ORACLE
ENTERPRISE DATA QUALITY

Using Java Messaging Service (JMS) with Oracle Enterprise Data Quality (EDQ)

Technical Guide

ORACLE WHITE PAPER | OCTOBER 2014

Table of Contents

Introduction

This document provides a brief technical guide for how to get started using Java Messaging Service (JMS) technology with Oracle Enterprise Data Quality (EDQ).

Message Queue Architectures

JMS is a flexible technology that allows EDQ to integrate with a variety of different messaging systems and technologies, including WebLogic Server Messaging, WebSphere MQ, Oracle Advanced Queueing (OAQ), and many more.

For simpler architectures, and to allow customers to get started using JMS, EDQ is shipped with ActiveMQ (http://activemq.apache.org/) embedded. This can be enabled on an EDQ server simply by adding the following line to `director.properties`:

```
launch.activemq=1
```

Depending on the messaging architecture, an EDQ server may be acting as a 'client' of a message queue, or as a 'server'. Most commonly, EDQ acts as client. Where EDQ acts as a client, it may need additional client jars to be installed; this is the case if the server which EDQ is reading messages from and writing messages to uses a different message queueing technology from the application server EDQ is deployed on. For example, if EDQ is deployed on WebLogic or Tomcat but needs to act as a client to a server running WebSphere MQ, it will need additional client jars to be installed. In this case, the administrator of the MQ system should be consulted to determine which files are needed for a Java application to act as a client.

Uses of JMS with EDQ

There are two main uses of JMS with EDQ, as follows. This document is focused on the first of these:

» **Consuming and providing messages**: this is where EDQ can be configured to read messages from JMS queues, and write messages to them. This can be beneficial where several EDQ servers need to read from a single stream of messages that needs to be persistent to ensure no loss of messages. In this mode, each EDQ server will consume messages from a queue and only 'commit' when it has finished processing each message. Note that JMS is best used for Asynchronous communication, where EDQ is not expected to return a response to a calling application for a specific message.

» **Using JMS in Triggers**: EDQ may send JMS messages to other systems, or may consume JMS messages to start triggers (for example to use a JMS queue to distribute batch jobs amongst several EDQ servers; see the EDQ Triggers documentation: http://docs.oracle.com/middleware/1213/edq/DQSAG/triggers.htm#A1159457

Configuring EDQ to read and write JMS messages

JMS interfaces to and from EDQ are configured using XML interface files that define:

- The path to the queue of messages
- Properties that define how to work with the specific JMS technology
- How to decode the message payload into a format understood by an EDQ process (for Message Providers – where EDQ reads messages from a queue), or convert messages to a format expected by an external process (for Message Consumers – where EDQ writes messages to a queue).

The XML files are located in the EDQ Local Home directory (formerly known as the config directory), in the following paths:

- `buckets/realtime/providers` (for interfaces 'in' to EDQ)
- `buckets/realtime/consumers` (for interfaces 'out' from EDQ)

Once the XML files have been configured, Message Provider interfaces are available in Reader processors in EDQ and to map to Data Interfaces as 'inputs' to a process, and Message Consumer interfaces are available in Writer processors, and to map from Data Interfaces as 'outputs' from a process as in Figures 1 and 2 below:

