HTML Server.

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

- List
- Download
- Upload
- Delete

6.7.1 List

Method - getMediaObjectList
Description - Returns a list of Media Object details based on the information provided in the request.
Returns - MediaObjectListResponse
Parameters - LoginEnvironment, MediaObjectListRequest

MediaObjectListRequest

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

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

MediaObjectListResponse

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

MediaObjectListItem

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

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

6.7.2 Download

Method - downloadMediaObject
Description - Downloads a full sized media object to a file on the device, based on the key information provided in the request.
Returns - MediaObjectDownloadResponse
Parameters - LoginEnvironment, MediaObjectDownloadRequest

MediaObjectDownloadRequest

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

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

FileAttachment

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

6.7.3 Upload

Method - uploadMediaObject
Description - Uploads a single file to the media object database based on key values provided in the request.
Returns - MediaObjectUploadResponse
Parameters - LoginEnvironment, MediaObjectUploadRequest

MediaObjectUploadRequest

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

FileAttachment

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

MediaObjectUploadResponse

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

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

6.7.4 Delete

Method - deleteMediaObject
Description - Deletes the file in the media object database based on key values provided in the request. Deletes the file on the device file system also.
Returns - MediaObjectDeleteResponse
Parameters - LoginEnvironment, MediaObjectDeleteRequest

MediaObjectDeleteRequest

Field	Type	Description
moStructure	String	The Media Object structure ID (e.g. GT0801, GT48100).
moKey	ArrayList <String>	The set of key values, as strings, that is the key of the media object. The API will handle formatting into a bar delimited set. Use the JDEmfUtilities.convertMillisecondsToYMDStrin g() method to pass dates.
formName	String	The application and form usually associated with this media object structure (e.g. P01012_ W01012B). This is for security purposes. Security for media objects is based on the form being used.
version	String	The version of the application noted in the formName. This is for security purposes. Security for media objects is based on the form being used.
sequence	int	The sequence number of this individual file in the set of attachments.
fileLocation	String	The location of the file on the device file system.

MediaObjectDeleteResponse

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

6.7.5 Add Media Object URL

This section discusses adding a media object URL.

6.7.5.1 Add URL

Method - addUrlMediaObject
Description - Adds a media object of the type url.
Returns - MediaObjectAddUrlResponse
Parameters - LoginEnvironment, MediaObjectAddUrlRequest

MediaObjectAddURLRequest

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

MediaObjectAddUrlResponse

Field	Type	Description
urlText	String	The text of the url (for example, www.oracle.com or http://www.oracle.com).
saveUrl	String	The url as it was saved in the media object table F00165.
sequence	int	The sequence number for the newly added url type media object.

Example 6–31 Add Media Object URL

```
    public void addMediaObjectURL()
    {
        MediaObjectAddUrlRequest moAddUrlRequest = new
MediaObjectAddUrlRequest(loginEnv);
        moAddUrlRequest.setFormName(formName);
        moAddUrlRequest.setVersion(version);
        moAddUrlRequest.setMoStructure(moStructure);
        moAddUrlRequest.addMoKeyValue(moKey);
        moAddUrlRequest.setUrlText(moURL);

        try
        {
            MediaObjectAddUrlResponse mediaObjectAddUrlResponse =
MediaObjectOperations.addUrlMediaObject(loginEnv, moAddUrlRequest);

        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
```

```
            System.out.println(e);
        }
    }
```

6.8 Understanding Processing Options

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

The URI for the service is:

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

Example 6–32 ADF Code Retrieve PO Values

The input JSON is:

```
{
 "token":
"044uZUG2Uk1Vd6hzmAhPfILitA2pVLDVKLOYdh4HR71D7s=MDE5MDA2ODM3NTQ3NzU4MDMwNTg2MzY4MF
NvYXBVSTEzOTcwNTYxMTIxMTE=",

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

}
```

The response JSON is like this:

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

   },
   "errors": ""
```

```
}
```

6.8.1 Using the AIS Service for Processing Options in ADF Application

You can call the service passing an application and version. You can obtain the version from the ADF application processing options, which specify the version of the EnterpriseOne application that the ADF application uses.

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

Example 6–33 ADF Code Retrieve PO Values

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

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

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

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

Example 6–34 ADF Code Retrieve PO Values

```
    private void retrievePOValues()
    {
        try
        {
            loginEnv.getRequiredCapabilities().add(AISClientCapability.PROCESSING_
OPTION);
            ProcessingOptionRequest poRequest = new
ProcessingOptionRequest(loginEnv);
            poRequest.setApplicationName("P01012");
            poRequest.setVersion("ZJDE0001");

            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
poRequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.PO_SERVICE);
            ProcessingOptionsSet p01012POSet =
loginEnv.getObjectMapper().readValue(response, ProcessingOptionsSet.class);

            if (p01012POSet != null)
            {
                String searchType = (String)
```

```
p01012POSet.getOptionValue("SearchTypeDefault_7");
                String typeCode = (String) p01012POSet.getOptionValue("cTypeCode_
11");
            }
        }
        catch (CapabilityException e)
        {
            System.out.println("Processing Option capability not supported");
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

6.9 Understanding the Application Stack Service

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

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

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

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

6.9.1 Service Endpoint

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

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

6.9.2 Capability

The application stack capability is exposed in the default configuration as "applicationStack." You must add this to your required capability list in your ADF

application using the AISClientCapability.APPLICATION_STACK constant before using this capability.

6.9.3 Prerequisite

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

6.9.4 JSON Example of an Application Stack Request

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

- Accessing the Address Book application to find a record.

  See Open Application: Request and Response.

- Taking the row exit to the Phones form.

  See Execute Actions on Application: Request and Response.

- Adding a phone number.

  See Adding a Phone Number.

- Saving and closing the application stack.

  See Execute Close Application: Request and Response.

6.9.4.1 Open Application: Request and Response

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

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

> **Important:** The maxPageSize indicated in this originating request will be the max page size for all requests in this stack. If you do not specify a maxPageSize, it uses the default setting in JAS_INI (DBFetchLimitBeforeWarning).

Example 6–35 Open Application - Request

```
{
    "token":
"044J57pHCCB3AzH7/W6Vetm0+yVG6pnKOPl823587SfY6o=MDE5MDA2MTYxOTE0NTE0NjE2OTEyNTg4OF
NvYXBVSTE0MDEyMTA5MDE1MDM=",
    "action": "open",
    "formRequest": {
        "returnControlIDs": "54|1[19,20]",
        "maxPageSize": "5",
        "formName": "P01012_W01012B",
        "version": "ZJDE0001",
        "findOnEntry": "TRUE",
        "formInputs": [
            {
                "value": "E",
```

```
                "id": "2"
            }
        ]
    },
    "deviceName": "RESTclient"
}
```

Example 6–36 Open Application - Response

```
{
   "fs_P01012_W01012B":    {
      "title": "Work With Addresses",
      "data":       {
… condensed….
      },
      "errors": [],
      "warnings": []
   },
   "stackId": 1,
   "stateId": 1,
   "rid": "e51b593df7a884ea",
   "currentApp": "P01012_W01012B_ZJDE0001",
   "sysErrors": []
}
```

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

6.9.4.2 Execute Actions on Application: Request and Response

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

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

Example 6–37 Execute Actions on Application - Request

```
{
    "token":
"044DCcituJDHbxBhPcgOuhCb0Av/xnNiUFxqPTVD6hDfnU=MDE5MDA2MTc1NjYwMzYwMzk0MTE0OTY5Nl
NvYXBVSTE0MDEyMTkzOTIxMzA=",
    "stackId": 1,
    "stateId": 1,
    "rid": "e51b593df7a884ea",
    "action": "execute",
    "actionRequest": {
        "returnControlIDs": "32|7|1[28,29,66]",
        "maxPageSize": "4",
        "formOID": "W01012B",
        "formActions": [
            {
                ".type": "com.oracle.e1.jdemf.FormAction",
                "command": "SelectRow",
```

```
                "controlID": "1.0"
            },
            {
                ".type": "com.oracle.e1.jdemf.FormAction",
                "command": "DoAction",
                "controlID": "65"
            }
        ]
    },
    "deviceName": "RESTclient"
}
```

