JD Edwards EnterpriseOne Tools

Internet of Things Orchestrator Guide

Release 9.1.5.x

E64166-04

September 2015

Provides an overview of the JD Edwards EnterpriseOne Orchestrator and describes how to design and configure orchestrations to enable the immediate, real-time transformation of raw data from third-party devices to valuable and transaction-capable information in JD Edwards EnterpriseOne.

ORACLE®

JD Edwards EnterpriseOne Tools Internet of Things Orchestrator Guide, Release 9.1.5.x

E64166-04

Contents

Preface

Welcome to the *JD Edwards EnterpriseOne Tools Internet of Things Orchestrator Guide*. This guide describes how to design and create IoT orchestrations for the JD Edwards EnterpriseOne Internet of Things (IoT) Orchestrator. The IoT Orchestrator processes orchestrations to enable the immediate, real-time transformation of raw data from devices to valuable and transaction-capable information in JD Edwards EnterpriseOne.

Audience

This guide is intended for business analysts and project managers who are responsible for analyzing and managing the design process for an Internet of Things data integration with JD Edwards EnterpriseOne. It is also intended for developers who are responsible for configuring the orchestration XML files that comprise an IoT orchestration.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

This guide references related information in the following guides:

- *JD Edwards EnterpriseOne Tools System Overview Guide*
- *JD Edwards EnterpriseOne Tools Server Manager Guide*
- *JD Edwards EnterpriseOne Tools Interoperability Guide*
- *JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
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 the JD Edwards EnterpriseOne Internet of Things (IoT) Orchestrator

This chapter contains the following topics:

- Section 1.1, "Overview"
- Section 1.2, "How It Works"
- Section 1.3, "EnterpriseOne Architecture for IoT"

1.1 Overview

Companies use devices such as sensors and beacons ("things") to monitor everything from the performance of machinery, temperatures of refrigerated units, on-time averages of commuter trains, and so forth. Capturing the data from these devices traditionally involves a complex integration using specialized hardware, expensive network connectivity, and high system integration expenditure to build the machine information into an enterprise business process. Even with a complex integration in place, an operations manager or controller still might have to manually enter the data into a spreadsheet, a software program, or an application in an ERP system. Regardless of the method used, it takes time to transfer the raw data into information that can be acted upon in a way that provides value to the business.

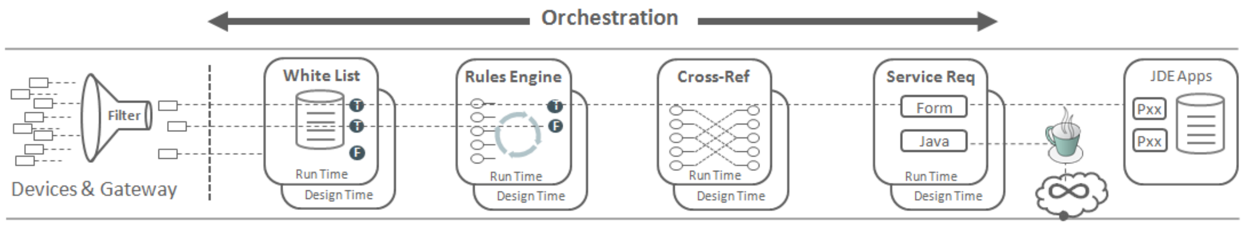
For JD Edwards EnterpriseOne, you can devise processes called orchestrations that consume raw data from disparate devices and transform the data into actionable business processes in JD Edwards EnterpriseOne. The EnterpriseOne IoT Orchestrator processes these orchestrations to enable the immediate, real-time transformation of raw data to valuable and transaction-capable information in JD Edwards EnterpriseOne. For examples, you can create orchestrations that enable EnterpriseOne to:

- Alert users to a required activity.
- Alert users to perform preventative maintenance to reduce equipment downtime.
- Provide audit data for safety compliance and security.

1.2 How It Works

The illustration in Figure 1–1 shows how the IoT Orchestrator processes data from external devices and transforms it into data that can be consumed by EnterpriseOne.

Figure 1–1 EnterpriseOne IoT Orchestration Processing



This illustration depicts how third-party devices and a gateway collect and process information from one or more devices, converts the information to a platform-independent format and communicates this information over the internet. The gateway usually deploys intelligence to filter sensor data, secure data transfer, automate software updating, run diagnostics, start or stop the device, and support other features.

The IoT Orchestrator uses the five components described in the following list to transform raw data into data that can be used by EnterpriseOne. To create an orchestration, you define each of these components in separate XML files and then place them in an IoT orchestration directory.:

- **Orchestration**. The master process that provides a unique name for the orchestration process in the IoT Orchestrator. The orchestration uses the next four components in this list to run a single orchestration instance.
- **White List**. An initial rudimentary pass/fail check of the incoming message's device signature against a predefined list of signatures. A white list provides an additional layer of security to the IoT Orchestrator security.
- **Rules Engine**. A set of conditions against which the input from the IoT devices is evaluated to produce a true or false state. Rules can be nested to produce complex evaluations. You design the rules that the engine uses to determine how to act upon the data at runtime. You can also use custom Java to define additional rules.
- **Cross-Reference**. A set of data relationships defined by the designer of the orchestration that enriches the minimal input from devices. For example, a cross reference can convert an incoming ID into an EnterpriseOne value for use in service requests.
- **Service Request**. An invocation of a JD Edwards EnterpriseOne interactive application or a Java application via a REST service call to the EnterpriseOne Application Interface Services (AIS) Server.

Also, you can use custom Java to create custom client applications (that run on the AIS Server) for viewing and working with the filtered IoT data. You can create a custom Java application to perform a specific business process or a process for storing the data in another database outside of EnterpriseOne.

1.3 EnterpriseOne Architecture for IoT

The IoT Orchestrator uses the Application Interface Services (AIS) Server as its foundation. The AIS Server is a REST services server that when configured with the EnterpriseOne HTML Server, enables access to EnterpriseOne forms and data.

For an illustration of the AIS Server architecture, see "AIS Server Architecture" in the *JD Edwards EnterpriseOne Tools System Overview Guide*.

For instruction on how to deploy the AIS Server through Server Manager, see "Create an Application Interface Services (AIS) Server as a New Managed Instance" in the *JD Edwards EnterpriseOne Tools Server Manager Guide*.

This guide contains information about Server Manager AIS Server settings that are used to manage an IoT configuration. See "Managing IoT Orchestrations".

2 Getting Started

This chapter contains the tasks that you need to perform before designing and configuring an orchestration. It contains the following topics:

- Section 2.1, "Certifications (Formerly Known as Minimum Technical Requirements)"
- Section 2.2, "Prerequisites"
- Section 2.3, "Installing and Configuring the EnterpriseOne IoT Components"
- Section 2.4, "Testing the EnterpriseOne IoT Implementation"

2.1 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`

2.2 Prerequisites

Before you can use the IoT Orchestrator, you must complete the following prerequisites:

- You must be running EnterpriseOne Tools Release 9.1.5.5.
- Deploy an Application Interface Services (AIS) Server.

  You can use an existing AIS Server or deploy a new AIS Server instance through Server Manager for the sole purpose of running IoT orchestrations. For instructions on how to deploy an AIS Server instance, see "Create an Application Interface Services (AIS) Server as a New Managed Instance" in the *JD Edwards EnterpriseOne Tools Server Manager Guide*.

  Server Manager contains specific AIS Server settings that you must configure for an IoT configuration. See "Managing IoT Orchestrations" in this guide for more information.

  If you are using an AIS Server deployed on Oracle WebLogic Server, see "Configuring Oracle WebLogic Server Domain for AIS Basic Authentication" in the

JD Edwards EnterpriseOne Application Interface Services Server for Mobile Enterprise Applications Configuration Guide for additional configuration instructions.

- Download IoT_Orchestrator_Components_1.0.0 from the Update Center on My Oracle Support: `https://support.oracle.com`. This download is also available on the Oracle Software Delivery Cloud.

  To locate the download in the Update Center, in the Type field select **EnterpriseOne IoT Orchestrator**.

  The download contains these files for developing custom Java, which is optional:

  – AIS_Client_Java_API_1.2.1. This is the API for developing custom Java.

  – OrchestratorCustomJava.jar. This file contains the definition of the interfaces for developing custom Java.

    See Chapter 5, "Creating Custom Java for Orchestrations" for more information.

- Download Orchestration_Samples_1.0.1 from the Update Center on My Oracle Support: `https://updatecenter.oracle.com/`. In the Update Center, select **EnterpriseOne IoT Orchestrator** in the Type field to locate the download.

  The download contains:

  - Sample orchestration XML files for:

    – Add Condition Based Maintenance Alert

    – Update Equipment Locations

    – Update Meter Readings

    Use these sample orchestrations to test running orchestrations in an EnterpriseOne environment with Pristine data. You can also use them as a template for creating your own orchestrations. See Appendix A, "Sample Orchestrations" for more information.

  - Data Pack for EnterpriseOne Applications 9.1.

- Download the JD Edwards EnterpriseOne IoT Orchestrator ESU for Applications [9.0] [9.1].

  To locate the ESU in the Update Center, in the Type field select **Electronic Software Update**; in the Object field enter `P952000`. The ESU bug number is 20539214 for 9.1 and 20983552 for 9.0. The ESU is also available on the Oracle Software Delivery Cloud.

2.3 Installing and Configuring the EnterpriseOne IoT Components

After downloading the components listed in the Prerequisites, follow these configuration steps.

1. Use the Change Assistant in EnterpriseOne to apply the data pack included in the Sample Orchestration download.

   See "Working with Packages" in the *JD Edwards EnterpriseOne Tools Software Updates Guide* on how to apply updates through the Change Assistant.

2. Create the orchestration directory structure on the application server machine running the AIS Server. To do so:

   a. Unzip the Orchestration_Samples_1.0.0 par file and locate the JDE_IOT_Orchestrator_XML directory:

\Orchestration_Samples_1.0.0_26_99\SampleOrchestrations\JDE_IOT_Orchestrator_XML

b. Copy the JDE_IOT_Orchestrator_XML directory and paste it to a location on the AIS Server machine.

c. Make sure the directory is read/writable by the Java virtual machine (JVM) so that the AIS Server can access it.

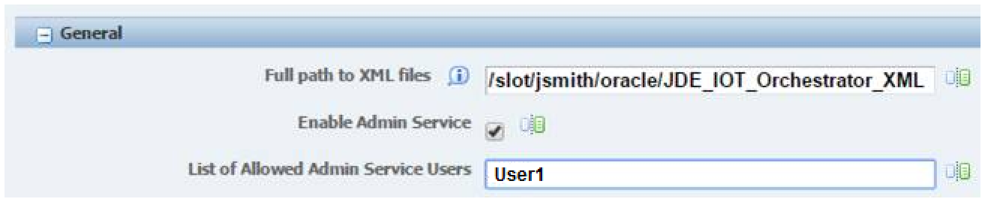
Note: As you create orchestrations, you will place your orchestration XML files in this same directory. See Understanding Orchestrations and the Orchestration Directory Structure for more information.

---

3. In Server Manager, in the General section of AIS Server Configuration group settings, enter the fully qualified path to the XML directory, for example:

   /slot/jsmith/oracle/JDE_IOT_Orchestrator_XML

   Make sure the path contains forward slashes (/). If the host is a Windows machine, replace the backslashes with forward slashes.



General

Full path to XML files: /slot/jsmith/oracle/JDE_IOT_Orchestrator_XML

Enable Admin Service: ✓

List of Allowed Admin Service Users: User1

4. Oracle recommends to select the Enable Admin Service option on the General tab as well, and define the users allowed to run the admin service.

   This enables users to run the admin service, which registers modifications to orchestration XML files to the AIS Server. See Managing IoT Orchestration Modifications Using the AIS Administration Service for more information.

5. Make sure that you set up Basic Authentication, which is required for the Orchestrator Client (the client used to test orchestrations). See "Configuring Oracle WebLogic Server Domain for HTTP Basic Authentication" in the *JD Edwards EnterpriseOne Application Interface Services Server for Mobile Enterprise Applications Configuration Guide*

6. Apply the ESU listed in the Prerequisites section to EnterpriseOne.

2.4 Testing the EnterpriseOne IoT Implementation

Use the EnterpriseOne Orchestrator Client, a standalone web application available on the AIS Server, to test your EnterpriseOne IoT implementation.

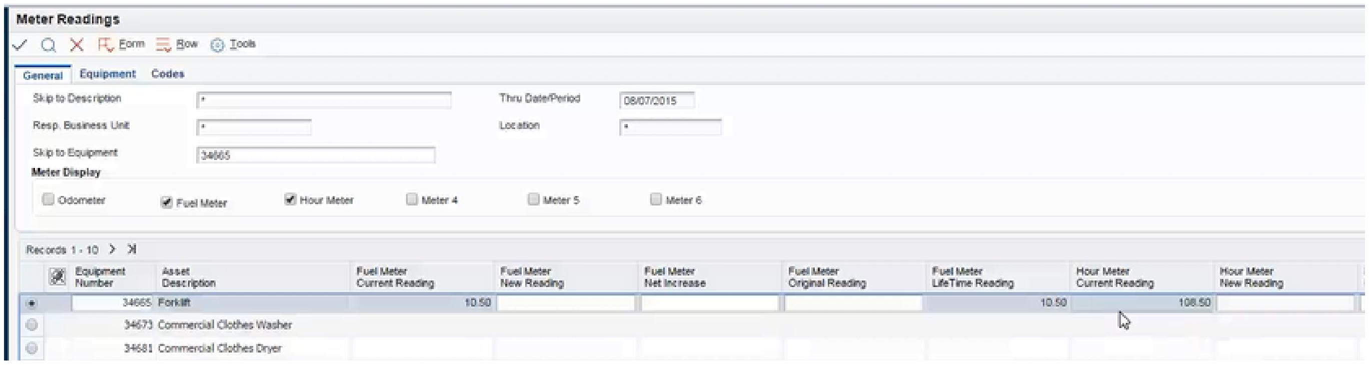
Note: The Orchestrator Client is also used for testing custom orchestrations that you create before deploying them in a production environment. See Using the EnterpriseOne Orchestrator Client to Build the Input Message and Test the Orchestration in this guide for more information.