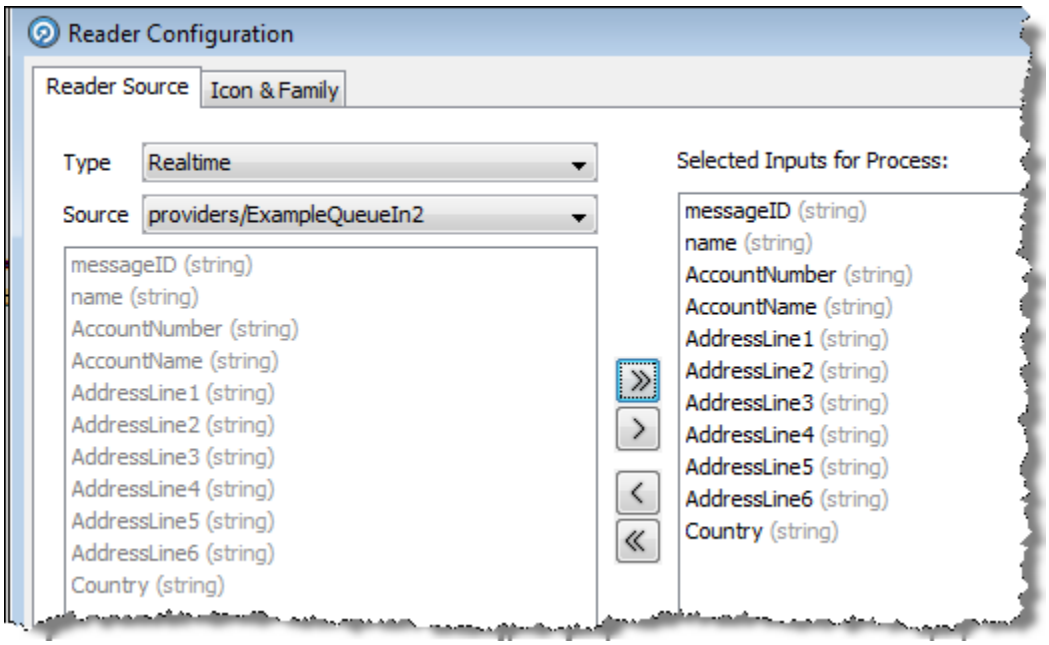

Figure 1 - Reader processor in Director showing JMS message provider

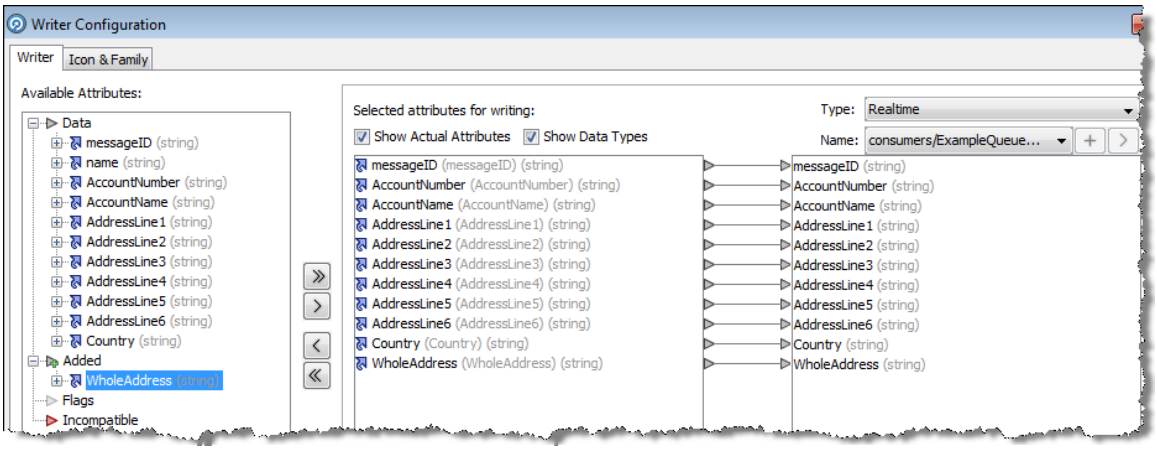

Figure 2 - Writer processor showing JMS message consumer

Defining the interface files

An interface file in EDQ consists of three sections, as follows:

1. The **<attributes>** section, defining the shape of the interface as understood by EDQ
2. The **<messengerconfig>** section, defining how to connect to the JMS queue or topic
3. The **<messagebody>** section, defining how to extract contents of a message (e.g. from XML) into attribute data readable by EDQ (for inbound interfaces), or how to convert attribute data from EDQ to message data (e.g. in XML).

The <attributes> section

The `<attributes>` section defines the shape of the interface, i.e. the attributes that EDQ will be available when configuring a Reader or Writer. For example the following excerpt from the beginning of an interface file configures three string attributes that can be used in EDQ, and their names:

```
<?xml version="1.0" encoding="UTF-8"?>

<realtimedata messenger="jms">

        <attributes>

              <attribute type="string" name="messageID"/>

              <attribute type="string" name="name"/>

              <attribute type="string" name="AccountNumber"/>

       </attributes>

[file continues]...
```

The supported attribute types are all the standard types supported in EDQ, as follows:

- `string`
- `number`
- `date`
- `stringarray`
- `numberarray`
- `datearray`

The <messengerconfig> section

The `<messengerconfig>` section of the interface file configures the settings needed to connect to a given JMS queue or topic on a particular type of JMS technology. The text in `<messengerconfig>` is parsed as a Java properties file and is merged with the set of properties from the `realtime.properties` file in the EDQ configuration path, with settings in the `<messengerconfig>` section overriding any matching settings in `realtime.properties`. The `realtime.properties` file therefore allows the configuration of a number of global properties that do not need to be stated in every JMS interface file.