Example 6–38 Execute Actions on Application - Response

```
{
   "fs_P0115_W0115A":    {
      "title": "Phone Numbers",
      "data":       {

….condensed..
            "summary":             {
               "records": 3,
               "moreRecords": false
            }
         }
      },
      "errors": [],
      "warnings": []
   },
   "stackId": 1,
   "stateId": 2,
   "rid": "e51b593df7a884ea",
   "currentApp": "P0115_W0115A_ZJDE0001",
   "sysErrors": []
}
```

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

6.9.4.3 Adding a Phone Number

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

Example 6–39 Adding a Phone Number - Request

```
{
    "token":
"044DCcituJDHbxBhPcgOuhCb0Av/xnNiUFxqPTVD6hDfnU=MDE5MDA2MTc1NjYwMzYwMzk0MTE0OTY5Nl
NvYXBVSTE0MDEyMTkzOTIxMzA=",
    "stackId": 1,
    "stateId": 2,
    "rid": "e51b593df7a884ea",
    "action": "execute",
    "actionRequest": {
        "formOID": "W0115A",
        "formActions": [
```

```
            {
                "gridAction": {
                    "gridID": "1",
                    "gridRowInsertEvents": [
                        {
                            "gridColumnEvents": [
                                {
                                    "value": "720",
                                    "command": "SetGridCellValue",
                                    "columnID": "28"
                                },
                                {
                                    "value": "12345",
                                    "command": "SetGridCellValue",
                                    "columnID": "29"
                                },
                                {
                                    "value": "CAR",
                                    "command": "SetGridCellValue",
                                    "columnID": "27"
                                }
                            ]
                        }
                    ]
                }
            }
        ]
    },
    "deviceName": "RESTclient",
    "ssoEnabled": true
}
```

6.9.4.4 Execute Close Application: Request and Response

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

Example 6–40 Execute Close Application - Request

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

Example 6–41 Execute Close Application - Response

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

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

6.9.4.5 ADF Application Example

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

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

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

Example 6–42 Open Application

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

```
    public void getABRecords()
    {
    // Retrieve a list of employees from Address Book (P01012)
        loginEnv.getRequiredCapabilities().add(AISClientCapability.APPLICATION_
STACK);
        FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P01012_W01012B");
        formRequest.setVersion("ZJDE0001");
        formRequest.setReturnControlIDs("54|1[19,20]");
        formRequest.setFormServiceAction("R");
        formRequest.setMaxPageSize("10");
```

```
        FSREvent formEvents = new FSREvent();
        formEvents.setFieldValue("54", "E");
        formEvents.doControlAction("15");       // Find button
        formRequest.addFSREvent(formEvents);

        try
        {
            // Open P01012_W01012B app
            String response = appStackAddress.open(loginEnv, formRequest);

            addressBookRecords = loginEnv.getObjectMapper().readValue(response,
P01012_W01012B_FormParent.class);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

Example 6–43 Get ALT Address Records

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

```
    public String goToAltAddress()
    {
        String nav = "";

        try
        {
            int rowIndex = getSelectedAddressBookRecord();
            if (rowIndex >= 0)
            {
                loginEnv.getRequiredCapabilities().add(AISClientCapability.SELECT_
ALL_GRID_ROWS);
                ActionRequest getAddressAction = new ActionRequest();
                getAddressAction.setFormOID("W01012B");
                FSREvent selectEvent = new FSREvent();
                selectEvent.unselectAllGridRows(loginEnv, "1");
                selectEvent.selectRow("1", rowIndex);
                selectEvent.doControlAction("67");      // Select Who's Who row
exit
                getAddressAction.addFSREvent(selectEvent);

                String response = appStackAddress.executeActions(loginEnv,
getAddressAction);

                if (response != null)
                {
```

```
                    whosWho = loginEnv.getObjectMapper().readValue(response,
P0111_W0111A_FormParent.class);

                    if
(appStackAddress.getLastAppStackResponse().checkSuccess("P0111_W0111A"))
                    {
                        selectWhosWho();
                        if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E"))
                        {
                            nav = "to_Addresses";
                        }
                    }
                }
            }
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }

        return nav;
    }

    public void selectWhosWho()
    {
        try
        {
            // Send another request to go to Alt Address
            ActionRequest selectAction = new ActionRequest();
            selectAction.setFormOID("W0111A");

            FSREvent findEvent = new FSREvent();
            findEvent.selectRow("1", 0);
            findEvent.doControlAction("148");
            selectAction.addFSREvent(findEvent);

            String response = appStackAddress.executeActions(loginEnv,
selectAction);

            if ((response != null) &&
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E")))
            {
                altAddresses = loginEnv.getObjectMapper().readValue(response,
P01111_W01111E_FormParent.class);
            }
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

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

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

Example 6–44 Save Address

```
    private String saveAddress(String formMode, int rowIndex)
    {
        String nav = "";

        try
        {
            loginEnv.getRequiredCapabilities().add(AISClientCapability.GRID);
            ActionRequest addAction = new ActionRequest();
            addAction.setFormOID("W01111E");
            FSREvent saveEvent = new FSREvent();
            GridAction gridAction = new GridAction(loginEnv);
            if (formMode.equals("A"))
            {
                GridRowInsertEvent gri = new GridRowInsertEvent();

                // Set column values
                gri.setGridColumnValue("34", newAddressRow.getAddressType());
                gri.setGridColumnValue("26", newAddressRow.getAddressLine1());
                gri.setGridColumnValue("22", newAddressRow.getCity());
                gri.setGridColumnValueDate("18", newAddressRow.getBeginDate(),
loginEnv);

                // Add grid row
                gridAction.insertGridRow("1", gri);
            }
            else
            {
                GridRowUpdateEvent gru = new GridRowUpdateEvent();

                // Set column values
                gru.setGridColumnValue("34", currentAddressRow.getAddressType());
                gru.setGridColumnValue("26", currentAddressRow.getAddressLine1());
                gru.setGridColumnValue("22", currentAddressRow.getCity());

                // Update grid row
                gridAction.updateGridRow("1", rowIndex, gru);
            }

            // Add grid action to form events.
            saveEvent.addGridAction(gridAction);
            saveEvent.doControlAction("12");        // Press OK button
            addAction.addFSREvent(saveEvent);

            String response = appStackAddress.executeActions(loginEnv, addAction);

            if (response != null)
            {
                // After save, application returns to Who's Who.  Open Alt Address
again to get updated list.
                if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A"))
                {
                    selectWhosWho();
```

```
                    if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E"))
                    {
                        nav = "__back";
                    }
                }
                else
                {
                    if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E"))
                    {
                        P01111_W01111E_FormParent tempW01111E =
loginEnv.getObjectMapper().readValue(response, P01111_W01111E_FormParent.class);

                        // Check for errors on form.
                        if (tempW01111E.getFs_P01111_W01111E().getErrors().length
> 0)
                        {
                            displayFormErrors();
                        }
                    }
                }
            }
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }

        return nav;
    }
```

Example 6–45 Delete Address

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

```
    public String deleteAddress(int rowIndex)
    {
        String nav = "";

        ActionRequest deleteAddress = new ActionRequest();
        deleteAddress.setFormOID("W01111E");
        FSREvent deleteEvent = new FSREvent();
        deleteEvent.selectRow("1", rowIndex);
        deleteEvent.doControlAction("41");      // Press Delete button
        deleteEvent.doControlAction("12");      // Press OK button
        deleteAddress.addFSREvent(deleteEvent);

        try
        {
            String response = appStackAddress.executeActions(loginEnv,
deleteAddress);

            if (response != null)
```

```
            {
                if (appStackAddress.getLastAppStackResponse().checkSuccess("P0111_
W0111A"))
                {
                    selectWhosWho();
                    if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01111_W01111E"))
                    {
                        nav = "__back";
                    }
                }
                else if
(appStackAddress.getLastAppStackResponse().checkSuccess("P01012_W01012B"))
                {
                    nav = "to_AB";
                }
            }
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }

        return nav;
    }
```

Example 6–46 Close App Stack

The following sample code shows an example of the method to close the address.

```
public void closeAppStackAddress()
    {
        try
        {
            appStackAddress.close(loginEnv);
        }
        catch (JDERestServiceException e)
        {
            System.out.println(JDERestServiceProvider.handleServiceException(e));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
```

6.9.5 ApplicationStack Methods

The following table describes the ApplicationStack methods:

Modifier and Type	Method	Description
`String`	`open(LoginEnvironment loginEnvironment, FormRequest formRequest)`	Opens a form, allowing further interactions with subsequent service calls.