The steps in this section use the JDE_ORCH_Sample_UpdateMeterReadings sample orchestration in the JDE_IOT_Orchestrator_XML directory to test your implementation. The JDE_ORCH_Sample_UpdateMeterReadings sample orchestration is designed to update a meter reading record in the Meter Readings program (P12120) in EnterpriseOne.

Before performing the test:

1. Access P12120 in EnterpriseOne.
2. In the Skip to Equipment field, enter `34665`.
3. Select the **Fuel Meter** and **Hour Meter** check boxes.
4. Click the **Find** button.

   In the record for equipment number 34665, notice the values in the Fuel Meter Current Reading and Hour Meter Current Reading.



Skip to Description: *
Thru Date/Period: 08/07/2015
Resp. Business Unit: *
Location: *
Skip to Equipment: 34665

Meter Display: Odometer [ ] Fuel Meter [✓] Hour Meter [✓] Meter 4 [ ] Meter 5 [ ] Meter 6 [ ]

Records 1 - 10

Equipment Number	Asset Description	Fuel Meter Current Reading	Fuel Meter New Reading	Fuel Meter Net Increase	Fuel Meter Original Reading	Fuel Meter LifeTime Reading	Hour Meter Current Reading	Hour Meter New Reading
34665	Forklift	10.50				10.50	108.50	
34673	Commercial Clothes Washer							
34681	Commercial Clothes Dryer							

Next, test the IoT implementation by running the JDE_ORCH_Sample_UpdateMeterReadings sample orchestration in the Orchestrator Client, which if successful, updates the Fuel Meter and Hour Meter with inputs sent through the sample orchestration.

To test your EnterpriseOne IoT implementation:

Caution: You must perform this test in an environment with EnterpriseOne pristine data because sample orchestrations are designed to work with pristine data only.

---

1. Access the Orchestrator Client by entering the following URL in a Web browser address bar:

   ```
   http://<ais_server>:<port>/jderest/client
   ```

2. On the Orchestrator Client Sign In screen, enter your EnterpriseOne User credentials, environment, and role. It is highly recommended that you enter an EnterpriseOne environment used for testing, not a production environment.
3. Click the **Login** button.
4. In the Orchestrator Client, in the Orchestration Name field, enter JDE_ORCH_Sample_UpdateMeterReadings.
5. In the left panel, click the **add icon** (plus symbol) to add the following name-value pair inputs in the Name and Value columns:

Name	Value
EquipmentNumber	`34665`

Name	Value
NewFuelMeterReading	`11.2`
NewHourMeterReading	`110.15`

These name-value pairs should already be defined in the orchestration as the expected input for the orchestration.

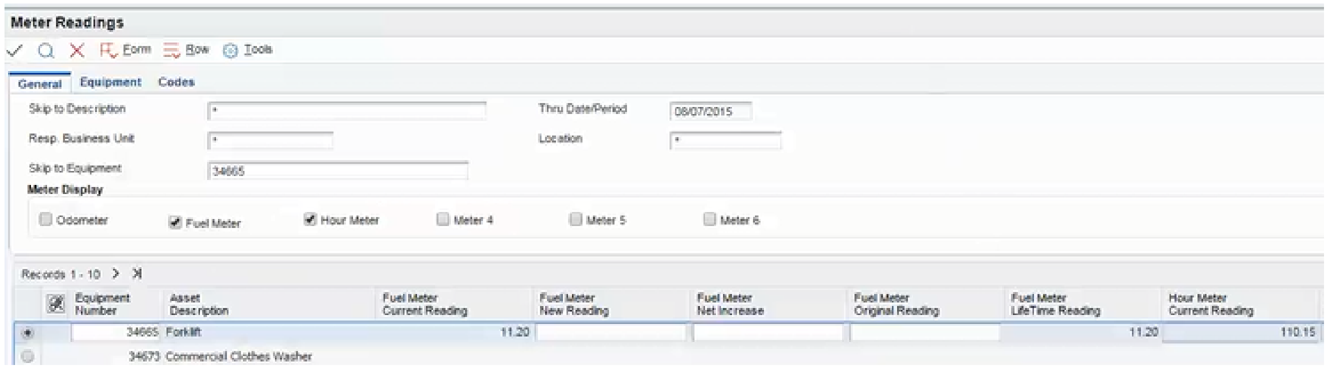
6. Click the **Run** button to test the orchestration.

   The Input area shows the input message in JSON format.

   The Output area shows the result of the orchestration. If successful, it shows the resulting forms from each form service call in JSON format. If unsuccessful, it shows an error response in JSON format. A warning may appear even if the update is successful.

To verify that the orchestration invoked the transaction in EnterpriseOne, access the Meter Readings program (P12120) and perform a search for equipment number 34665. Make sure to select the Fuel Meter and Hour Meter check boxes.

If the results in the grid show the record with the new values as shown in the following screenshot, then your EnterpriseOne IoT implementation is successful.



Skip to Description	*
Thru Date/Period	08/07/2015
Resp. Business Unit	*
Location	*
Skip to Equipment	34665

Meter Display

[ ] Odometer [✓ Fuel Meter] [✓ Hour Meter] [ ] Meter 4 [ ] Meter 5 [ ] Meter 6

Records 1 - 10

Equipment Number	Asset Description	Fuel Meter Current Reading	Fuel Meter New Reading	Fuel Meter Net Increase	Fuel Meter Original Reading	Fuel Meter LifeTime Reading	Hour Meter Current Reading
34665	Forklift	11.20				11.20	110.15
34673	Commercial Clothes Washer						

If the orchestration fails, verify that you properly followed the prerequisites and installation steps in this chapter. Also, make sure the test was performed in an environment with pristine data. In this environment, the equipment number 34665 that is referenced by the cross references in P952000 should be in the database.

3 Designing an Orchestration

This chapter contains the following topics:

3.1 Understanding the Orchestration Design Process

You might already have a business process in EnterpriseOne that involves manually entering data into EnterpriseOne from a device that collects data. Or you might use a non-EnterpriseOne system to record data from various devices. Or you might not understand how data from these devices can be used by JD Edwards EnterpriseOne applications.

Before you can create an orchestration, you need to perform an analysis to:

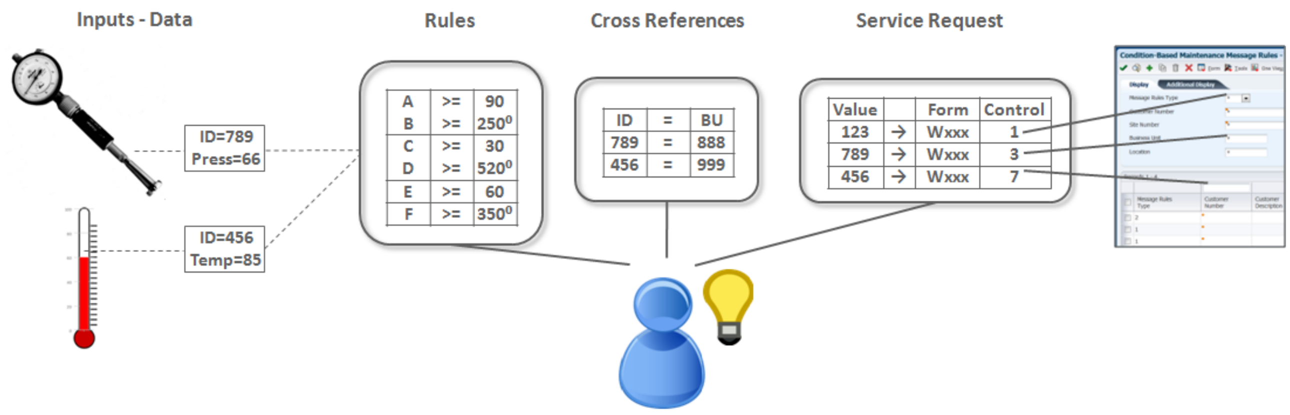
- Identify the problem and the solution.
- Identify the data that you want to collect.
- Define the rules and conditions that determine how to process the data.
- Identify the EnterpriseOne application inputs (fields and grid columns).
- Identify additional applications in which to work with the data, such as a custom Java application to perform a specific business process or a process for storing the data in another database.

You can use a simple worksheet for your analysis or you could use a storyboard, flow chart, or a combination of methods depending on the complexity of your orchestration. Use the information captured from your analysis to configure each of the XML files used in the orchestration as described in the next chapter, "Configuring an Orchestration".

Example: Company A's Orchestration Design Process

Company A used a storyboard as part of their orchestration design process to illustrate the design of a simple orchestration. Figure 3–1 shows an illustration of the overall result of Company A's design process. The remaining sections in this chapter contain additional details about each part of the design process and examples of how Company A identified the information required for the orchestration.

Figure 3–1 Example of an Orchestration Design Process



A	>=	90
B	>=	250^0
C	>=	30
D	>=	520^0
E	>=	60
F	>=	350^0

ID	=	BU
789	=	888
456	=	999

Value		Form	Control
123	→	Wxxx	1
789	→	Wxxx	3
456	→	Wxxx	7

3.2 Identifying the Problem and Solution

Begin the analysis by identifying the problem or the data gap, and then identify how you want to use the data in EnterpriseOne, or in other words, determine which EnterpriseOne business process or transaction you want to invoke.

Example: Company A's Problem and Solution

Problem

Company A currently uses sensors to detect issues in certain assets to prevent potential breakdowns. Specifically, the company uses vibration and temperature sensors on various pieces of equipment. The data read from the sensors is not tied into their EnterpriseOne system; instead, it is integrated with a third-party software program that an operations manager has to access several times a day to monitor equipment. It would be ideal if the company could eliminate the need to use a disparate software program to manually oversee the performance of its equipment.

Solution

Company A wants to design a process that uses the EnterpriseOne Condition-Based Maintenance program for monitoring. With the IoT Orchestrator, Company A can design an orchestration with rules and conditions that:

- Invoke a transaction in EnterpriseOne Condition-Based Maintenance that sends a *warning* message if vibration or temperature levels are within a certain range above normal operating levels.
- Invoke a transaction in EnterpriseOne Condition-Based Maintenance that sends an *alarm* message if vibration or temperature levels exceed a certain level.

3.3 Identifying the Data for the Orchestration

After you identify the problem and determine the EnterpriseOne applications or processes that you want to invoke, you need to identify the device data or payload to use in the orchestration. For example, a reading from a sensor might include a vibration measurement, a temperature measurement, and the date and time of the readings.

Example: Company A's Data Analysis

The "Input - Data" area in Figure 3–1 highlights the data that Company A identified for their orchestration.

3.4 Identifying the Rules for the Orchestration

Next, identify the rules and conditions to determine how to process data from the sensors.

You will use the information from this part of the analysis to configure the rules XML as described in the Section 4.3, "Configuring Rules XMLs" section in this guide.

Example: Company A's Rules Analysis

In this part of the analysis, Company A identified the following conditions:

- A vibration reading greater than or equal to 90 and a temperature greater than or equal to 250 will trigger an alarm message.
- A vibration reading greater than or equal to 30 and a temperature greater than or equal to 520 will trigger an alarm message.
- A vibration reading greater than or equal to 60 will trigger a warning message.
- A temperature reading greater than or equal to 350 will trigger a warning message.

The "Rules" area in Figure 3–1 shows the rules and conditions that Company A is using to determine which data should be processed by the orchestration.

3.5 Identifying the Cross-Reference Information for the Orchestration

This process entails identifying and mapping each piece of data to a value or data item in an EnterpriseOne form field or grid column.

Use the information from this part of the analysis to configure the cross-reference XML as described in the "Configuring Cross-Reference XMLs" section in this guide.

Also, in this phase you can decide if you want to incorporate a white list into the orchestration by configuring a white list XML as described in the "Configuring White List XMLs" section in this guide.

Example: Company A's Cross-Reference Analysis

Company A used this phase of the analysis to map the following IoT inputs to EnterpriseOne fields:

- sensor ID -> equipment number | measurement location
- equipment number -> warning recipient | alarm recipient

The "Cross-Reference" area in Figure 3–1 highlights the cross-references Company A is using in the orchestration.

3.6 Identifying the Service Request Information for the Orchestration

Next, you need to identify how the data is used to invoke a business process or transaction in EnterpriseOne. You identify the EnterpriseOne applications and inputs including the control IDs for EnterpriseOne buttons, fields, and so forth.

You can also determine if you want to use custom Java to execute a custom process or to route IoT data into another database.

Use the information gathered from this phase of the analysis to design the services request XMLs as described in the "Configuring Service Request XMLs" section in this guide.

Example: Company A's Service Request Analysis

The "Service Request" area in Figure 3–1 highlights the data Company A is using for the service request.