> **Note:** As with all property files stored in the EDQ Local Home directory, properties are themselves merged with a set of 'base' properties in the EDQ Home directory which should not be modified, with any property stated in a file in the Local Home directory used in preference to any property in the EDQ Home directory. For example the `realtime.properties` file in the EDQ Local Home directory will be merged with the file of the same name in the EDQ Home directory to form the final set of properties used.

Note that `realtime.properties` values may also be used to set properties for Web Services in EDQ. A prefix of **jms.** is therefore used (in this file only, not in the `<messengerconfig>` section of a JMS interface file) to set properties for JMS (**ws.** is used for Web Services).

The messenger configuration properties, specified either in the `<messengerconfig>` section of a JMS interface file, or inherited from `realtime.properties`, are used:

- To configure a JNDI name store to look up queues
- To specify the name of the queue/topic and connection factory
- To supply authentication information to the MQ server, if required

JNDI setup properties

java.naming.factory.initial: the class name of the JNDI initial context factory, used to bootstrap JNDI naming

java.naming.provider.url: the JNDI URL used to connect to the JNDI server or local store

In some cases, authentication is required to connect to the JNDI service. In this case the following properties must be added:

java.naming.security.principal: JNDI user name

java.naming.security.credentials: JNDI password

These JNDI properties are a standard Java feature - see http://docs.oracle.com/javase/8/docs/technotes/guides/jndi/index.html for full documentation.

JMS names

The other properties are:

connectionfactory: the JNDI name of the JMS 'connection factory'. The default for this (if the property is not set) is 'ConnectionFactory' which is correct for several MQ servers.

destination: the JNDI name of the JMS queue or topic. There is no default for this and it is always required.

Authentication

If the MQ server requires authentication when making connections, add these properties:

username: connection user name
password: connection password

Note that these properties are used to authenticate against the MQ server; the properties above are used to connect to JNDI. In rare cases, both sets may be required.

Notes for specific MQ servers

ActiveMQ

For an installation of EDQ that is not on WebLogic, the installed `realtime.properties` file in the EDQ Home directory contains the following settings, suitable for use with the local embedded ActiveMQ server:

```
jms.java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
jms.java.naming.provider.url=vm://localhost?create=false
```

Therefore, if the local embedded ActiveMQ server is used, the `<messengerconfig>` section of a JMS interface file requires only a **destination** property setting.

To connect to a remote ActiveMQ server, use a property setting like:

```
java.naming.provider.url=tcp://host:port
```

This can be set in the `<messengerconfig>` section to apply to a single interface, or can be set in `realtime.properties` (prefixed with **jms.**) to set the default for all interfaces.

Remote ActiveMQ servers generally require connection authentication, so the **username** and **password** properties will be required. The embedded ActiveMQ server uses EDQ user authentication, so the user account used needs the 'Connect to Messaging System' permission.

To use ActiveMQ on an EDQ server installed on WebLogic, the property settings above need to be entered into the `realtime.properties` in the EDQ Local Home directory. (The settings are present, but commented out, in the file in the Home directory, as it is assumed that WebLogic JMS will normally be used where EDQ is installed on WebLogic.)

WebLogic JMS

When running in a full JavaEE application server, such as WebLogic, Glassfish or WebSphere, the JNDI settings for local JMS are configured automatically and the JNDI factory and url information do not need to be specified in any EDQ configuration file.

The recommended approach in these cases is to define the JMS configuration within the application server and then just specify the JNDI connection factory and destination names in `<messengerconfig>`.

WebLogic supports the concept of 'foreign' JMS servers. In this case, you define the connection information in the WebLogic Console and expose the destination(s) and connection factory as names in the native WebLogic JNDI store. This works well with Oracle Advanced Queueing (AQ). See below for a snippet of the configuration of a 'foreign JMS server' pointing at an AQ schema.

When JMS is configured like this, a typical `<messengerconfig>` section might be:

```
<messengerconfig>
   connectionfactory = jms/cf1
   destination       = jms/queue1
</messengerconfig>
```

Here `jms/cf1`and `jms/queue1` are JNDI names defined in WebLogic.