Modifier and Type	Method	Description
String	executeActions(LoginEnvironment loginEnvironment, ActionRequest actionRequest)	Executes actions on an already open form.
String	close(LoginEnvironment loginEnvironment, ActionRequest actionRequest)	Closes all applications in the application stack, after executing the actions requested.
String	close ()	Closes all applications in the application stack, without doing any further actions.
ApplicationStackResponse	getLastAppStackResponse()	Returns the last response which includes the stack ID, state ID, rid and system errors.

6.9.6 ApplicationStackResponse Methods

The following table describes the ApplicationStackResponse methods:

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

6.10 Understanding Jargon Labels

The labels used in an ADF application called from EnterpriseOne should be translated into the EnterpriseOne users' language, and should follow the jargon code defined on the EnterpriseOne task used to launch them.

In EnterpriseOne, the task to launch an ADF application has a Jargon field. The value in this Jargon field will be passed to the ADF application to fetch the text for field labels. In the EnterpriseOne data dictionary, a data item may have description overrides defined based on two key fields: Language and Jargon Code.

6.10.1 Service Endpoint

The jargon service is exposed by the AIS Server at the endpoint:

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

6.10.2 Capability

The jargon capability is exposed in the default configuration as "jargon." If your ADF application requires a certain capability, you must validate that AIS provides the capability.

6.10.3 Getting Jargon Labels to use in ADF Application

To get the correct labels to display in the ADF application, you must use the Jargon Service provided by AIS to fetch the overrides for each data item displayed on your application, based on the Jargon code that was sent when the application was launched by EnterpriseOne.

The following steps explain how to get the label values and use them in your application pages.

Note: The following steps describe only one way to accomplish the desired result; it is not required to use this exact method. The only requirement is that the labels shown in your application are translated and are based on the Jargon code defined in the EnterpriseOne task.

You should not have any hard coded label text or error message text or provide a translation bundle for your application.

To get the label values and use them in your application pages:

1. In your main data control class for your ADF application/bounded task flow, create static values and a hash map to store all the labels you will be using.
2. Create a default code in case the Jargon code does not get passed in (for example, when running outside of the container).
3. Initialize the map with the default hard coded values for your field labels.

```
 private static final String DEFAULT_JARGON = "01";
    //AN8, ALPH,MLNM,ADD1,CTY1,ADDS,ADDZ,CTR
    private static final String ADDRESS_NUMBER = "Address Number";
    private static final String ALPH_NAME = "Name";
    private static final String MAIL_NAME = "Mailing Name";
    private static final String ADDDRESS = "Address";
    private static final String CITY = "City";
    private static final String STATE = "State";
    private static final String POSTAL = "Zip Code";
    private static final String COUNTRY = "Country";

    private HashMap<String, String> fieldLabels = new HashMap<String,
String>() {
        {
            put("AN8", ADDRESS_NUMBER);
            put("ALPH", ALPH_NAME);
            put("MLNM", MAIL_NAME);
            put("ADD1", ADDDRESS);
            put("CTY1", CITY);
            put("ADDS", STATE);
            put("ADDZ", POSTAL);
            put("CTR", COUNTRY);
        }
    };
```

4. Create a method to retrieve the override values from the AIS jargon service, and populate the hash map with those values.

```
    private void retriveJargonValues() {
//get the jargonCode from page flow scope
string jargon code = (string)
ADFContext.getCurrent().getPageFlowScope().get("jargonCode");
   //if no jargon, use default
        if (jargonCode == null || jargonCode != null && jargonCode.isEmpty())
{

            jargonCode = DEFAULT_JARGON;
        }
        try {
      //get login environment, populate the request with DD items, call the
jargon service
            LoginEnvironment le = E1AdfUtils.getLoginEnvironment(userBean);
            le.getRequiredCapabilities().add("AISClientCapability.JARGON);
```

```
            JargonRequest jargonReq = new JargonRequest(le, jargonCode);
            //AN8, ALPH,MLNM,ADD1,CTY1,ADDS,ADDZ,CTR
            jargonReq.addDataItem("AN8");
            jargonReq.addDataItem("ALPH");
            jargonReq.addDataItem("MLNM");
            jargonReq.addDataItem("ADD1");
            jargonReq.addDataItem("CTY1");
            jargonReq.addDataItem("ADDS");
            jargonReq.addDataItem("ADDZ");
            jargonReq.addDataItem("CTR");
            String response =
                JDERestServiceProvider.jdeRestServiceCall(le, jargonReq,
JDERestServiceProvider.POST_METHOD,

JDERestServiceProvider.JARGON_SERVICE);
            jargonResp = le.getObjectMapper().readValue(response,
JargonResponse.class);

         //replace the default values in the hash map with the jargon values
            if (jargonResp != null) {
                if (jargonResp.getRequestedItems() != null &&
jargonResp.getRequestedItems().size() > 0) {
                    for (JargonResponseItem item :
jargonResp.getRequestedItems()) {

                        fieldLabels.put(item.getSzDict().trim().toUpperCase(),
item.getRowDescription());

                    }
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

5. Call the new method inside your initialize method of your DC class.

```
public void initialize() {

        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, JAS_SERVER);
        } else {
            //initialize application for about properties, only do this when
running in the container
            E1AdfUtils.intializeAppInstance("/com/oracle/e1/A01012/");

        }
        retriveJargonValues();
        retrievePOValues();
    }
```

6. Create getters for each of the labels you have fetched. These will be exposed as elements by the data control so you can place the values on the page.

```
    public String getAn8Label() {
        return fieldLabels.get("AN8");
    }
```

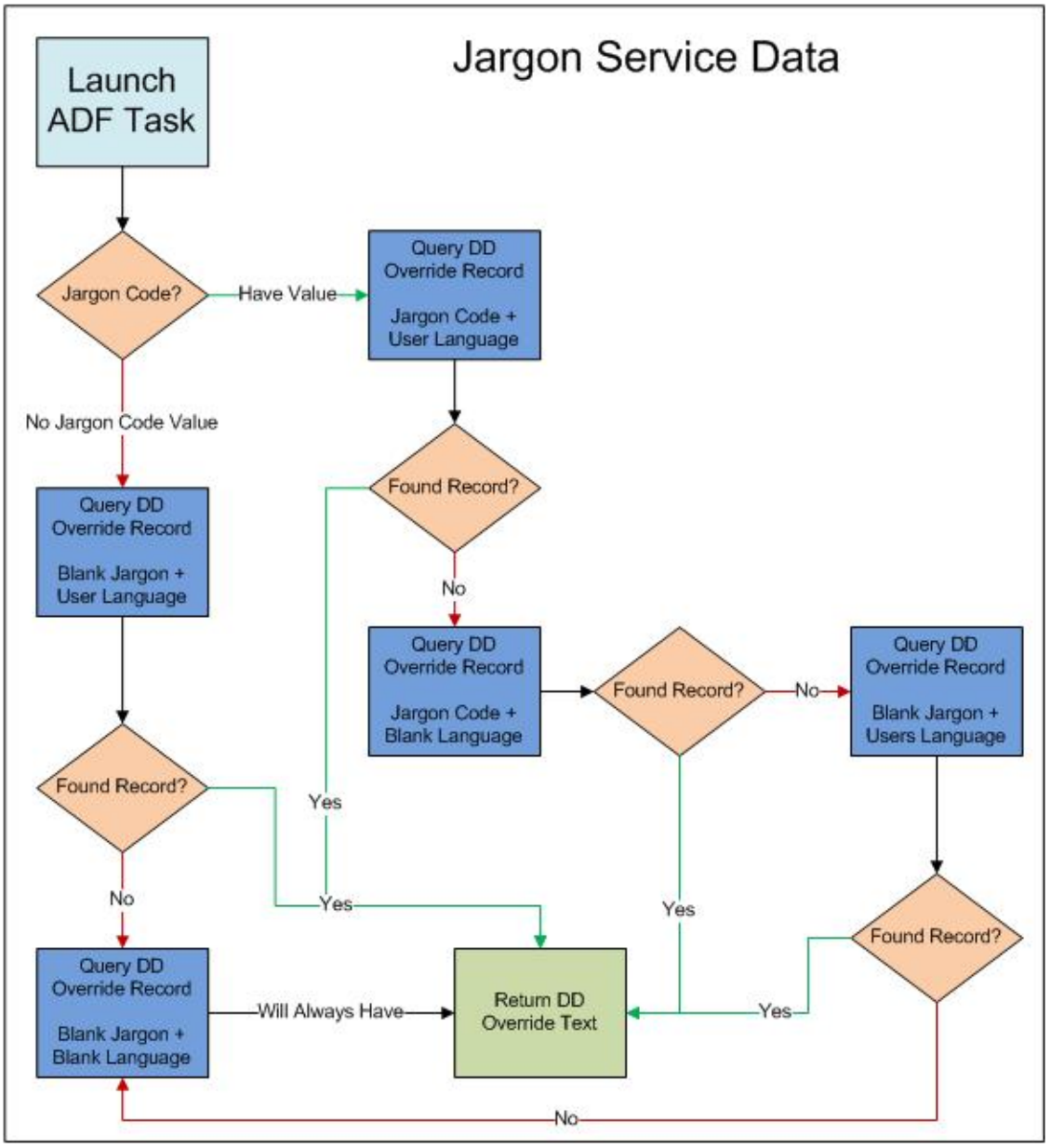
```
public String getAlphLabel() {
    return fieldLabels.get("ALPH");
}