4 Configuring an Orchestration

This chapter describes how to take the data from analysis to implementation. It contains the following topics:

4.1 Understanding Orchestrations and the Orchestration Directory Structure

An orchestration is a process that enables the transformation of raw data from disparate devices into actionable business processes in JD Edwards EnterpriseOne. An orchestration can be comprised of four components: service requests, rules, cross-references, and white lists. You define the metadata for each component in XML files, and then place each component's XML file in the appropriate folder for that component. The IoT Orchestrator processes an orchestration using the XML files in these folders:

- **Orchestrations**. This folder contains orchestration XML files that provide the master information for an orchestration. An orchestration XML file contains one or more orchestration steps. Each orchestration step points to a service request, rule, cross-reference, or white list XML files located in the other folders.
- **ServiceRequests**. This folder contains XML files with the service request information. The AIS Server uses this information to interact with EnterpriseOne applications.

- **Rules**. This folder contains XML files with the conditions to be evaluated, such as true or false conditions that control the orchestration flow. With rules, a false outcome or true outcome can invoke further orchestration steps.
- **CrossReferences**. This folder contains XML files with cross-references that map the third-party values to EnterpriseOne values.
- **WhiteLists**. This folder contains XML files with an inclusive list of values permitted in the orchestration and terminates the orchestration process if incoming data is not recognized.

In most cases, the IoT Orchestrator enables you to use the orchestration metadata to perform AIS Server calls that invoke EnterpriseOne transactions; Java is not required. The AIS Server exposes each orchestration as an endpoint at:

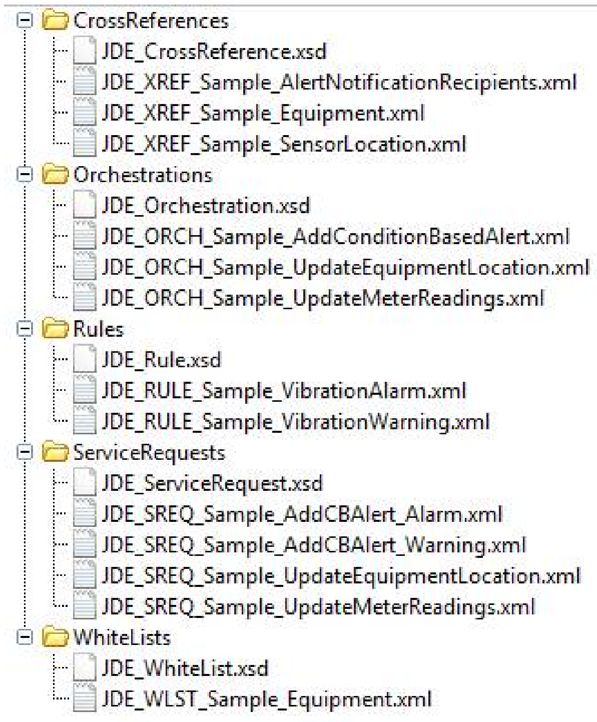
```
http://<server>:<host>/jderest/orchestrator/<orchestrationName>
```

The orchestration name is the file name of the orchestration XML file. The file name must NOT include any spaces.

The orchestration requires a properly formatted JSON message for processing.

Figure 4–1 shows an example of the folders and files in an orchestration directory:

Figure 4–1 Orchestration Directory Folders and Files



```
CrossReferences
    JDE_CrossReference.xsd
    JDE_XREF_Sample_AlertNotificationRecipients.xml
    JDE_XREF_Sample_Equipment.xml
    JDE_XREF_Sample_SensorLocation.xml
Orchestrations
    JDE_Orchestration.xsd
    JDE_ORCH_Sample_AddConditionBasedAlert.xml
    JDE_ORCH_Sample_UpdateEquipmentLocation.xml
    JDE_ORCH_Sample_UpdateMeterReadings.xml
Rules
    JDE_Rule.xsd
    JDE_RULE_Sample_VibrationAlarm.xml
    JDE_RULE_Sample_VibrationWarning.xml
ServiceRequests
    JDE_ServiceRequest.xsd
    JDE_SREQ_Sample_AddCBAlert_Alarm.xml
    JDE_SREQ_Sample_AddCBAlert_Warning.xml
    JDE_SREQ_Sample_UpdateEquipmentLocation.xml
    JDE_SREQ_Sample_UpdateMeterReadings.xml
WhiteLists
    JDE_WhiteList.xsd
    JDE_WLST_Sample_Equipment.xml
```

4.2 Configuring Service Request XMLs

This section contains the following topics:

- Section 4.2.1, "Understanding Service Request XMLs"
- Section 4.2.2, "Understanding the Service Request XML Structure"
- Section 4.2.3, "Defining the Service Request Metadata for the Orchestration"

- Section 4.2.4, "Example of a Service Request XML"

4.2.1 Understanding Service Request XMLs

Service request XMLs provide the metadata that the IoT Orchestrator uses to invoke AIS endpoints on the AIS Server. AIS endpoints provide services for interacting with EnterpriseOne applications. The IoT Orchestrator converts the metadata into Java code that automatically calls the AIS service. See "Table 1-1 Endpoint URIs for Accessing EnterpriseOne Applications and Data" in the *JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide* for a description of the AIS endpoints.

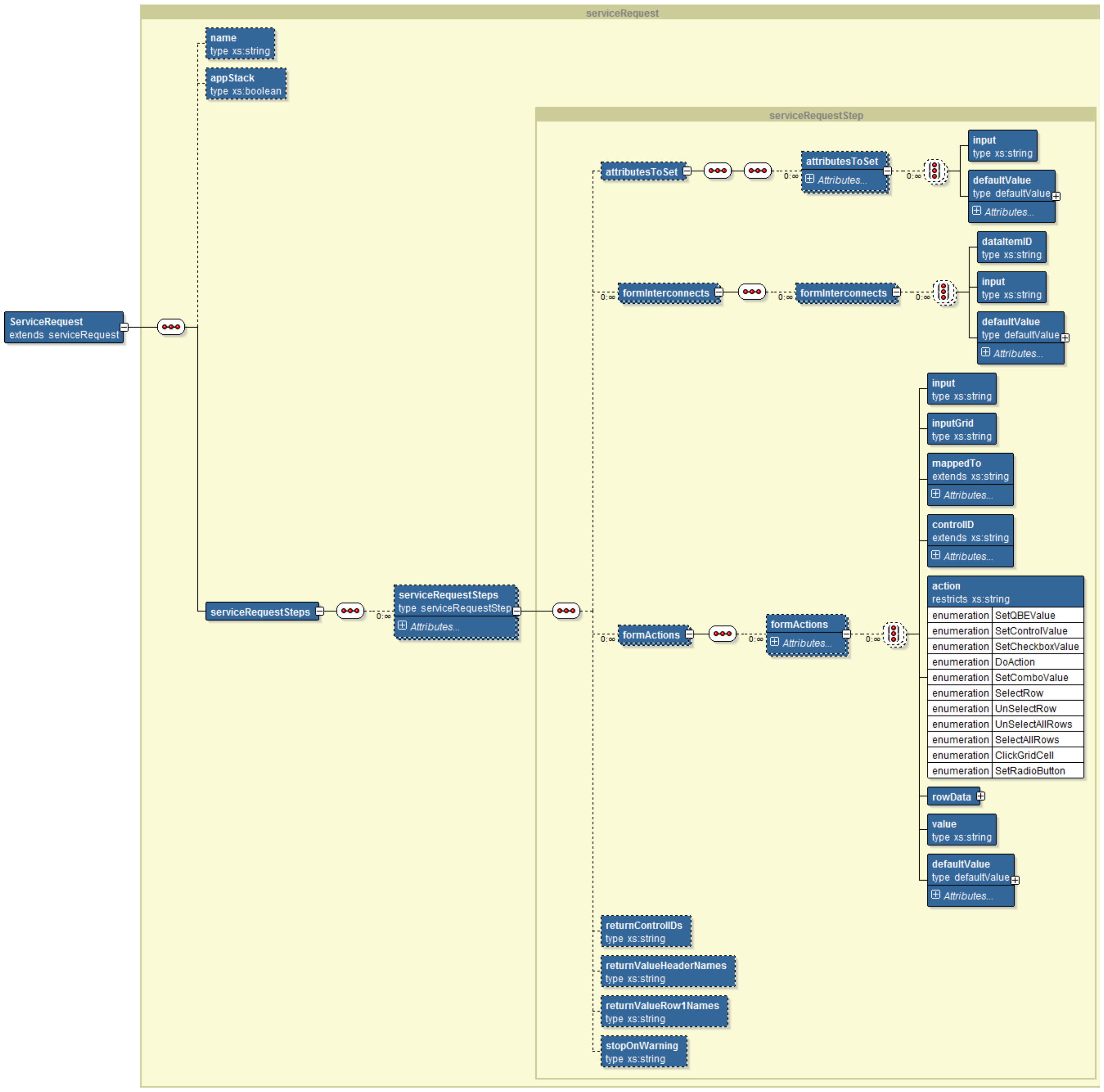
The service request XML must conform to the service request .xsd file included with the project. Do not use any spaces when naming a service request XML file.

Not all AIS services can be invoked by metadata in the service request XML. For orchestrations that require EnterpriseOne actions not supported by the metadata definitions, you can create a custom Java class. See Chapter 5, "Creating Custom Java for Orchestrations."

4.2.2 Understanding the Service Request XML Structure

The ServiceRequest element contains secondary and tertiary child elements and attributes that provide the metadata for the orchestration. Figure 4–2 shows the basic XML structure of a service request:

Figure 4–2 Service Request XML Structure


serviceRequest
name type xs:string
appStack type xs:boolean
ServiceRequest extends serviceRequest
serviceRequestSteps
serviceRequestSteps type serviceRequestStep Attributes...
serviceRequestStep
attributesToSet
attributesToSet Attributes...
input type xs:string
defaultValue type defaultValue Attributes...
formInterconnects
formInterconnects
dataItemID type xs:string
input type xs:string
defaultValue type defaultValue Attributes...
formActions
formActions Attributes...
input type xs:string
inputGrid type xs:string
mappedTo extends xs:string Attributes...
controlID extends xs:string Attributes...
action restricts xs:string
enumeration SetQBEValue
enumeration SetControlValue
enumeration SetCheckboxValue
enumeration DoAction
enumeration SetComboValue
enumeration SelectRow
enumeration UnSelectRow
enumeration UnSelectAllRows
enumeration SelectAllRows
enumeration ClickGridCell
enumeration SetRadioButton
rowData
value type xs:string
defaultValue type defaultValue Attributes...
returnControlIDs type xs:string
returnValueHeaderNames type xs:string
returnValueRow1Names type xs:string
stopOnWarning type xs:string
0..∞

4.2.3 Defining the Service Request Metadata for the Orchestration

A service request XML file contains a parent or container serviceRequest element. The following list describes the child elements and attributes that comprise a serviceRequest element:

- **appStack**. This element contains a boolean value of true or false, indicating if the service request follows the Application Stack Service pattern for calling service requests. To use this service, you must make sure that every form listed as a serviceRequestStep is included in the application stack flow as well. Also, each serviceRequestStep must match the form that was returned after the previous step.

For more information, see "Application Stack Service" in the *JD Edwards EnterpriseOne Application Interface Services Client Java API Developer's Guide*.

- **serviceRequestSteps**. This element contains one or more child **serviceRequestSteps** elements. A child **serviceRequestSteps** element provides the service request metadata for a single AIS call. It contains five attributes for the service request. See Table 4–1 for descriptions of the attributes. It also contains these child elements:
  - **formInterconnects**. An optional element that contains a list of **formInterconnects** elements. See Table 4–2 for a description of the elements and attributes in a formInterconnects element.
  - **formActions**. This element contains a list of **formActions** elements. A **formActions** element contains a *type* attribute that you set to `input`, `detail`, or `action`. The value in the *type* attribute determines the additional elements to complete for the formAction. See Table 4–3 for a description of each formActions type.
  - **returnControlIDs**. This element contains the control IDs in the form that contains the data you want returned. If left blank, all data on the form is returned.

    For example, `13|15|18|1[38,40,42]` would return the values in the form controls with ID 13, 15 and 18 and grid columns 38, 40 and 42 corresponding to the grid with ID 1.
  - **returnValueHeaderNames**. Allows the returned data to be named to be used in subsequent service request steps. For example:

    A value of OrderNumber|OrderType|Description used with the returnControlIDs element would add the contents of control ID 18 to a value called OrderNumber; the contents of control ID 15 to a value called OrderType; and the contents of control ID 18 to a value called Description. OrderNumber, OrderType and Description can be used in subsequent service request steps of the same service request.
  - **returnValueRow1Names**. Allows the returned data of the first row of a grid to be used in subsequent service request steps. For example:

Table 4–1 Description of serviceRequestSteps Attributes

Attribute	Description
type	Valid values are: ■ `formRequest` ■ `customServiceRequest` If you enter customServiceRequest, you must also define the following items: ■ className attribute. See the description below in this table. ■ **attributesToSet**. This element contains a list of **attributeToSet** elements. See Table 4–4 for a description of these elements.
appOID	The application and form name to be called, for example `P01012_W01012B`.
version	The version of the application, for example `XJDE0019`.