The <messagebody> section

This section uses JavaScript to parse message payloads into attributes that EDQ can use for inbound interfaces, and perform the reverse operation (convert EDQ attribute data into message payload data) for outbound interfaces. A function named 'extract' is used to extract data from XML into attribute data for inbound interfaces, and a function named 'build' is used to build XML data from attribute data. Scripts are best illustrated using examples – see the next section.

Example Interface files

The following example interface files show typical settings that may be used with the embedded ActiveMQ JMS server (enabled by adding `launch.activemq=1` to `director.properties` in the EDQ Local Home directory and restarting the server).

Example 1 – Simple Provider File

The following XML is a simple example of a provider interface file, reading messages from a queue in the path 'dynamicQueues/InputQueue'.

```
<?xml version="1.0" encoding="UTF-8"?>
<realtimedata messenger="jms">
        <attributes>
              <attribute type="string" name="messageID"/>
              <attribute type="string" name="name"/>
              <attribute type="string" name="AccountNumber"/>
              <attribute type="string" name="AccountName"/>
              <attribute type="string" name="Country"/>
        </attributes>
<messengerconfig> destination = dynamicQueues/InputQueue </messengerconfig>
        <incoming>
              <messagebody>
```

```
                    <script>
                         <![CDATA[ function extract(str) { var screeningRequest = new
                         XML(str); var rec = new Record(); rec.messageID =
                         screeningRequest.individual.messageID; rec.name =
                         screeningRequest.individual.name; rec.AccountNumber =
                         screeningRequest.individual.AccountNumber; rec.AccountName =
                         screeningRequest.individual.AccountName; rec.Country =
                         screeningRequest.individual.Country; return [rec]; } ]]>
                     </script>
                </messagebody>
               <eof>
                      <messageheader name="JMSType" value="t1eof"/>
               </eof>
        </incoming>
</realtimedata>
```

Example 2 – Simple Consumer File

The following XML is a simple example of a consumer file representing similar data to the provide file above but with additional attributes, which might for example be added by an EDQ process:

```
<?xml version="1.0" encoding="UTF-8"?>
<realtimedata messenger="jms">
        <attributes>
               <attribute type="string" name="messageID"/>
               <attribute type="string" name="AccountNumber"/>
               <attribute type="string" name="AccountName"/>
               <attribute type="string" name="Country"/>
               <attribute type="string" name="AccountType"/>
       </attributes>
       <messengerconfig> destination = dynamicQueues/OutputQueue </messengerconfig>
       <outgoing>
               <messagebody>
                      <script>
                          <![CDATA[ function build(recs) { var rec = recs[0]; var xml =
                          <Request> <individual>
                          <messageID>{checkNull(rec.messageID)}</messageID>
                          <AccountNumber>{checkNull(rec.AccountNumber)}</AccountNumber>
                          <AccountName>{checkNull(rec.AccountName)}</AccountName>
                          <Country>{checkNull(rec.Country)}</Country>
                          <AccountType>{checkNull(rec.AccountType)}</AccountType> </individual>
                          </Request>; return xml.toXMLString(); } function checkNull(value) {
                          return value != null ? value : ""; } ]]>
                      </script>
               </messagebody>
       </outgoing>
</realtimedata>
```

Example 3 – Date parsing and formatting

This example file shows an example of how to define a DATE attribute type and include date parsing and formatting when decoding the message payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<realtimedata messenger="jms">
      <attributes>
            <attribute type="date" name="processingDate"/>
      </attributes>
      <messengerconfig> destination = dynamicQueues/HoldQueue </messengerconfig>
      <incoming>
            <messagebody>
                  <script>
                      <![CDATA[ var df = Formatter.newDateFormatter("yyyy-MM-
                      dd'T'HH:mm:ss.SSSZ"); function extract(str) { var
                      screeningRequest = new XML(str); var rec = new
                      Record();rec.processingDate =
                      date(screeningRequest.processingDate; return [rec]; } function
                      date(x) { var s = x.text().toString(); return s == "" ? null :
                      df.parse(s); } ]]>
                  </script>
             </messagebody>
             <eof>
                  <messageheader name="JMSType" value="t1eof"/>
             </eof>
      </incoming>
 </realtimedata>
```

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Using Java Messaging Service (JMS) with Oracle Enterprise Data Quality (EDQ)
October 2014
Author: Mike Matthews
Contributing Authors: Richard Evans, Gerry Kelley