public String getMlnmLabel() {
    return fieldLabels.get("MLNM");
}

public String getAdd1Label() {
    return fieldLabels.get("ADD1");
}

public String getCty1Label() {
    return fieldLabels.get("CTY1");
}

public String getAddsLabel() {
    return fieldLabels.get("ADDS");
}

public String getAddzLabel() {
    return fieldLabels.get("ADDZ");
}

public String getCtrLabel() {
    return fieldLabels.get("CTR");
}
```

7. On the page, drag the elements from the data control to the page, to create a data binding to the element and make the label value equal to the binding element input value.

```
        <af:panelLabelAndMessage label="#{bindings.an8Label.inputValue}"
id="plam23">
                                                <af:outputText
value="#{bindings.selectedAddressNumber.inputValue}"
                                                                    id="ot13"
inlineStyle="padding:5.0px;"/>
                                            </af:panelLabelAndMessage>
```

6.10.4 Order of Fetched Records

The following graphic illustrates the order in which the records are fetched when the Jargon Service runs, based on Jargon code passed in, users Language preference, and records that exist in the overrides table.



6.10.5 JSON Example of a Jargon Service Request

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

6.10.5.1 Open Application: Request and Response

Example 6–47 JSON Request - Jargon Labels

```
 {
    "token":
"044lVWt6GoyepCAyopvq9Sgx5Pb+LeWjhG505XntrmfJFw=MDE5MDEwNDMwMDM3NjU2MTM5OTExNDcyNW
phdmFjbGllbnQxNDI0ODczNjM0NjQ0",
    "deviceName": "javaclient",
    "defaultSystemCode": "01",
    "requestedDataItems": [
        {
```

```
            "szDict": "AN8",
            "systemCode": "01"
        },
        {
            "szDict": "ALPH",
            "systemCode": null
        },
        {
            "szDict": "MCU",
            "systemCode": "40"
        }
    ]
}
```

Example 6–48 JSON Response - Retrieve Jargon Labels

```
{
    "requestedItems": [
        {
            "szDict": "AN8",
            "systemCode": " ",
            "language": "  ",
            "colTitle": "Address             \nNumber",
            "rowDescription": "Address Number"
        },
        {
            "szDict": "ALPH",
            "systemCode": " ",
            "language": "  ",
            "colTitle": "Alpha               \nName",
            "rowDescription": "Alpha Name"
        },
        {
            "szDict": "MCU",
            "systemCode": "40",
            "language": "  ",
            "colTitle": "Branch              \nPlant",
            "rowDescription": "Branch/Plant"
        }
    ]
}
```

6.11 Understanding Query Service

A single query object may be included in a form service request. A query object contains some top level parameters, including the match type and one or more condition objects. Each condition object contains a control id, and operator, and one or more query value objects. A query value has a content field, and a special value id.

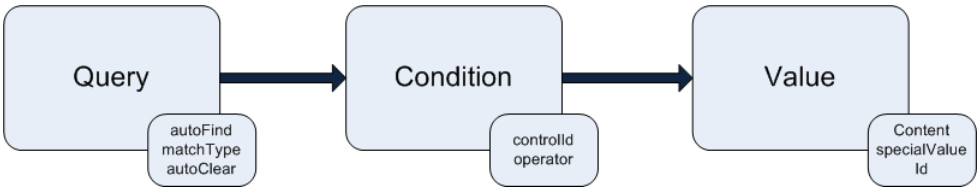


Table 6–1

Parameter	Description	Values
Query		
autoFind	Automatically click Find on the form to populate the grid records. You do not need to use event to press Find if you use autoFind.	true, false
matchType	If you want the records to match all (AND) or any (OR) of the conditions specified.	MATCH_ALL, MATCH_ANY
autoClear	If you want to clear all other fields on the form, for example, the default filter fields.	true, false
Condition		
controlId	This condition is the control id to which the condition is applied. This is the field that you add to the query from the form when using the web client to create a query. It is either a filter field, or a grid column that is associated with the business view.	"28", "1[34]"
operator	This condition is the comparison operation to use with the query.	For all types: BETWEEN, LIST, EQUAL, NOT_ EQUAL, LESS, LESS_EQUAL, GREATER, GREATER_EQUAL For strings: STR_START_WITH, STR_END_WITH, STR_CONTAIN, STR_BLANK, STR_ NOT_BLANK
content	This is either a literal value, which will be used in the comparison operation. Or, it relates to a special value id.	"23", "Joe", "2"
specialValueId	This is a special value, mostly for dates that may be today, or calculated dates from today. For calculated dates, the content field is used in the calculation.	LITERAL, TODAY, TODAY_;LUS_DAY, TODAY_MINUS_DAY, TODAY_PLUS_ MONTH, TODAY_MINUS_MONTH, TODAY_PLUS_YEAR, tODAY_MINUS_ YEAR

Example 6–49 Query - JSON Example

```
public void executeQuery()
{
    if (AISClientCapability.isCapabilityAvailable(loginEnv,
AISClientCapability.QUERY))
    {
        FormRequest formRequest = new FormRequest(loginEnv);

formRequest.setReturnControlIDs("350|360|41[116,118,122,123,124,125,129,130,131,13
2]");
        formRequest.setFormName("P42101_W42101C");
        formRequest.setVersion("ZJDE0001");
        formRequest.setFormServiceAction("R");
        formRequest.setMaxPageSize("100");
```

```
        try
        {
            Query query = new Query(loginEnv);
            query.setAutoFind(true);                // Automatically press find
button (don't need to use form action to find).
            query.setMatchType(Query.MATCH_ALL);    // Perform an AND operation on
all of the conditions.
            query.setAutoClear(false);              // Clear any existing filter
values on the form.