Table 4–1 (Cont.) Description of serviceRequestSteps Attributes

Attribute	Description
bypassFormServiceEREvent	This is used to skip the form service request event on the form. Valid values are: `true` (default) or `false`
className	Used only if type=`customServiceRequest`. This contains the package and Java class with the custom service request implementation, for example: `com.oracle.e1.rest.orchestrator.customjava.CustomServiceRequest`

Table 4–2 Description of formInterconnects Elements

Element	Description
dataItemID	The ID found in the Form Data Structure definition of the form being called.
input	The name of the input field to map into the form data structure element.
defaultValue	To be used if the input field is not specified or not found. defaultValue has one element called "value" which holds the value to be defaulted. defaultValue can have type and dateFormat attributes.

Table 4–3 Description of formActions Types

Type	Description
input	Use this type to set form controls and QBE values using these elements: ■ **input**. (optional) The name of a value coming from the input message, cross-reference or named return value from a prior form service request ■ **mappedTo**. Contains the control ID of the form control or QBE column, for a QBE column, the value will be the grid ID with the column ID in square braces, for example `1[32]`. ■ **action**. Contains either SetControlValue or SetQBEValue ■ **defaultValue**. (optional) Has a value element along with type and dateFormat attributes if needed. ■ **textSubstitution**. (optional) Contains a string element and **elements** which contain the named values for the text substitutions. For example, you might have a string with "Temp was {0} at {1}" and two elements, the first element containing a temperature input value and the second element containing a time input value.

Table 4–3 (Cont.) Description of formActions Types

Type	Description
detail	Use this type to populate rows of a grid. It contains the following elements: ■ **inputGrid**. Contains the name of the repeating input from the input message used to load multiple rows of data. ■ **mappedTo**. Contains the control ID of the grid. ■ **rowData**. Contains a list of **rowData** which represent the grid columns to populate and each **rowData** has four elements: **input**, **mappedTo**, **action**, **defaultValue**, and **textSubstitution** are the same as the input for formAction; action can contain SetGridCellValue or SetGridComboValue.
action	Use this type to invoke push buttons, set check box values, and set combo box values. It contains the following elements: ■ **controlID**. References the control that the action will affect. ■ **action**. Can contain the following values: `SetCheckBoxValue`, `DoAction`, `SetComboValue`, `SelectRow`, `UnSelectRow`, `SelectAllRows`, `UnSelectAllRows`, or `ClickGridCell`. ■ **value**. Used by `SetCheckBoxValue` and `SetComboValue`. For SetCheckBoxValue, it can be on or off. For SetComboValue, the value corresponds to the value to be selected.

Table 4–4 Description of attributesToSet Elements and Attribute

Element or Attribute	Description
name (attribute)	The value for this attribute should match the attribute name in the custom Java class.
input (element)	This element can have an input element with the name of a value from the input message.
defaultValue (element)	This element has one element, a value, and two attributes: type and dateFormat. See Table 4–10 in the Orchestration section for dateFormat definition.

4.2.4 Example of a Service Request XML

Example 4–1 shows the service request XML in the "Add Condition Based Maintenance Alert" sample orchestration.

Example 4–1 Service Request XML in the "Add Condition Based Maintenance Alert" Sample Orchestration

```
<?xml version="1.0" encoding="UTF-8" ?>
<ServiceRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="JDE_ServiceRequest.xsd">
    <serviceRequestSteps>
        <serviceRequestSteps type="formRequest" appOID="P1311_W1311B"
version="ZJDE0001">
            <formActions>
                <formActions type="input">
                    <input>EquipmentNumber</input>
                    <mappedTo>20</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <formActions type="input">
                    <input>MeasurementLocation</input>
                    <mappedTo>31</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <formActions type="input">
                    <textSubstitution>
                        <string>Temp: {0}; Vibration: {1}</string>
                        <elements>
                            <elements>TemperatureReading</elements>
                            <elements>VibrationReading</elements>
                        </elements>
                    </textSubstitution>
                    <mappedTo>29</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <!-- Alert Level -->
                <formActions type="input">
                    <mappedTo>33</mappedTo>
                    <action>SetControlValue</action>
                    <defaultValue>
                        <value>1</value>
                    </defaultValue>
                </formActions>
                <formActions type="input">
                    <input>Date</input>
                    <mappedTo>35</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <formActions type="input">
                    <input>Time</input>
                    <mappedTo>48</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <formActions type="input">
                    <input>WarningRecipient</input>
                    <mappedTo>66</mappedTo>
                    <action>SetControlValue</action>
                </formActions>
                <!-- Automated Response Type -->
                <formActions type="input">
                    <mappedTo>40</mappedTo>
                    <action>SetControlValue</action>
                    <defaultValue>
                        <value>1</value>
                    </defaultValue>
                </formActions>
                <formActions type="action">
                    <controlID>11</controlID>
                    <action>DoAction</action>
                </formActions>
            </formActions>
        </serviceRequestSteps>
    </serviceRequestSteps>
</ServiceRequest>
```

4.3 Configuring Rules XMLs

This section contains the following topics:

- Section 4.3.1, "Understanding Rule XMLs"
- Section 4.3.2, "Understanding the Rule XML Structure"
- Section 4.3.3, "Defining the Rule Metadata for the Orchestration"
- Section 4.3.4, "Example of a Rule XML"

4.3.1 Understanding Rule XMLs

Rule XMLs contain conditional logic that the IoT Orchestrator uses to evaluate conditions, such as true or false conditions that determine how the orchestration processes the incoming data. You can define a rule with a list of conditions or you can define a more complex rule using a custom Java class. For more information about using a custom Java class for rules, see Chapter 5, "Creating Custom Java for Orchestrations."

An orchestration rule functions similar to an EnterpriseOne query in that each rule has:

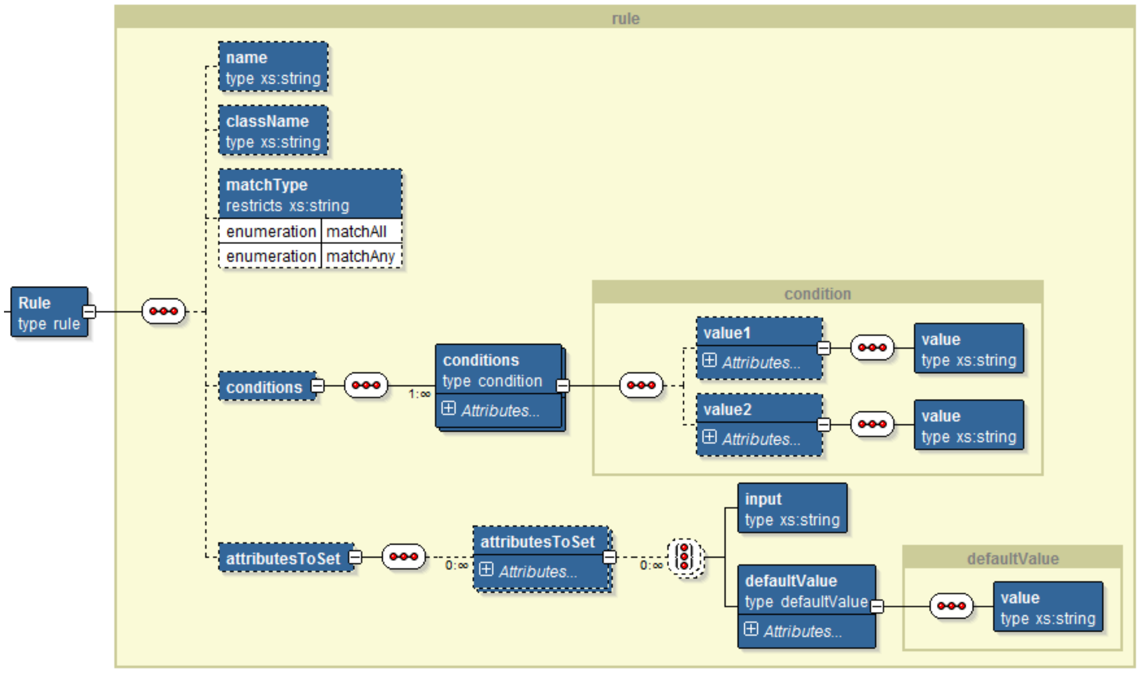
- A "Match Any" or "Match All" setting.
- One or more conditions defined, each being a binary operation on two values.

Each rule XML file must conform to the rules .xsd file included with the project. Do not use any spaces when naming a rule XML file.

4.3.2 Understanding the Rule XML Structure

The Rule element in a rule XML file provides the metadata for the rule. Figure 4–3 shows the basic structure of a rule XML file.

Figure 4–3 Rule XML Structure



4.3.3 Defining the Rule Metadata for the Orchestration

Table 4–5 provides a description of the Rule element attributes.

The following list describes the secondary and tertiary child elements that comprise a Rule element:

- **conditions**. This container element can include one or more child conditions elements. A child conditions element provides the definition of the comparison. See Table 4–6 for a description of the attributes for the conditions element. The conditions element also contains these child elements:
  - **value1**. See Table 4–7 for a list of the child elements and attributes for this element.
  - **value2**. This element has the same child elements and attributes as the **value1** element. See Table 4–7.
- **attributesToSet**. Custom rules have an attributesToSet element with a list of child attributesToSet elements just like the custom service request. See Table 4–4.

Table 4–5 Description of Rule Attributes

Attribute	Description
matchType	Valid values are: ■ `matchAll`. (default) Requires that all conditions return true before the rule returns true. ■ `matchAny`. Returns true if any conditions return true.
className	Identifies the custom Java class for a custom rule.

Table 4–6 Description of the conditions Element Attributes

Attribute	Description
operator	Valid values are: `=`, `!=`, `>`, `>=`, `&lt;`, `&lt;=`, `startsWith`, `endsWith`, `contains`, `between` or `inList`
type	Valid values are: `String`, `Numeric` or `Date`

Table 4–7 Description of Elements and Attributes in Value1 and Value2 Elements

Element or Attribute	Description
value (element)	Contains either a literal value or a named value.
literalValue (attribute)	Boolean type attribute used to indicate the value is literal.
dateFormat (attribute)	Identifies the format of the date if it is a literal value.

4.3.4 Example of a Rule XML

Example 4–2 shows the rule XML in the "Add Condition Based Maintenance Alert" sample orchestration.

Example 4–2 Rule XML in the "Add Condition Based Maintenance Alert " Sample Orchestration

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

```
xsi:noNamespaceSchemaLocation="JDE_Rule.xsd">
  <matchType>matchAll</matchType>
  <conditions>
    <conditions operator=">=" type="Numeric">
      <value1>
        <value>VibrationReading</value>
      </value1>
      <value2 literalValue="true">
        <value>90</value>
      </value2>
    </conditions>
    <conditions operator=">=" type="Numeric">
      <value1>
        <value>TemperatureReading</value>
      </value1>
      <value2 literalValue="true">
        <value>250</value>
      </value2>
    </conditions>
  </conditions>
</Rule>
```

4.4 Configuring Cross-Reference XMLs

This section contains the following topics:

- Section 4.4.1, "Understanding Cross-Reference XMLs"
- Section 4.4.2, "Understanding the Cross-Reference XML Structure"
- Section 4.4.3, "Defining the Cross-Reference Metadata"
- Section 4.4.4, "Examples of Cross-Reference XML and Cross-Reference Records in EnterpriseOne"

4.4.1 Understanding Cross-Reference XMLs

Cross-referencing converts third-party IDs into EnterpriseOne values for use in EnterpriseOne applications. You define the cross-references used in an orchestration in two places:

- P952000 in EnterpriseOne.

  In this application, you define the cross-references using "AIS" as the cross-reference type. The application supports many to many lookups using | delimited strings in the Third Party Value and EnterpriseOne value columns. For more information, see Setting Up Cross-References and White Lists in EnterpriseOne (P952000) in this guide.

- Metadata stored in a cross-reference XML file.

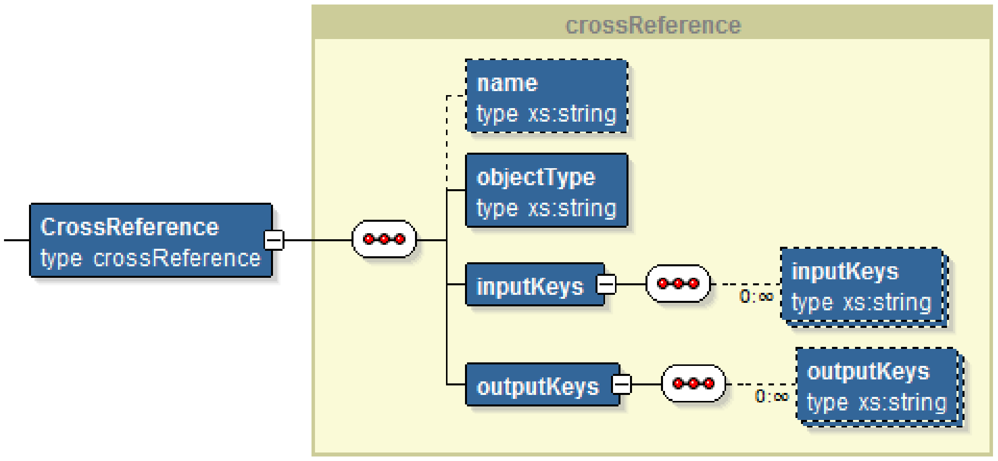
  Cross-reference XML files contain elements that identify the cross-reference metadata for an orchestration. The cross-reference XML must conform to the cross-reference .xsd file included with the project. Do not use any spaces when naming a cross-reference XML file.