            // Find all order lines where line number is 1.000,
            NumberCondition numCon = query.addNumberCondition("41[129]",
NumericOperator.EQUAL());
            numCon.setValue("1.000");

            // requested date is within 2 years from today,
            DateCondition dateCon = query.addDateCondition("41[116]",
DateOperator.GREATER());
            dateCon.setSpecialDateValue(DateSpecialValue.TODAY_MINUS_YEAR(), 2);

            // sold to is between 4242 and 4245,
            BetweenCondition betCon = query.addBetweenCondition("41[125]");
            betCon.setValues("4242", "4245");

            // and company is in list (00001, 00200).
            ListCondition listCon = query.addListCondition("360");
            listCon.addValue("00001");
            listCon.addValue("00200");

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

            // Execute form request.
            String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,

formRequest,

JDERestServiceProvider.POST_METHOD,

JDERestServiceProvider.FORM_SERVICE_URI);

            formParent = loginEnv.getObjectMapper().readValue(response, P42101_
W42101C_FormParent.class);

            if (formParent != null)
            {
                // Process form service response.
            }
        }
        catch (JDERestServiceException e)
        {
            JDERestServiceProvider.handleServiceException(e);
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
    else
    {
```

```
        System.out.println("Query capability is not available.");
    }
}
```

6.12 Understanding Log Service

Starting with EnterpriseOne Tools release 9.1.4.6, an AIS service is available for adding messages to the AIS logs so that they are visible to system administrators for troubleshooting.

The URI for this service is:

http://<AISServer>:<port>/jderest/log

Example 6–50 JSON Request with token

```
{
   "token" :
"044SgzFLFFN7O6HKpAc2qFb0fKZiV3ariuYuC7lKbv2WMY=MDIwMDA2LTI2NDA3MzE4NDcyODM0MDAyNj
NTb2FwVUkxNDM1NzkyMjM2NjM0",
   "deviceName" : "SoapUI",
   "message" : "Test Message",
   "level" : "1"
}
```

Example 6–51 Resulting message in AIS log

```
01 Jul 2015 17:12:00,381[SEVERE][JDE][AIS]AIS LOG REQUEST: --Level SEVERE
--Application: null--Application Version: null --User: jde --Device Name: SoapUI
--Log Message: Test Message
```

Example 6–52 JSON Request without token

```
{
   "deviceName" : "SoapUI",
   "message" : "Test Message - No Token",
   "level" : "1"
}
```

Example 6–53 Resulting message in AIS log

```
01 Jul 2015 17:14:13,531[SEVERE][JDE][AIS]AIS LOG REQUEST: --Level SEVERE
--Application: null--Application Version: null --User: null --Device Name: SoapUI
--Log Message: Test Message - No Token
```

Example 6–54 ADF Code to log AIS message

```
if (AISClientCapability.isCapabilityAvailable(loginEnv, AISClientCapability.LOG))
{
    AISClientLogger.log(loginEnv, "Text that will be logged",
AISClientLogger.SEVERE);
}
```

There are four possible log levels you can use:

- "SEVERE
- "WARN
- "APP
- "DEBUG

You may also send an exception along with the log request. The class of the exception will be included in the log.

AISClientLogger.log(loginEnv, "Text that will be logged", AISClientLogger.SEVERE, e);

7

EnterpriseOne ADF Container

EnterpriseOne ADF applications are composed of ADF bounded task flows packaged into ADF Library JAR files. These JAR files are not deployed as web applications and cannot run individually on an ADF Server. Instead, the packaged ADF applications are deployed in an ADF Library and run inside the EnterpriseOne ADF Container (JDEADFContainer), which is a web application that runs bounded task flows in dedicated, dynamic regions. Within the JDEADFContainer, multiple ADF applications launched from an EnterpriseOne menu run simultaneously with only one ADF application visible to the user at any one time.

To launch an 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 app 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.

This chapter contains the following topics:

7.1 EnterpriseOne Proxy Applications

Each EnterpriseOne ADF application should have an accompanying EnterpriseOne "proxy" application for security, version control, and processing options. Oracle's standard is to name both the EnterpriseOne proxy and ADF applications beginning with the letter E, like E01012.

7.1.1 Understanding Processing Options in Proxy Applications

The EnterpriseOne proxy application must have a processing option field to specify a version for every EnterpriseOne application called by the ADF application that has versions. The versions specified in the processing options can then be used when the ADF application executes form service requests in the data control methods.

7.1.2 Retrieving Proxy Application's Processing Options

You use the AIS PO Service to retrieve processing options for EnterpriseOne proxy applications and applications called by the ADF application. The JDEADFContainer will pass the proxy application's name and version to the ADF application as input

parameters, which are stored as Page Flow Scope variables. The below example first retrieves processing options for the proxy application and then retrieves processing options for Address Book based on the P01012 version specified for the proxy application.

Example 7–1 Calling AIS Processing Option Service

```
if (AISClientCapability.isCapabilityAvailable(loginEnv,
AISClientCapability.PROCESSING_OPTION))
{
   try
    {
        String appName = (String)
ADFContext.getCurrent().getPageFlowScope().get("appName");
        if ((appName == null) || (appName.length() == 0))
{
    appName = "E01012";
        }

        String appVersion = (String)
ADFContext.getCurrent().getPageFlowScope().get("appVersion");
        if ((appVersion == null) || (appVersion.length() == 0))
{
    appVersion = "ZJDE0001";
        }

        ProcessingOptionRequest e01012PORequest = new
ProcessingOptionRequest(loginEnv);
        e01012PORequest.setApplicationName(appName);
        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_
2");
            if ((p01012Version == null) || (p01012Version != null &&
p01012Version.length() == 0))
            {
                p01012Version = "ZJDE0001";
            }

            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)
    {
        processErrorException(e);
    }
    catch (JDERestServiceException e)
    {
        processErrorException(JDERestServiceProvider.handleServiceException(e));
    }
    catch (Exception e)
    {
        processErrorException(e);
    }
}
else
{
   // Processing Option cability is not supported on AIS.  Use default search
type.
    searchTypeDefault = "E";
}
```

As shown in the above example, you retrieve a processing option value from the ProcessingOptionSet object using an identifier. This identifier can be found in the data structure generated for the Processing Option Template in the Object Management Workbench. You use this data structure to find the identifier for each processing option value you need for your ADF application.

Example 7–2 Generated Data Structure for Processing Option Template

```
/**
 * TYPEDEF for Data Structure
 * T01012 : Address Book - New Processing Option
```

```
 *
 * Copyright Oracle USA
 *
 * This is a JDEdwards EnterpriseOne generated file.
 * Do not modify this file.
 * Only re-generate with the appropriate tool.
 * Generation Date : 7/9/2015
 */

#ifndef DATASTRUCTURE_T01012
#define DATASTRUCTURE_T01012

typedef struct tagDST01012
{
  JCHAR          HideTax;
  JCHAR          GoToSupplierMaster;
  JCHAR          GoToCustomerMaster;
  JCHAR          SearchTypeDefault[4];
  JCHAR          GoToCSMS;
  JCHAR          Version[11];
  JCHAR          cSelfServiceMode;
  JCHAR          cTypeCode;
  JCHAR          cPostalCodeValidate;
  JCHAR          cPostalAddressRetrieval;
  JCHAR          szCustMasterVersion_VER[11];
  JCHAR          szSuppMasterVersion_VER[11];
} DST01012 , *LPDST01012;

#define IDERRHideTax_4                          4L
#define IDERRGoToSupplierMaster_5               5L
#define IDERRGoToCustomerMaster_6               6L
#define IDERRSearchTypeDefault_7                7L
#define IDERRGoToCSMS_8                         8L
#define IDERRVersion_9                          9L
#define IDERRcSelfServiceMode_10                10L
#define IDERRcTypeCode_11                       11L
#define IDERRcPostalCodeValidate_12             12L
#define IDERRcPostalAddressRetrieval_13         13L
#define IDERRszCustMasterVersion_VER_14         14L
#define IDERRszSuppMasterVersion_VER_15         15L

#endif /* DATASTRUCTURE_T01012 */
```

7.2 E1ADFUtils Overview

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

Note: These features are only available when the EnterpriseOne ADF application launches from an EnterpriseOne Solution Explorer task and executes in the JDEADFContainer web application. If the EnterpriseOne ADF application is run on JDeveloper's Integrated WLS, the application code to utilize these features will execute without error, but the JDEADFContainer 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(E1FormInterconnect 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<E1Message> messages)	Adds a list of E1Message objects to the JDEADFContainer's information message list.
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 ADF application to display in the About dialog.

7.2.1 Launching EnterpriseOne Menu Task

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

Example 7–3 Launching EnterpriseOne Menu Task

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

7.2.2 Launching an EnterpriseOne Application

To launch an EnterpriseOne application from your 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 7–4 Launching and 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);
```

7.2.3 Displaying Error and Warning Messages

When a form service request executes on the JAS server, any errors and warnings displayed on the EnterpriseOne form are returned to the ADF application in the JSON response. You can retrieve these messages from the response object and display them to the user using the JDEADFContainer'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 JDEADFContainer's exception lists.
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 7–5 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 7–6 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 JDEADFContainer's message popup will only display when the ADF application is running inside the container. In order to display messages when the application is running on JDeveloper's Integrated WLS, you can conditionalize your error reporting code to use the Java Server Faces messaging feature, as shown in the below example:

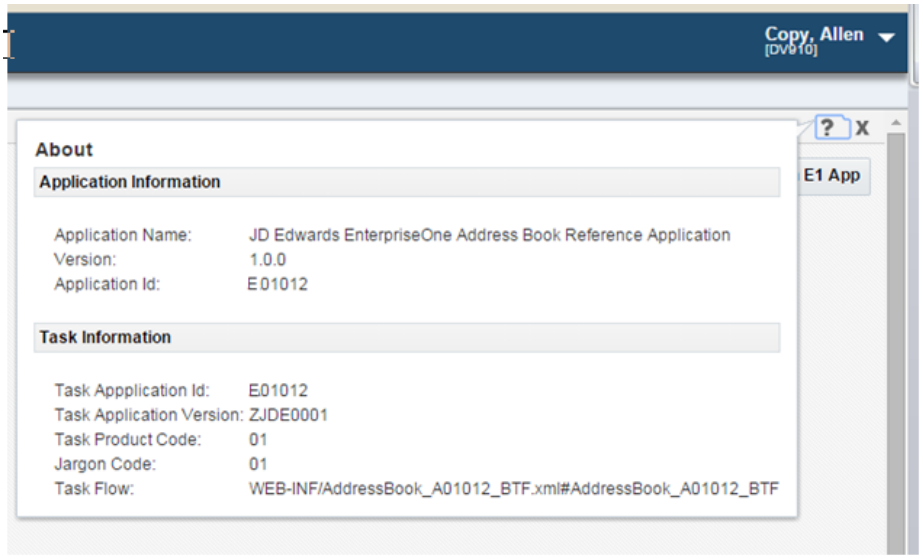
Example 7–7 Display Error Messages When Running Locally on JDeveloper's Integrated WLS

```
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 Section 4.1, "Creating a Connection to AIS Server" to review the code that first determines whether the ADF application is running locally or in the JDEADFContainer and then sets the runningInJDEADFContainer boolean accordingly.

7.2.4 Supporting About Information

The EnterpriseOne container provides an About button that enables you to display information about the 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 setup 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:

Note: The example application shown above is named E01012. "E" is the naming convention Oracle recommends you use for your ADF applications.

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

A

Creating a Sample ADF Application

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

This appendix contains the following topics:

A.1 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.

The AIS Client Class Generator extension for JDeveloper should already be installed. See Chapter 3, "Getting Started".

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

A.1.1 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 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 ensure that you understand the Generator and how to create the JSON file with REST testing tool. The AISCGE 12c install is included with the Application Interface Services Client Java API 1.2.1 download from Update Center.

A.2 Creating the Sample 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.

A.2.1 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 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-2.2.4.jar
   - jackson-core-2.2.4.jar
   - jackson-databind-2.2.4.jar
   - E1UserSession.jar

A.2.2 Creating a New ADF Application

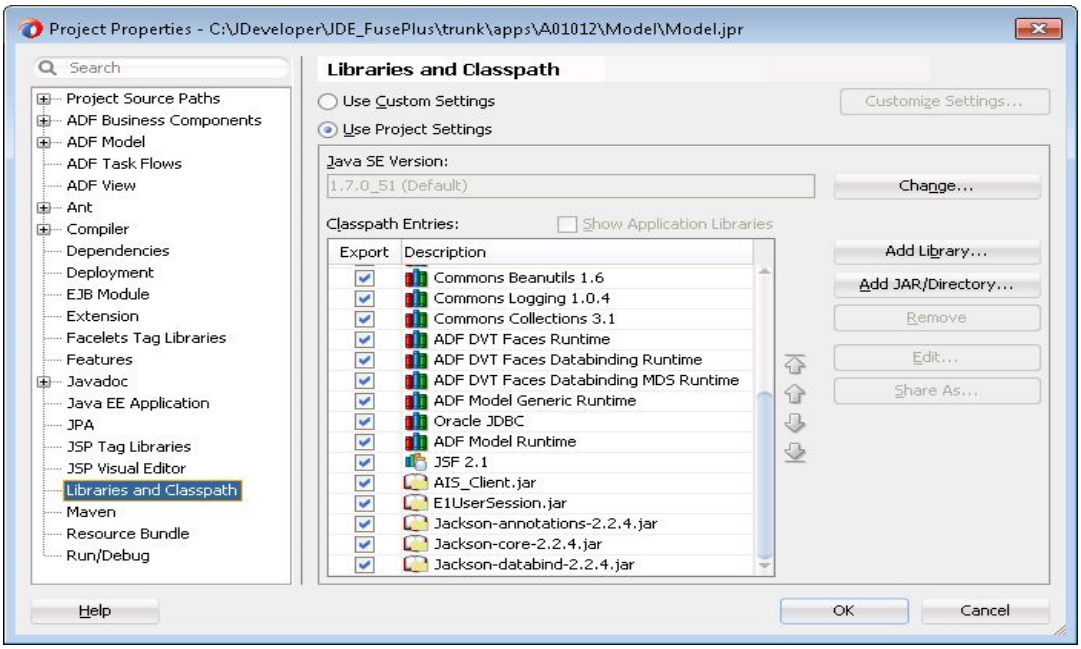
This section describes how to create a new ADF application.

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 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-2.2.4.jar
     - jackson-core-2.2.4.jar
     - jackson-databind-2.2.4.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 the **Add Library** button again.
10. Select **ADF Faces Runtime 11 Classpath** entry and then Click OK.
11. Click **OK** to save the changes.
12. In the Projects panel on the left, delete the **WEB-INF folder** that has been added to the Model project.
13. **Save** changes made to the Model project.

A.3 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. You should also be able to use the AIS Client Class Generator.

A.3.1 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 set preferences in your JDeveloper for AIS class generator:

1. From the **Tools** menu, select **Preferences**.
2. From the **Preferences panel** click **AIS Class Generator**.

If you do not see the AIS Class Generator form, click the Load Extension button to refresh the form.

3. See Section 5.2, "Configuring the AIS Client Class Generator" for information on what to enter in the fields that display.

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 Class Generator.**
3. Enter the following inputs:

   Generate W01012B Class

   - 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>
   - Click Generate.

   Generate the W01012A Class

   - Application Name: P01012
   - Form Name: W01012A
   - Version: ZJDE0001
   - FormServiceAction: R
   - FindOnEntry: <checked>
   - DemoMode: <checked>
   - Preview JSON: <checked>
   - Keep JSON: <checked>
   - Return Control IDs: [blank]

     Refresh the Model project to see the newly generated classes available for consumption.

A.3.2 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:

Example A–1

```
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:

Example A–2

```
    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:

Example A–3

```
    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.

A.3.3 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

Example A–4

```
    // 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:

Example A–5

```
    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
        {
            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(le,
e01012PORequest, JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.PO_
SERVICE);
            ProcessingOptionsSet e01012POSet =  loginEnv
.getObjectMapper().readValue(response, ProcessingOptionsSet.class);

            if (e01012POSet != null)
            {
                p01012Version = (String)
e01012POSet.getOptionValue("szP01012Version_2");
                if (p01012Version == null)
                {
                    p01012Version = "ZJDE0001";
                }
```

```
                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 launchE1App()
    {
        E1FormInterconnect form = new E1FormInterconnect();
        form.setApplication("P01012");
```

```
        form.setForm("W01012B");
        form.setVersion("ZJDE0001");
        form.addFormInterconnectValue("2", "C");
        E1AdfUtils.launchE1Application(form);
    }

    public void getAddressBookList()
    {
        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());

                // 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.
                ADFContext adfContext = ADFContext.getCurrent();
                ELContext elContext = adfContext.getELContext();
                ExpressionFactory expFactory = adfContext.getExpressionFactory();
                MethodExpression method =
expFactory.createMethodExpression(elContext,
"#{addressBookListenerBean.resetListViewSelectedItem}", Object.class, new Class[]
{});
                method.invoke(elContext, null);

                if (addressBook.getFs_P01012_
W01012B().getData().getGridData().getRowset().size() > 0)
                {