If a cross-reference lookup fails in an orchestration, the orchestration is terminated.

4.4.2 Understanding the Cross-Reference XML Structure

Figure 4–4 shows the structure of the cross-reference XML.

Figure 4–4 Cross-Reference XML Structure



4.4.3 Defining the Cross-Reference Metadata

Use the elements in a cross-reference XML file to define the cross-reference metadata for an orchestration. A cross-reference XML file contains a parent or container CrossReference element. Table 4–8 describes the child elements in the CrossReference element. You use the child elements to define the cross-reference metadata for the orchestration.

Table 4–8 Description of CrossReference Elements

Element	Description
objectType	The value in this element must correspond to the cross-reference object type in P952000.
inputKeys	This element contains a list of inputKey elements in which inputKey is a named value. If more than one inputKey value is present, the values are joined with a | separator. After all inputKey values are read, the entire value is used to query the F952000 table in EnterpriseOne.
outputKeys	This element contains a list of outputKey elements in which outputKey is a string value that becomes a named value for use in subsequent orchestration steps. If more than one outputKey value is present, the EnterpriseOne value returned from the P952000 lookup is parsed into multiple strings using | as the separator.

4.4.4 Examples of Cross-Reference XML and Cross-Reference Records in EnterpriseOne

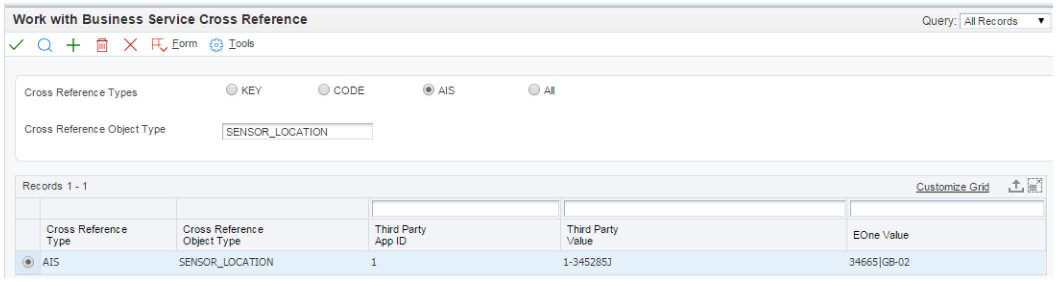
Example 4–3 shows the cross-reference XML in the "Add Condition Based Maintenance Alert" sample orchestration.

Example 4–4 shows the required cross-reference records in EnterpriseOne that correspond to the cross-reference XML defined in Example 4–3.

Example 4–3 Cross-Reference XML in the "Add Condition Based Maintenance Alert " Sample Orchestration

```
<?xml version="1.0" encoding="UTF-8"?>
<CrossReference xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="JDE_CrossReference.xsd">
  <objectType>SENSOR_LOCATION</objectType>
  <inputKeys>
    <inputKeys>SensorID</inputKeys>
  </inputKeys>
  <outputKeys>
    <outputKeys>EquipmentNumber</outputKeys>
    <outputKeys>MeasurementLocation</outputKeys>
  </outputKeys>
</CrossReference>
```

Example 4–4 Cross-Reference Record in EnterpriseOne for the "Add Condition Based Maintenance Alert" Sample Orchestration



Cross Reference Types: ( ) KEY ( ) CODE [✓ AIS] ( ) All

Cross Reference Object Type: SENSOR_LOCATION

Records 1 - 1

Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
AIS	SENSOR_LOCATION	1	1-345285J	34665|GB-02

4.5 Configuring White List XMLs

This section contains the following topics:

- Section 4.5.1, "Understanding White List XMLs"
- Section 4.5.2, "Understanding the White List XML Structure"
- Section 4.5.3, "Defining the White List Metadata for an Orchestration"
- Section 4.5.4, "Examples of White List XML and White List Records in EnterpriseOne"

4.5.1 Understanding White List XMLs

A white list contains a list of IDs permitted in the IoT Orchestrator. If you use a white list in your orchestration, any data not represented by an ID in the white list is not permitted in the IoT Orchestrator. You set up the white list in two places:

- P952000 in EnterpriseOne.

  This is the same application that is used to set up and store orchestration cross-references. As with cross-references, use the "AIS" cross-reference type in P952000 for a white list. The Third Party App ID should have a value of `WHITELIST`. The EnterpriseOne value is not used for white lists and will default to NA.

  For more information, see "Setting Up Cross-References and White Lists in EnterpriseOne (P952000)" in this guide.

- Metadata stored in a white list XML file.

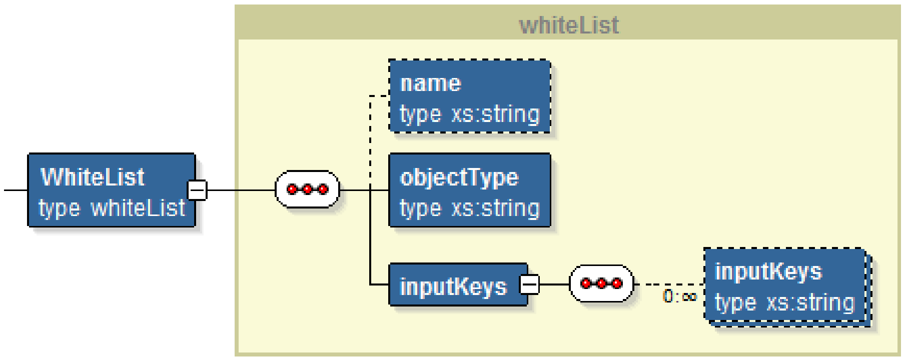
All white list metadata is stored in a white list XML. The white list XMLs must conform to the white list .xsd file included with the project. Do not use any spaces when naming a white list XML file.

If a white list lookup fails in an orchestration, the orchestration is terminated.

4.5.2 Understanding the White List XML Structure

Figure 4–5 shows the basic structure of the white list XML.

Figure 4–5 White List XML Structure



4.5.3 Defining the White List Metadata for an Orchestration

You use the elements in a white list XML file to define the white list metadata for the orchestration. A white list XML file contains a parent WhiteList element. Table 4–9 describes the elements in the WhiteList element.

Table 4–9 Description of WhiteList Elements

Element	Description
objectType	The value in this element must correspond to the cross-reference object type in P952000.
inputKeys	This element contains a list of inputKey elements in which inputKey is a named value. If more than one inputKey value is present, the values are joined with a | separator. After all inputKey values are read, the entire value is used to query the F952000 table in EnterpriseOne.

4.5.4 Examples of White List XML and White List Records in EnterpriseOne

Example 4–5 shows the white list XML in the "Add Condition Based Maintenance Alert" sample orchestration.

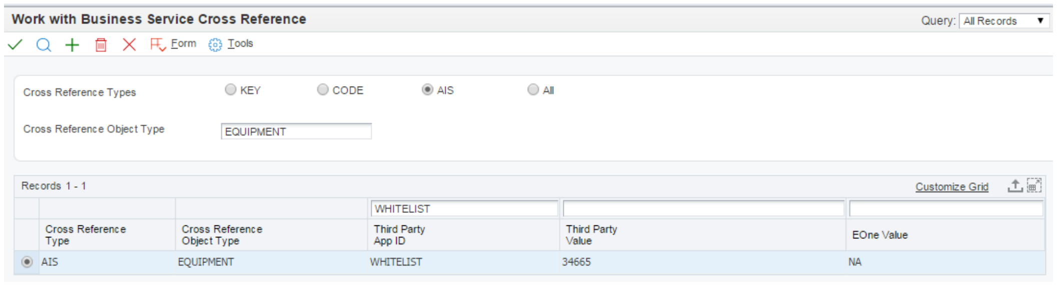
Example 4–6 shows the required white list records in EnterpriseOne that correspond to the white list entries defined in the "Add Condition Based Maintenance Alert" cross-reference XML in Example 4–7.

Example 4–5 White List XML in the "Add Condition Based Maintenance Alert" Sample Orchestration

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

```
xsi:noNamespaceSchemaLocation="JDE_WhiteList.xsd">
  <objectType>EQUIPMENT</objectType>
  <inputKeys>
    <inputKeys>EquipmentNumber</inputKeys>
  </inputKeys>
</WhiteList>
```

Example 4–6 White List Record in EnterpriseOne for the "Add Condition Based Maintenance Alert" Sample Orchestration



Cross Reference Types: KEY CODE [✓ AIS] All

Cross Reference Object Type: EQUIPMENT

Records 1 - 1

Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
		WHITELIST		
AIS	EQUIPMENT	WHITELIST	34665	NA

4.6 Configuring Orchestration XMLs

This section contains the following topics:

- Section 4.6.1, "Understanding Orchestration XMLs"
- Section 4.6.2, "Understanding the Orchestration XML Structure"
- Section 4.6.3, "Defining the Metadata for the Orchestration"
- Section 4.6.4, "Example of an Orchestration XML"

4.6.1 Understanding Orchestration XMLs

An orchestration XML file contains a series of orchestration steps that tie together the service request, rule, cross-reference, and white list metadata for an orchestration. The orchestration metadata is stored in an orchestration XML file. Orchestration XMLs must conform to the orchestration .xsd file included with the project.

The name of orchestration XML file is used to define an endpoint which is used to run the orchestration. The endpoint URL is:

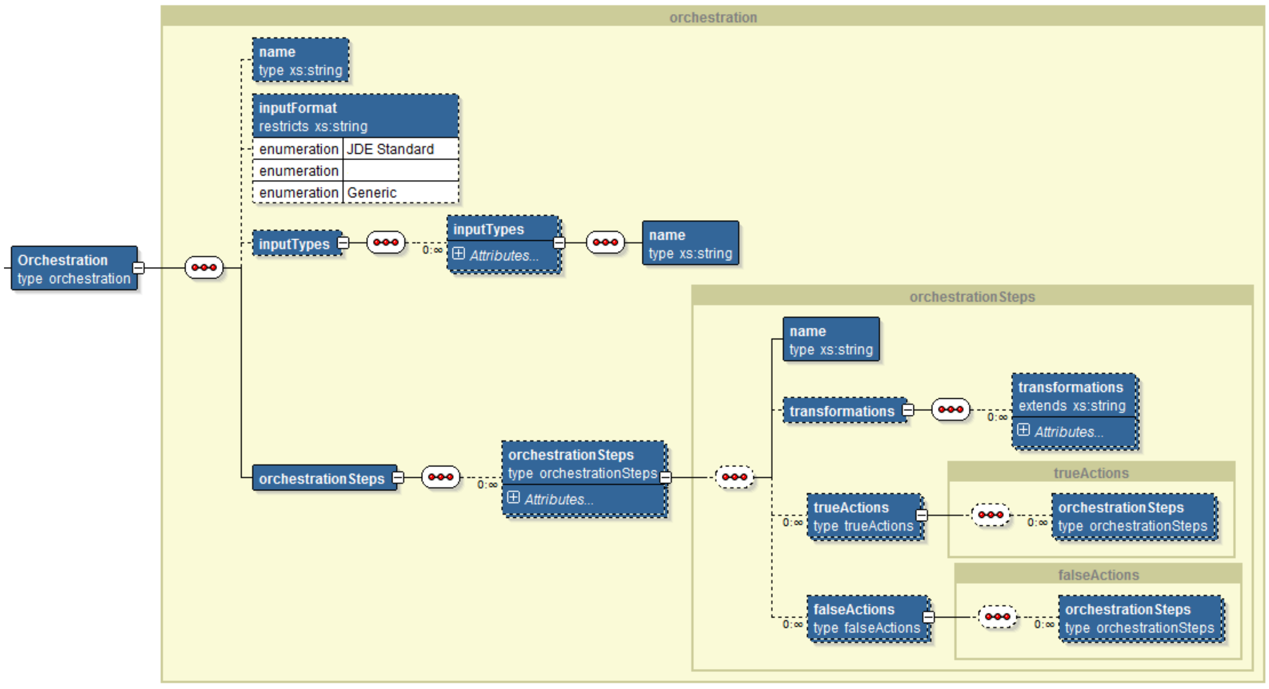
```
http://<server>:<port>/jderest/orchestrator/<orchestrationname>
```

Do not use any spaces when naming an orchestration XML file.

4.6.2 Understanding the Orchestration XML Structure

Figure 4–6 shows the basic structure of an orchestration XML:

Figure 4–6 Orchestration XML Structure



4.6.3 Defining the Metadata for the Orchestration

The following list describes the attributes and secondary and tertiary child elements that comprise an orchestration element. Use the following elements and attributes to define the metadata for the orchestration:

- **inputFormat** element. Valid values include: `JDE Standard` (default), `Generic`.
- **inputTypes** element. This element contains a list of inputTypes elements. See Table 4–10 for a description of the child elements and attributes of the inputTypes element.
- **orchestrationSteps** element. This element contains a list of orchestrationSteps elements. See Table 4–11 for a description of the child elements and attributes of the orchestrationSteps element.

Table 4–10 Description of inputTypes Element and Attributes

Element or Attribute	Description
name (element)	Identifies the input value.

type (attribute)	Valid values are:
	`String`
	`Numeric`
	`Date`

Table 4–10 (Cont.) Description of inputTypes Element and Attributes

Element or Attribute	Description
dateFormat (attribute)	Valid values are: ■ `dd/MM/yyyy` ■ `dd/MM/yy` ■ `yyyy/MM/dd` ■ `MM/dd/yyyy` ■ `MM/dd/yy` ■ `yy/MM/dd` ■ `Milliseconds -` When Milliseconds is used, the incoming value is converted to a date in the users date format and the following two additional values are available for use: `<name>_time` will contain the time portion of the incoming value in HH:mm:ss format. `<name>_uTime` will contain the incoming value converted so it can be used in a UTime field in EnterpriseOne. ■ yyyy-MM-dd'T'HH:mm:ss.SSSZ - When this date format is used, the following three additional values are available for use: <name>_date will contain the date portion of the incoming value converted to the users date format. <name>_time will contain the time portion of the incoming value in HH:mm:ss format. <name>_uTime will contain the incoming value converted so it can be used in a UTime field in EnterpriseOne.

Table 4–11 Description of OrchestrationSteps Element and Attributes

Element or Attribute	Description

type (attribute)	Valid values are:
	`CrossReference`
	`WhiteList`
	`Rule`
	`ServiceRequest`
	If an orchestrationSteps element has a type`=Rule`, then it can contain the following elements:
	■ **trueActions** element that contains a list of orchestrationSteps elements.
	■ **falseActions** element that contains a list of orchestrationSteps elements.
	Rules can be nested as deeply as needed by defining more Rule orchestration steps in the **trueActions** of **falseActions** of a parent rule.

iterateOver (attribute)	You can include the name of a repeating structure in the input message. The orchestration step will be repeated for each occurrence of the structure in the input message.
name (element)	This value should correspond to a name in one of the other XML files based on the specified type.

Table 4–11 (Cont.) Description of OrchestrationSteps Element and Attributes

Element or Attribute	Description
transformations (element)	Use this element to change the name of input values. If you have an existing orchestration step such as a service request that you want to reuse, but the name of the incoming input does not match the name the orchestration step is expecting, use this element to change the name of the input value. The transformations element has repeating transformations child elements, each with an input and output attribute. The input value matching the input element will be available for the orchestration step as the output element name. transformations are specific to an orchestration step. The output names are not available in subsequent orchestration steps. However, if the transformations are on a rule orchestration step, the output names are available for all of the true actions and false actions of that rule.

4.6.4 Example of an Orchestration XML

Example 4–7 shows the orchestration XML in the "Add Condition Based Maintenance Alert" sample orchestration.

Example 4–7 Orchestration XML in the "Add Condition Based Maintenance Alert" Sample Orchestration

```
<?xml version="1.0" encoding="UTF-8" ?>
<Orchestration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="JDE_Orchestration.xsd">
    <inputTypes>
        <inputTypes type="Date" dateFormat="Milliseconds">
            <name>Date</name>
        </inputTypes>
    </inputTypes>
    <orchestrationSteps>
        <orchestrationSteps type="Rule">
            <name>JDE_RULE_Sample_VibrationAlarm</name>
            <trueActions>
                <orchestrationSteps type="CrossReference">
                    <name>JDE_XREF_Sample_SensorLocation</name>
                </orchestrationSteps>
                <orchestrationSteps type="CrossReference">
                    <name>JDE_XREF_Sample_AlertNotificationRecipients</name>
                </orchestrationSteps>
                <orchestrationSteps type="ServiceRequest">
                    <name>JDE_SREQ_Sample_AddCBAlert_Alarm</name>
                </orchestrationSteps>
            </trueActions>
            <falseActions>
                <orchestrationSteps type="Rule">
                    <name>JDE_RULE_Sample_VibrationWarning</name>
                    <trueActions>
                        <orchestrationSteps type="CrossReference">
                            <name>JDE_XREF_Sample_SensorLocation</name>
                        </orchestrationSteps>
                        <orchestrationSteps type="CrossReference">
                            <name>JDE_XREF_Sample_
AlertNotificationRecipients</name>
                        </orchestrationSteps>
                        <orchestrationSteps type="ServiceRequest">
```

```
                            <name>JDE_SREQ_Sample_AddCBAlert_Warning</name>
                        </orchestrationSteps>
                    </trueActions>
                </orchestrationSteps>
            </falseActions>
        </orchestrationSteps>
    </orchestrationSteps>
</Orchestration>
```

4.7 Supported Input Message Formats

The IoT Orchestrator supports two input message formats for orchestrations: a standard JD Edwards EnterpriseOne format and generic format. Example 4–8 and Example 4–9 show an example of the code for each format.

Example 4–8 Standard JD Edwards EnterpriseOne Input Message Format

```
{
     "inputs": [
         {
             "name": "equipmentNumber",
             "value": "41419"
         },
         {
             "name": "description",
             "value": "test"
         },
         {
             "name": "date",
             "value": "1427774400000"
         },
         {
             "name": "time",
             "value": "12:15:15"
         },
         {
             "name": "temperature",
             "value": "99"
         }
     ]
}
```

Example 4–9 Generic Input Message Format

```
{
      "equipmentNumber": "41419",
      "description": "test",
      "date": "1427774400000",
      "time": "12:15:15",
      "temperature": "99"
}
```

Note: Additional Supported Input Message Formats

Additional formats are supported when using the generic input format, as long as the orchestration input values are defined using the full path to the elements used. You may have a deeper JSON structure like this.

```
{
    "equipmentNumber": "41419",
    "equipementDetail": {
        "type": "thermometer",
        "readingDetail": {
            "temperature": 200,
            "units": "F"
        }
    }
}
```

To reference the temperature within the orchestration you can use the full path delimited by periods, for example:

`equipmentDetail.readingDetail.temperature`

One difference between the two formats is the iterateOver attribute for the orchestrationStep element, which is supported only by the standard JD Edwards EnterpriseOne input message format. Also, when using the "detail" form action type with the standard format, it will automatically iterate over all detailInputs and repeatingInputs in order to add multiple rows to a grid. If the generic format is used, only a single grid row can be added.

As shown in Example 4–10, "detailInputs" would correspond to grid data; "repeatingInputs" would correspond to individual rows that contain "inputs" that correspond to columns.

If you have a power form with two grids, you could populate both grids using two "detailInputs" structures with different "name" values that correspond to the different grids. "repeatingInputs" would contain a list of rows and for each row you would define a list of "inputs".

You could also define a CrossReference orchestration step with `iterateOver="GridData"` that converts the item value into an EnterpriseOne specific item number. For example, if A123=220 and A124=230, the single orchestration step would convert both.

Example 4–10

```
{
    "inputs": [
        {
            "name": "BranchPlant",
            "value": "30"
        },
        {
            "name": "customer",
            "value": "4242"
        }
    ],
    "detailInputs": [
        {
                "name": "GridData",
                "repeatingInputs": [
                    {
                        "inputs": [
                            {
                                "name": "item",
                                "value": "A123"
                            },
                            {
                                "name": "Quantity",
                                "value": "3"
                            }
                        ]
                    },
                    {
                        "inputs": [
                            {
                                "name": "item",
                                "value": "A124"
                            }
                        ]
                    }
                ]
            }
        ]
    }
```

4.8 Setting Up Cross-References and White Lists in EnterpriseOne (P952000)

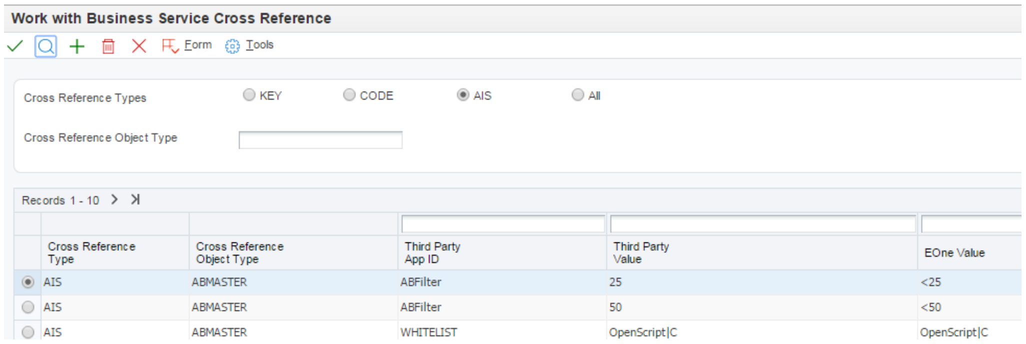
Use the EnterpriseOne Business Service Cross Reference (P952000) application to create and manage cross-reference and white list records for EnterpriseOne IoT orchestrations. Orchestration cross-references and white lists contain key-value data pairs used by the IoT Orchestrator. You must add records for all IoT orchestration key-value data pairs in P952000.

For example, an IoT device might provide a machine ID that equates to the Equipment Number field in EnterpriseOne. After including this in the cross-reference XML, you also have to add a record for this cross-reference to P952000 in order for the orchestration to invoke and perform the intended transaction in EnterpriseOne.

P952000 is generally used to set up cross-references for EnterpriseOne business services. Before you can use it for IoT orchestration cross-references and white lists, you must create a new user defined code (UDC) with which to associate the records for use with IoT orchestrations. See Creating the "AIS" User Defined Code for IoT Cross Reference and White List Records in this guide for details.

For instructions on how to add cross-reference and white list records, see the "Setting Up Orchestration Cross-References" in the *JD Edwards EnterpriseOne Tools Interoperability Guide*. When following the steps, make sure to perform these specific steps when creating records for IoT cross-references and white lists:

1. When creating a cross-reference or white list record, you must select the **AIS** option for the Cross Reference Type, which specifies that these records are for use with IoT orchestrations.



Work with Business Service Cross Reference

Form Tools

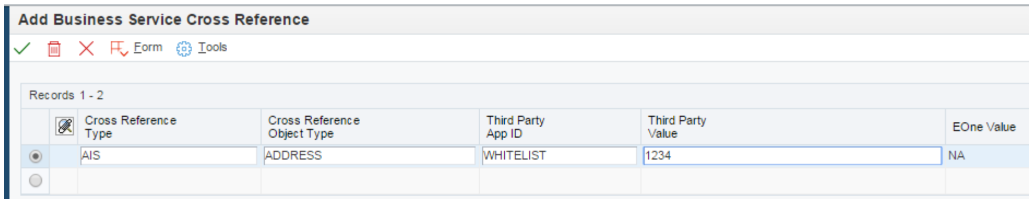
Cross Reference Types: KEY CODE [✓ AIS] All

Cross Reference Object Type

Records 1 - 10

	Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
●	AIS	ABMASTER	ABFilter	25	<25
	AIS	ABMASTER	ABFilter	50	<50
	AIS	ABMASTER	WHITELIST	OpenScript|C	OpenScript|C

2. When you add a white list record, you must enter `WHITELIST` for the Third Party App ID. This automatically changes the EOneValue column value to NA because the EOneValue column is not applicable to a white list.



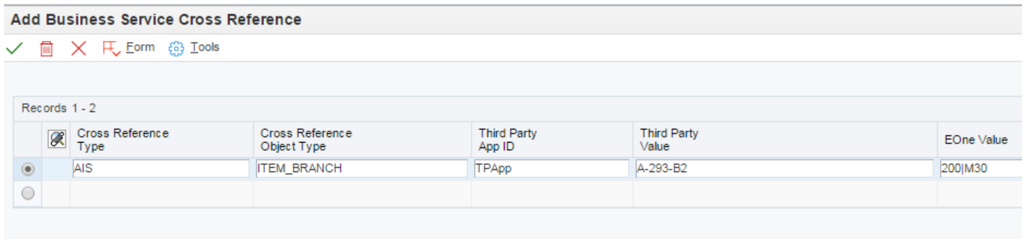
Add Business Service Cross Reference

Form Tools

Records 1 - 2

	Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
●	AIS	ADDRESS	WHITELIST	1234	NA

3. When setting up cross-references, you can enter multiple key cross-references by delimiting the values with a pipe (|). These will be consumed based on the cross-reference definition XML in the orchestration.



Add Business Service Cross Reference

Form Tools

Records 1 - 2

	Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
●	AIS	ITEM_BRANCH	TPApp	A-293-B2	200|M30

4.8.1 Creating the "AIS" User Defined Code for IoT Cross Reference and White List Records

Before you can create cross-reference and white list records in EnterpriseOne, you must create a user defined code (UDC) named "AIS" for these types of records.

Note: You do not have to create this UDC if you applied the EnterpriseOne Applications ESU described in the Prerequisites section in this guide. The UDC is installed with the ESU.

To create a UDC for AIS:

1. In EnterpriseOne, access Work With User Defined Codes (P0004A).
2. On Work With User Defined Codes, enter `H95` in the Product Code field and enter `CT` in the User Defined Codes field and click **Find**.
3. On the Work With User Defined Codes form, click **Add**.
4. On the User Defined Codes form, scroll to the last empty row of the detail area and complete each column with the values listed here:

- **Codes** = `AIS`
- **Description 1** = `AIS Type Cross Reference`
- **Hard Coded** = `N`

5. Click the **OK** to save the new UDC.

See Also: "Working with User Defined Codes" in the *JD Edwards EnterpriseOne Tools Foundation Guide* for more information about setting up UDCs.

4.9 Setting up Orchestration Security

Before the EnterpriseOne Orchestrator can process an orchestration, authentication of the JD Edwards EnterpriseOne user ID and password must take place. It is the responsibility of the originator of the service request to tell the orchestration about the user. The user's credentials must be supplied in a basic authorization header or in the JSON body of the request. The user must also have authorized access to the EnterpriseOne application in which the resulting transaction takes place.