ADFContext.getCurrent().getPageFlowScope().put("selectedRowKey", "0");

                    processSelectedAddress(0);
                }
                else
                {
                    ADFContext.getCurrent().getPageFlowScope().put("showDetail",
"false");
                }
            }
            catch (JDERestServiceException e)
            {

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


    public void addressSelected(SelectionEvent selectionEvent)
    {
        System.out.println("addressSelected called by bean " + this.hashCode());

        RichListView listView = (RichListView) selectionEvent.getSource();

        // Access the rowset used by the list view.
        CollectionModel model = (CollectionModel) listView.getValue();
        JUCtrlHierBinding binding = (JUCtrlHierBinding) model.getWrappedData();

        // Retrieve index of selected list item row.
        RowKeySetTreeImpl selectedRow = (RowKeySetTreeImpl)
listView.getSelectedRowKeys();
        List nextObj = (List) selectedRow.iterator().next();
        JUCtrlHierNodeBinding node = binding.findNodeByKeyPath(nextObj);
        int rowIndex = (Integer) node.getRowKey().getKeyValues()[0];

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

```
        processSelectedAddress(rowIndex);
    }


    private void processSelectedAddress(int rowIndex)
    {
        System.out.println("processSelected called by bean " + this.hashCode());

        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");
        }

        getAddressBookDetail(selectedAddressNumber);
    }

    public void getAddressBookDetail(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();
                }

                BindingContainer bindings =
BindingContext.getCurrent().getCurrentBindingsEntry();
                OperationBinding abDetailIterator =
bindings.getOperationBinding("Execute");
                if (abDetailIterator != null)
                {
                    abDetailIterator.execute();
                }
            }
            catch (JDERestServiceException e)
            {

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

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

    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
        {
            FacesContext.getCurrentInstance().addMessage("E1ADFApp", new
FacesMessage(FacesMessage.SEVERITY_ERROR, 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.

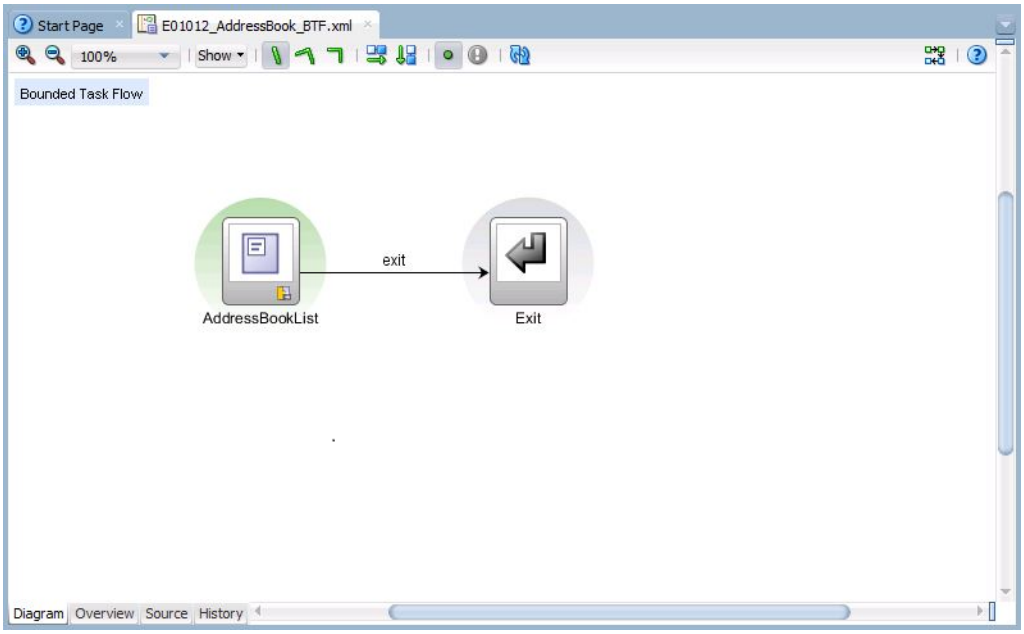
A.3.4 Creating a Task Flow

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, complete the following fields:

   - **Name**: E01012_AddressBook_BTF

   Verify that Create as Bounded Task Flow and Create with Page Fragments options are both selected.
4. From the **Components** palette, drag a **View** component onto the E01012_ AddressBook_BTF.xml diagram.
5. Name the view **AddressBookList**.
6. Drag a **Task Flow Return** component onto the E01012_AddressBook_BTF.xml diagram.
7. 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.
8. Set the action string for the flow arrow to exit.
9. Use the following graphic as a guide for setting up the bounded task flow diagram:



Address Book Bounded Task Flow

****

10. 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

11. 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.
12. Select the **Parameters** tab and add the following Input Parameter Definitions:

Table A–1

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

A.3.5 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. From the **Data Controls** panel on the left, expand the **AddressBookDC** node and drag the getAddressBookList() method onto the page fragment after the Spacer component.

   You will be prompted to specify the type of component to add.

8. Select **ADF Button** and change the text attribute to **Search**.

9. Insert a **Spacer** component after the **ADF Button** and set its width attribute to 8px.

10. Insert a **Button** component after the **Spacer**, set its action attribute to exit, and then set its text attribute to Close.

11. 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.

12. Select **Table/List View > ADF List View** and then click **Next and Finish** to add the component.
13. Delete the **separator** facet inside of the Panel Group Layout.
14. Select the **Bindings** tab at the bottom of the panel.
15. In the **Bindings** list on the left, highlight the **rowset** binding and click the pencil icon to edit the binding.
16. On the **Edit Tree Binding** form, click the green **'+'** icon and select **mnAddressNumber_19**.
17. 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.
18. Click the **OK** button to save the binding changes.
19. In the **Bindings** list, click the green '**+**' icon to add a new binding.
20. On the Insert Item form, select **action** from the list and click the **OK** button.
21. On the **Create Action Binding** form, expand the **AddressBookDC** node and select the **Execute** operation under addressBookDetail.
22. Click the **OK** button to add the binding.
23. Click the green '**+**' icon again to add a second binding.
24. On the Insert Item form, select **methodAction** from the list and click the **OK** button.
25. On the Create Action Binding form, expand the **AddressBookDC** node and select the **addressSelected(SelectionEvent) method**. Click the **OK** button to add the binding.
26. Click **Save All** from the toolbar.
27. Select the **Source** tab at the bottom of the panel.
28. 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}.
29. Set focus on the **List View** component and then select the **Edit** option next to the SelectionListener property in the Properties panel.
30. On the Edit Property 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)
31. Click the **OK** button to create the class and the managed bean declaration in the E01012_AddressBook_BTF.xml.
32. Select the **New** button next to the Method drop-down list and set the following field:

    - **Method Name**: addressRecordSelected
33. Click the **OK** button to create the method and click **OK** again to close the Edit Property form.

34. Set the **List View binding** attribute to #{addressBookListenerBean.listView} and the selection attribute to single.
35. Set focus on the **List View component** and then select the **Edit** option next to the PartialTriggers property in the Properties panel.
36. 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.
37. Click the **OK** button to set the attribute value.
38. In the Projects panel, expand the **ViewController and Application Sources** nodes and open the com.oracle.e1.E01012.view.AddressBookListenerBean.java class.
39. Add the code shown below to AddressBookListenerBean.java.

Example A–6

```
    private RichPanelFormLayout form;
    private RichListView listView;

    public AddressBookListenerBean()
    {
    }

    public void addressRecordSelected(SelectionEvent event)
    {
        BindingContainer bc =
BindingContext.getCurrent().getCurrentBindingsEntry();
        OperationBinding addressSelected =
bc.getOperationBinding("addressSelected");
        if (addressSelected != null)
        {
            Map args = addressSelected.getParamsMap();
            args.put("selectionEvent", event);
            addressSelected.execute();

            // Refresh detail panel.
            AdfFacesContext.getCurrentInstance().addPartialTarget(form);
        }
    }

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

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

    public RichPanelFormLayout getForm()
    {
        return form;
    }

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

```
public RichListView getListView()
{
    return listView;
}
```