The AIS service used with orchestrations is stateless; each call passes credentials to establish a separate EnterpriseOne session. After the transaction is complete, the session closes.

In addition to passing credentials for authentication, you can employ a second level of security for the IoT Orchestrator through whitelisting. Whitelisting enables an initial rudimentary pass/fail check of the incoming device signature against a predefined list of signatures. A white list provides an additional layer of security to the IoT Orchestrator security. If a value passed to the IoT Orchestrator is not a valid value included in the orchestration's white list, the IoT Orchestrator rejects the input. For more information, see Configuring White List XMLs in this guide.

4.10 Using the EnterpriseOne Orchestrator Client to Build the Input Message and Test the Orchestration

The EnterpriseOne Orchestrator Client is a standalone web application that enables you to build input JSON messages and test your orchestrations. In the Orchestration Client, you can input a set of name-value pairs that represent the expected input of the orchestration, and then execute a call to the orchestration with this input. The Orchestrator Client works with the AIS Server and EnterpriseOne to test an orchestration configuration.

The Orchestrator Client is available with a deployed AIS Server and runs in a Web browser.

In the Orchestrator Client, you identify inputs that you defined in the XML, and then you enter values for these inputs that you want the orchestration to pass into EnterpriseOne. The Orchestrator Client invokes the orchestration, passing the test values to the orchestration, which in turn uses the orchestration's service request information to invoke an EnterpriseOne transaction through the AIS Server.

Oracle recommends that you use the Orchestrator Client with an EnterpriseOne test environment, as the test results in an EnterpriseOne transaction that adds data to the database.

To access the Orchestrator Client:

1. Enter the following URL in the Web browser's address bar:

   `http://<ais_server>:<port>/jderest/client`

2. On the Orchestrator Client Sign In screen, enter your EnterpriseOne user credentials, environment, and role. It is highly recommended that you enter an EnterpriseOne environment used for testing, not a production environment.
3. Click the **Login** button.

To use the Orchestrator Client:

Note: Oracle recommends upgrading to EnterpriseOne Tools 9.1.5.6 before using the Orchestrator Client. If using Tools 9.1.5.5., you have to manually enter the name of the orchestration in the Orchestration Name field and manually enter the name of the inputs.

1. In the Orchestrator Client, click in the **Orchestration Name** field, and then select an orchestration from the list of available orchestrations on the AIS Server. Depending on the browser you are using, you might have to double-click in the field to display the orchestration list.
2. After selecting the orchestration, tab out of the field or click anywhere else on the form.

   The Orchestrator Client displays the inputs or name-value pairs, which should be defined in the orchestration.xml as the expected input for the orchestration. If the inputs do not appear, then they are not defined in the orchestration.xml. You need to update the orchestration.xml with a list of inputs before continuing.
3. In the Value column, enter a value for each input.
4. Click the **add icon** (plus symbol) to add and define additional inputs as necessary.
5. As an alternative to using the input fields, click the **JSON Input** check box if you want to enter the input in JSON format directly in the Input area. (Available with EnterpriseOne Tools release 9.1.5.6.)
6. Click the **Generic Inputs** check box if the orchestration uses the generic format for inputs. If this check box is cleared or left unchecked, it indicates that the orchestration uses the standard JD Edwards EnterpriseOne format.

   For more information about input formats, see Supported Input Message Formats in this guide.
7. Click the **Run** button to test the orchestration.

   The Input area shows the input message in JSON format.

   The Output area shows the result of the orchestration. If successful, it shows the resulting forms from each form service call in JSON format. If unsuccessful, it shows an error response in JSON format.

   To verify the orchestration invoked the transaction in EnterpriseOne, access the EnterpriseOne application and perform a search. If the results in the grid show records with the new values, then the orchestration test was successful.

   If the orchestration fails, you might have to modify the XML files on the server. Make sure to click the **XML Cache Refresh** button to ensure the modified XMLs are used on the next run.

8. To test another orchestration or start over, click the **Clear** button to reset the Orchestration Client, which clears all values in the form.

5

Creating Custom Java for Orchestrations

This chapter contains the following topics:

- Section 5.1, "Understanding Custom Java for Orchestrations"
- Section 5.2, "Creating Custom Java"
- Section 5.3, "Deploying Custom Java"

5.1 Understanding Custom Java for Orchestrations

You can include a custom Java class for service requests and rules within an orchestration. Within the custom Java classes, any number of private attributes can be declared. As long as the accessor (get/set) methods are generated for the attributes, the JD Edwards EnterpriseOne IoT Orchestrator can assign attributes from input values from the orchestration. Then within the appropriate method, those values can be used to make AIS calls into EnterpriseOne or any other logic to either evaluate a rule or perform another action.

5.2 Creating Custom Java

When creating custom Java classes, you must reference these JAR files, which are dependencies:

- **OrchestratorCustomJava.jar**. This contains the definition of the interfaces.
- **AIS_Client.jar 1.1.0 or higher**. This contains the loginEnvironment attribute and enables AIS calls to JD Edwards EnterpriseOne.

It is not necessary to include these JARs with the deployment as they are already deployed as part of the AIS Server deployment. After creating a custom Java class, you deploy the Java class or Java classes to a JAR file.

A custom rule should implement the CustomRuleInterface class included with the OrchestratorCustomJava.jar. The interface requires a loginEnvironment variable of type `com.oracle.e1.aisclient.LoginEnvironment` and an `evaluate()` method that takes no parameters and returns a Boolean value.

A custom service request should implement the CustomServiceRequestInterface class also included with the OrchestratorCustomJava.jar. The interface will require a loginEnvironment variable of type `com.oracle.e1.aisclient.LoginEnvironment` and a `process()` method that takes no parameters and returns a `javax.ws.rs.core.Response`.

5.2.1 Using Custom Java for the Orchestration Service Request

Custom service request Java classes should implement the `com.oracle.e1.rest.orchestrator.customjava.CustomServiceRequstInterface` which requires the following methods:

- `setLoginEnvironment(com.oracle.e1.aisclient.LoginEnvironment loginEnvironment)`. The method used to perform AIS calls.
- `process()`. The method that returns `javax.ws.rs.core.Response` which is called automatically after all the attributes are set.

5.2.2 Using Custom Java for the Orchestration Rule

Custom rule Java classes should implement the `com.oracle.e1.rest.orchestrator.customjava.CustomRuleInterface` which requries the following methods:

- `setLoginEnvironment(com.oracle.e1.aisclient.LoginEnvironment loginEnvironment)`. The method used to perform AIS calls.
- `evaluate()`. The method that returns Boolean and is called automatically after all the attributes are set.

5.3 Deploying Custom Java

You must deploy the JAR file that contains the custom Java to the AIS Server. Follow the instructions accordingly depending on the server on which the AIS Server is installed:

- Deploying Custom Java on AIS Server on Oracle WebLogic Server
- Deploying Custom Java on AIS Server on IBM WebSphere Application Server

5.3.1 Deploying Custom Java on AIS Server on Oracle WebLogic Server

To deploy the custom Java JAR file to an AIS Server on Oracle WebLogic Server:

1. Deploy the JAR as a shared library on the same WebLogic Server on which the AIS Server (otherwise referred to as the JDERestProxy) is deployed.
2. Restart the AIS Server using Server Manager.
3. Edit the weblogic.xml inside the JDERestProxy.war/WEB-INF to reference the custom java shared library, for example:

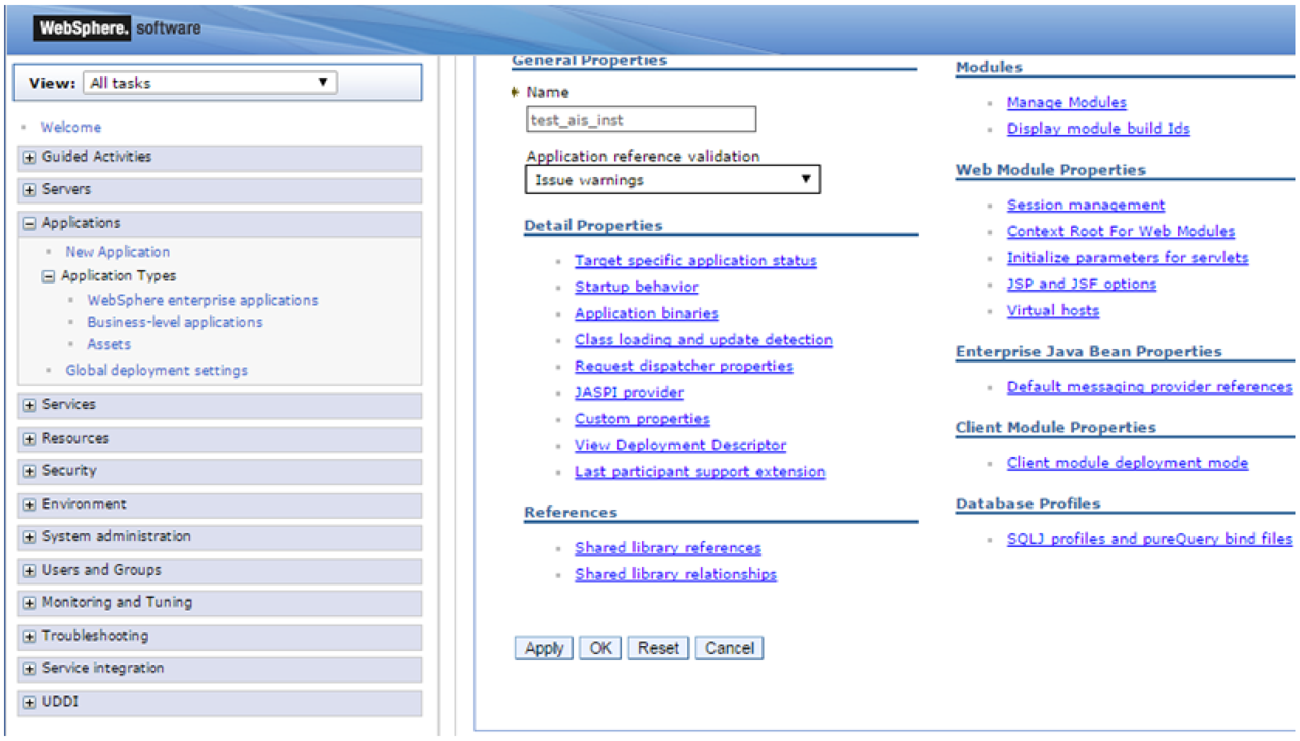
```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-web-app
http://www.bea.com/ns/weblogic/weblogic-web-app/1.0/weblogic-web-app.xsd"
                  xmlns="http://www.bea.com/ns/weblogic/weblogic-web-app">
    <session-descriptor>
        <cookie-path>/jderest</cookie-path>
        <cookie-http-only>true</cookie-http-only>
    </session-descriptor>
    <context-root>jderest</context-root>
    <library-ref>
        <library-name>CustomJava</library-name>
    </library-ref>
</weblogic-web-app>
```

4. Redeploy JDERestProxy from Server Manager.

5.3.2 Deploying Custom Java on AIS Server on IBM WebSphere Application Server

To deploy the custom Java JAR file to the AIS Server on Websphere Application Server:

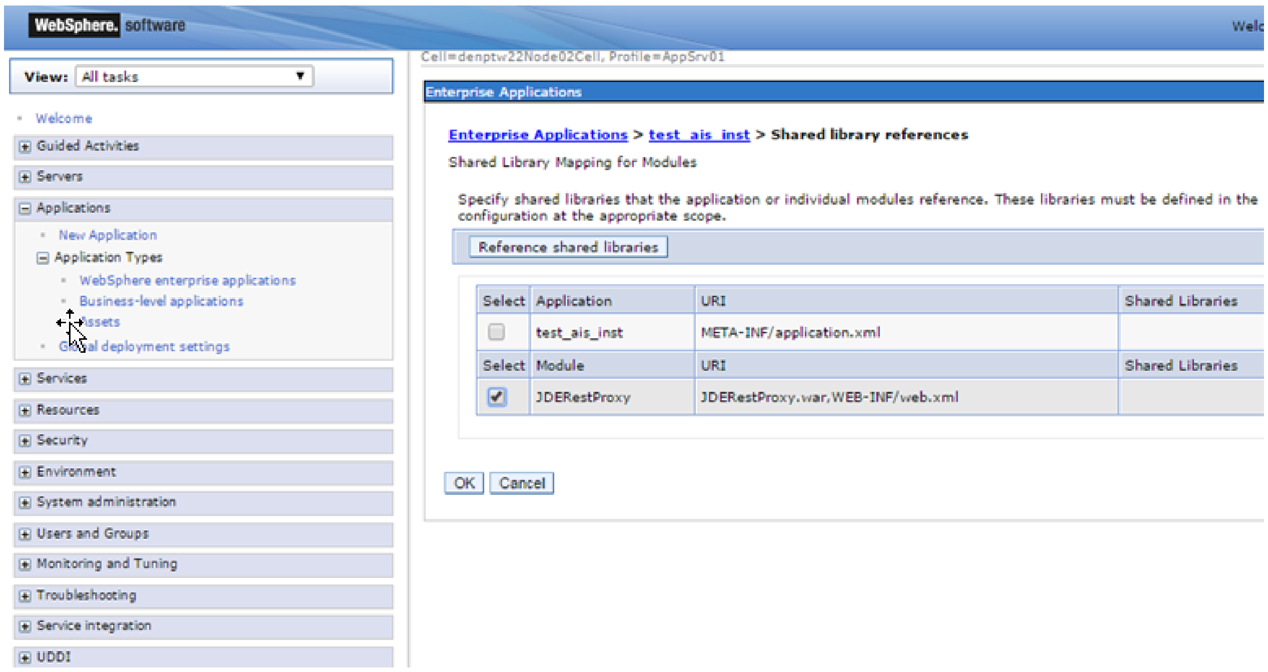
1. Add the JAR as a shared library:
   a. On WebSphere, expand Environment and select **Shared Libraries**.
   b. In ClassPath, add the path to the JAR location on the server.
2. Associate the shared library with the JDERestProxy application:



* Name: test_ais_inst

Application reference validation: Issue warnings

   a. In the left pane, expand **Applications**, **Application Types**, and then select **WebSphere enterprise applications**.
   b. Select the appropriate AIS deployment.
   c. Under References, select **Shared library references**.



Cell=denptw22Node02Cell, Profile=AppSrv01

Enterprise Applications

Enterprise Applications > test_ais_inst > Shared library references

Shared Library Mapping for Modules

Specify shared libraries that the application or individual modules reference. These libraries must be defined in the configuration at the appropriate scope.

Select	Application	URI	Shared Libraries
☐	test_ais_inst	META-INF/application.xml	
Select	**Module**	**URI**	**Shared Libraries**
✓	JDERestProxy	JDERestProxy.war,WEB-INF/web.xml	

   d. Select the **JDERestProxy** check box and then select the **Reference shared libraries** button.

   e. Use the directional arrow to move the JAR file to the Selected group.

   f. Click **OK**.

3. Synchronize Server Manager with the deployed application:

   Saving the configuration in Websphere will redeploy the application, but you must synchronize Server Manager to recognize the deployed application.

   a. In Server Manager, locate the AIS Server and update a setting in the Configuration section. This is required so that Server Manager detects a change in the AIS Server when you click the Synchronize Configuration button.

   b. Apply the changes and then return to the AIS Server home page.

   c. Click the **Synchronize Configuration** button to restart the AIS Server.

6 Managing IoT Orchestrations

This chapter contains the following topics:

6.1 Using Server Manager to Manage IoT Orchestrations

Server Manager provides AIS Server settings for managing IoT orchestrations. These settings include:

- **Full path to XML files**. Enter the location of IoT orchestrations for processing by the IoT Orchestrator. Make sure the path contains forward slashes (/). If the host is a Windows machine, replace the backslashes with forward slashes.
- **Enable Admin Service**. Click this option to activate the AIS Administration Service, a service on the AIS Server that enables you to delete AIS Server cache after modifying an orchestration. See Managing IoT Orchestration Modifications Using the AIS Administration Service in this guide for more information.
- **AdminServiceUserList**. Enter a comma delimited list of EnterpriseOne user IDs for users allowed to run the AIS Administration Service. For example:

  ```
  AdminServiceUserList=USR10,USR22, USR12
  ```

6.2 Managing the Life Cycle of IoT Orchestrations

Oracle recommends setting up different instances of the AIS Server: an instance for designing and testing IoT orchestrations and another instance for production. This enables you to test your orchestrations before making them available for runtime.

In Server Manager, for each AIS Server instance, make sure to include the path to the orchestration XML files in the **Full path to XML files** field.

When your testing is complete, you can simply copy and paste the orchestration XML files from the location identified in the test instance to the location used for production.

6.3 Managing IoT Orchestration Modifications Using the AIS Administration Service

The AIS Server caches all orchestration XML files processed by the IoT Orchestrator. If you modify any orchestration XML files currently in use, you must clear the AIS Server cache for the modified files to be used. Clearing the cache forces the AIS Server to reload files from disk to cache, which is required for modifications to take effect.

Note: Clearing the cache is not necessary to run new orchestration XML files.

You can clear the cache using either of the following methods:

- Restarting the AIS Server. This method is not recommended because:
  - If the IoT Orchestrator is currently running an orchestration, it will result in invalid processing of that orchestration.
  - It results in server downtime, which might affect other applications that use the AIS Server.
- Using the AIS Administration Service to clear the AIS Server cache (**recommended**). The AIS Administration Service is a REST service on the AIS Server that is exposed like any other service on the AIS Server.

Regardless of the method you use, Oracle recommends that you clear the cache only on an AIS Server instance used for developing and testing orchestrations. Either method clears the XML file cache for all orchestrations; you cannot clear the cache for individual orchestrations.

Using the AIS Administration Service to Clear Cache

Before you can use the AIS Administrator Service, you have to activate the AIS Admin Service option in the AIS Server instance's General settings in Server Manager.

You can run the service simply by placing a URL in a browser with valid EnterpriseOne credentials. The AIS Administrator Service takes an EnterpriseOne username and password as input. You can invoke the service using either a GET or a POST HTTP method.

The URI is:

```
http://<ais_server>:<port>//jderest/adminservice
```

The AIS Administrator Service is able to take credentials in several forms.

For the GET call, you can either use Basic Authorization or you can provide the username and password as URL parameters, for example:

```
http://<ais_
server>:<port>/jderest/adminservice?username=<userid>&password=<pwd>
```

For the POST operation you can use Basic Authorization, provide username and password as parameters, or provide username and password as JSON input, for example:

```
{
"username":"user",
"password":"pwd"
}
```

If the service succeeds, the response looks like this:

```
{"message":"All XML File Caches and X-Ref/Whitelist Caches have been
cleared and refreshed.","timeStamp":"2015-04-30:14.26.58"}
```

If the service fails, these are the possible reasons:

- Invalid credentials, which is indicated by the following response:

  ```
  {"message":"Error: Authorization Failure Server returned HTTP response
  code: 403 for URL: http://<jas_
  server>:<port>/jde/FormServiceRequest","exception":"com.oracle.e1.rest
  .session.E1LoginException","timeStamp":"2015-04-30:14.28.59"}
  ```

- Service disabled in settings, which is indicated by the following response:

  ```
  {"message":"Error: Admin Service is disabled in configuration. No
  action has been taken. ","timeStamp":"2015-04-30:14.31.41"}
  ```

- User has not been added to the AIS Server's AdminServiceUserList setting in Server Manager, which is indicated by the following response:

  ```
  {"message":"User is not authorized to use the Admin
  Service","timeStamp":"2015-07-07:14.23.25"}
  ```

A Sample Orchestrations

This appendix describes the prerequisite for using sample orchestrations and describes how to use the following sample orchestrations which are intended for testing purposes only:

- Add Conditioned Based Maintenance Alert Sample Orchestration
- Update Equipment Location Sample Orchestration
- Update Meter Reading Sample Orchestration

A.1 Prerequisite

If you have not already done so, download the Sample Orchestrations package. See Section 2.2, "Prerequisites" for download instructions.

A.2 Running the Sample Orchestrations

Use the EnterpriseOne Orchestrator Client to run each of the sample orchestrations. Before you can run the sample orchestrations, copy the XML files for each sample to the appropriate folder in the Orchestration directory. See Figure 4–1, "Orchestration Directory Folders and Files".

After the XML files are in the directory, use the Orchestration Client to enter inputs and test the sample orchestrations. The following sections in this appendix describe the inputs, or name-value pairs, in the sample orchestration XML files. You enter these inputs in the Orchestration Client to test the sample orchestrations. See Using the EnterpriseOne Orchestrator Client to Build the Input Message and Test the Orchestration for instructions on how to test the sample orchestrations.

A.3 Add Conditioned Based Maintenance Alert Sample Orchestration

The "Add Conditioned Based Maintenance Alert" sample orchestration conditionally creates conditioned based alerts based on vibration and temperature readings from a device. The orchestration is defined in the JDE_ORCH_Sample_AddConditionBasedAlert.xml located in the Orchestrations folder. This sample orchestration includes:

- A cross-reference that converts the incoming SensorID into EquipmentNumber and MeasurementLocation defined in the JDE_XREF_Sample_SensorLocation.xml file in the CrossReferences folder.
- An orchestration step to find the WarningRecipient and AlarmRecipient from the EquipmentNumber, which is also a cross-reference defined in JDE_XREF_Sample_AlertNotificationRecipients.xml.

- A series of rules to determine if an alert is needed and if so, whether it is an alarm or warning. The rules used for this orchestration are defined in the following XML files in the Rules folder:
  - JDE_RULE_Sample_CBMAlarm_1.xml
  - JDE_RULE_Sample_CBMAlarm_2.xml
  - JDE_RULE_Sample_CBMAlarm_3.xml
  - JDE_RULE_Sample_CBMWarning.xml
- If necessary, the alert is created using either the JDE_SREQ_Sample_AddCBAlert_Alarm.xml or JDE_SREQ_Sample_AddCBAlert_Warning.xml defined in the ServiceRequests folder. These service requests invoke the edit form (W1311B) of P1311 application to create the conditioned based alert.

A.3.1 Sample Input

In the Orchestrator Client, use the following inputs to test the Add Conditioned Based Maintenance Alert orchestration.

```
     "inputs": [
         {
             "name": "SensorID",
             "value": "1-345285J"
         },
         {
             "name": "Date",
             "value": "1433311200000"
         },
         {
             "name": "Time",
             "value": "12:15:15"
         },
         {
             "name": "VibrationReading",
             "value": "100"
         },
         {
             "name": "TemperatureReading",
             "value": "350"
         }
     ]
}
```

A.4 Update Equipment Location Sample Orchestration

The "Update Equipment Location" sample orchestration creates a new equipment location to store the current latitude and longitude of the asset. The orchestration is defined in the JDE_ORCH_Sample_UpdateEquipmentLocation.xml file located in the Orchestrations folder. This sample orchestration includes:

- A cross-reference orchestration step that converts the incoming DeviceID into EquipmentNumber. This cross-reference is defined in the JDE_XREF_Sample_Equipment.xml file in the CrossReferences folder.
- A final orchestration step that includes the service request JDE_SREQ_Sample_UpdateEquipmentLocation.xml. This service request runs a series of applications

using the application stack service to create the equipment location details. The application stack service flow performs the following tasks:

1. Accesses P1704 application - Work with Equipment Locations.
2. Invokes the Add button to create a new equipment location header record.
3. After the record is created, the record is queried back in the Work With Equipment Locations form.
4. It then accesses the Details form and saves the latitude, longitude, and elevation in this form.

A.4.1 Sample Input

```
{
    "inputs": [
        {
            "name": "CustomerNumber",
            "value": "4244"
        },
        {
            "name": "SiteNumber",
            "value": "4244"
        },
        {
            "name": "DeviceID",
            "value": "1-345213A"
        },
        {
            "name": "Latitude",
            "value": "39.649844"
        },
        {
            "name": "Longitude",
            "value": "-104.856342"
        },
        {
            "name": "Elevation",
            "value": "5642"
        }
    ]
}
```

A.5 Update Meter Reading Sample Orchestration

The "Update Meter Reading" sample orchestration updates meter readings of a piece of equipment. The orchestration is defined in the JDE_ORCH_Sample_UpdateMeterReadings.xml file located in the Orchestrations folder. This sample orchestration includes:

- An initial white list orchestration step that validates that the incoming EquipmentNumber is allowed to run the orchestration.
- A final orchestration step that includes the service request JDE_SREQ_Sample_UpdateMeterReadings.xml. This service request runs the Speed Meter Readings application (P12120U) to update the fuel meter reading and the hour meter reading of the passed in equipment number.

A.5.1 Sample Input

```
{
     "inputs": [
         {
             "name": "EquipmentNumber",
             "value": "34665"
         },
         {
             "name": "NewFuelMeterReading",
             "value": "13.2"
         },
         {
             "name": "NewHourMeterReading",
             "value": "101.7"
         }
     ]
}
```

B Troubleshooting

This appendix contains the following topics:

- Section B.1, "Enable Debugging on the AIS Server"
- Section B.2, "Troubleshooting Orchestration Runtime Issues"

B.1 Enable Debugging on the AIS Server

Turn on debugging on the AIS Server to generate an AIS Server log file. The log file includes information for all orchestrations unless you activate user level logging. User level logging generates logs based on the EnterpriseOne user ID. See "Available Log Files" in the *JD Edwards EnterpriseOne Tools Server Manager Guide* for more information about generating and managing log files.

The details in the log file can help you troubleshoot and resolve orchestration issues and EnterpriseOne connection issues.

B.2 Troubleshooting Orchestration Runtime Issues

Interruption of EnterpriseOne Web Client When Running the JD Edwards EnterpriseOne Orchestrator Client

If a user is signed in to the EnterpriseOne Web client and then opens the Orchestrator Client in a new tab in the same browser, the user can no longer perform any actions in the EnterpriseOne Web client. The EnterpriseOne Web client session is interrupted and prompts the user to sign in again.

This issue is caused by an Orchestrator Client JSESSIONID cookie conflict with WebSphere and occurs when both the EnterpriseOne AIS Server and EnterpriseOne HTML Server are deployed on WebSphere on the same host. Both servers use the JSESSIONID cookies for session management, which causes a conflict because both JSESSIONID cookies are configured with a generic path.

To fix this issue, set up the cookie path of the AIS Server:

1. In the WebSphere administration console, navigate to the server: **Application servers**, ***ais_server_name***.
2. Under Container Settings, select **Session Management**.
3. Click **Enable Cookies** link.
4. Under Cookie path, make sure the "**Set cookie path**" option is selected and set the cookie path to:

   ```
   /jderest
   ```

5. Restart the server.