40. Save changes to AddressBookListenerBean and build the class.

A.3.6 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. You will be prompted to specify the type of component to add. Select **ADF Form**.
5. 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

6. Click the **OK** button to add the component.
7. Set focus on the **Panel Form Layout component** and then select the **Edit** option next to the PartialTriggers property in the Properties panel.
8. 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.
9. Set the following attribute:

   binding='#{addressBookListenerBean.form}'

10. Set the Alta skin following the instructions in Section A.4.2, "Skinning".
11. The AddressBookList.jsff page fragment should look like the following:

Example A–7

```
<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" inlineStyle="margin:
10px;" dimensionsFrom="parent">
        <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"
binding="#{addressBookListenerBean.form}" partialTriggers="b1">
                        <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="it11">
                            <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="it2">
                            <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="it3">
                            <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="it4">
                            <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="it5">
                            <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="it6">
                            <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="it7">
                            <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="it8">
                            <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="it9">
                            <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="it10">
                            <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" binding="#{addressBookListenerBean.listView}"
```

```
                                    emptyText="#{bindings.rowset.viewable ? 'No data
to display.' : 'Access Denied.'}"
                                    fetchSize="#{bindings.rowset.rangeSize}" id="lv2"
selection="single"

selectionListener="#{addressBookListenerBean.addressRecordSelected}"
partialTriggers="::b1">
                        <af:listItem id="li2">
                            <af:panelGroupLayout layout="horizontal" id="pgl3">
                                <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 layout="horizontal" id="pgl1">
                        <af:inputText label="Name"
value="#{pageFlowScope.alphaNameFilter}" id="it1" columns="22"
maximumLength="25"/>
                        <af:spacer width="12px" id="s1"/>
                        <af:button
actionListener="#{bindings.getAddressBookList.execute}" text="Search"

disabled="#{!bindings.getAddressBookList.enabled}" id="b1"/>
                        <af:spacer width="8px" id="s2"/>
                        <af:button text="Close" action="exit" id="b2"/>
                    </af:panelGroupLayout>
                </f:facet>
            </af:panelStretchLayout>
        </f:facet>
        <f:facet name="end"/>
        <f:facet name="top"></f:facet>
    </af:panelStretchLayout>
</ui:composition>
```

A.3.7 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:

Example A–8

```
<f:view xmlns:f="http://java.sun.com/jsf/core"
xmlns:af="http://xmlns.oracle.com/adf/faces/rich">
```

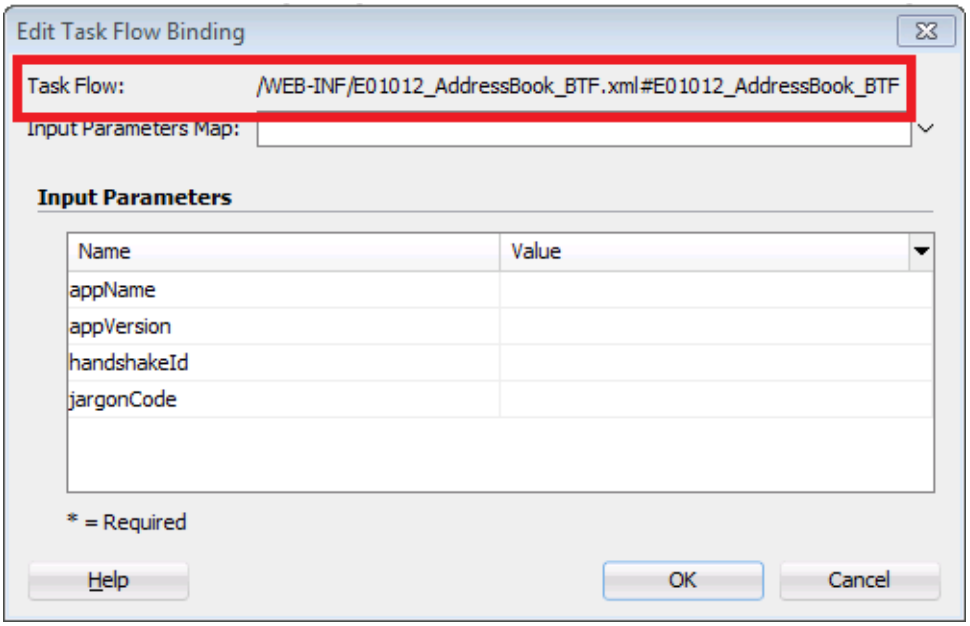
```
    <af:document title="E01012 Address Book" id="d1">
        <af:form id="f1">
            <af:region value="#{bindings.E01012_AddressBook_BTF1.regionModel}"
id="r1"/>
        </af:form>
    </af:document>
</f:view>
```

A.3.8 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 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:



This graphic is described in surrounding text.

****

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

A.3.9 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 WLS domain, JDeveloper will prompt you for an admin id and password before starting the Integrated WLS.

A.3.10 Creating ADF Library JAR File

Once your 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 section *Installing EnterpriseOne ADF Applications* in the JD Edwards EnterpriseOne Application Developer Framework (ADF) Configuration Configuration Guide for instructions on building and deploying this JAR file within the EnterpriseOne ADF library WAR file.

A.4 Other Considerations

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

A.4.1 Translations

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. The Use Jargon for Labels.docx contains instructions for using Jargon. This is the recommended approach when you are using an existing EnterpriseOne application.
2. The second approach is recommended if you are creating a new EnterpriseOne application to support your 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.

A.4.2 Skinning

When an ADF application is created, JDeveloper 12c assigns Skyros as the default skin. However, when the ADF application launches from an E1 menu and runs inside the JDEADFContainer, the ADF application is rendered using the Alta skin instead of Skyros. In order to use the Alta skin when running the test page locally on JDeveloper's Integrated WLS, 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 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.

A.5 Steps to Perform for a New ADF Application

This section provides a list of steps to help you create a new ADF application. It also provides links to sections in this guide that discusses in more detail how those steps are executed.

1. Create a **new ADF web application** in JDeveloper.

   See Appendix A.2.2, "Creating a New ADF Application"

2. Add the "**about.properties**" file.

   See Appendix 7.2.4, "Supporting About Information".

3. Change the default skin from Skyros to Alta.

   See Section A.4.2, "Skinning".

4. Create the **EnterpriseOne proxy application** and version.

   See Section 7.1.1, "Understanding Processing Options in Proxy Applications".

5. Generate **form service classes** using the AIS Client Class Generator extension for JDeveloper.

   See Appendix A.3.1, "Generating Form Service Foundation Classes for the Data Model".

6. Create the **Data Control class.**

   SeeAppendix A.3.2, "Creating the Data Control Class" and Appendix A.3.3, "Populating Address Book List and Creating the Data Control".

7. **Retrieve the EnterpriseOne proxy** application's processing option values.

   See Section 7.1.2, "Retrieving Proxy Application's Processing Options"and Example A–5.

8. **Retrieve Jargon text** from EnterpriseOne for component labels in ADF application.

   See Section 6.10, "Understanding Jargon Labels".

9. **Perform AIS service requests** to retrieve data from or insert/update data to EnterpriseOne.

   See Chapter 6, "Executing AIS Calls for Retrieving Data", and Example A–5 in Appendix A.

10. **Create a bounded task flow.**

See Section A.3.4, "Creating a Task Flow"

11. **Create view page fragments** and bind the Data Control to ADF UI components.

    See Section A.3.5, "Creating Address Book Search Panel", and Section A.3.6, "Creating Address Book Detail Panel".

12. **Create and run test page** for ADF application.

    See Section A.3.7, "Creating the ADF Test Page", and Section A.3.9, "Running the Test Page".

13. **Find the ADF Bounded Task Flow ID.**

    See Section A.3.8, "Finding the ADF Bounded Task Flow ID".

14. **Package ADF bounded task flow** into ADF Library JAR file.

    See Section A.3.10, "Creating ADF Library JAR File".

15. **Deploy application JAR** in the ADF Library WAR on the ADF server.

    See *Building and Installing EnterpriseOne ADF Application in the JD Edwards EnterpriseOne Application Developer Framework (ADF) Configuration Configuration Guide*.

16. **Create an EnterpriseOne menu task** to launch the ADF application.

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