

Integrating OpenEdge with Multiple JMS Providers Channels

OpenEdge JMS Adapter

Introduction

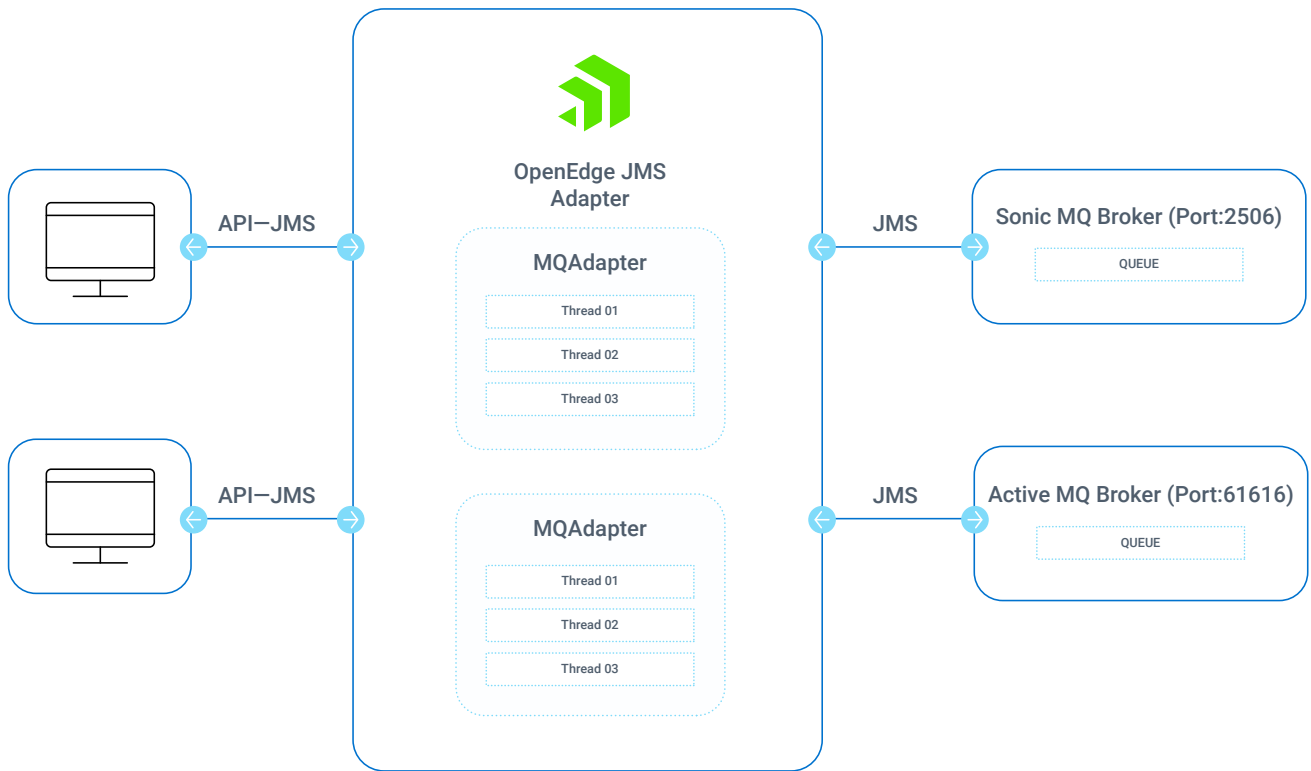
The Java Message Service (JMS) is a messaging standard Java API for sending and receiving messages between two or more applications. Progress has supported standard JMS messaging for a long time.

With the introduction of the [Progress® OpenEdge® JMS adapter](#) in 11.7.2 and 12.0.0, the dependency on the AdminServer and NameServer is removed, as the JMS Adapter is independent and should work with any OpenEdge server. This new component allows you to run multiple JMS adapters at the same time using BrokerConnect mode. This whitepaper covers how to configure an OpenEdge instance to work with multiple JMS adapters.

The Progress OpenEdge JMS adapter is available as a separate download from the OpenEdge installation in the OpenEdge 12 release family. For installation instructions for Windows and UNIX platforms, see [Install the Progress OpenEdge JMS Adapter](#) documentation.

Architecture

The OpenEdge JMS Adapter is built to work with any compliant JMS provider. In the diagram below, you can see that there is one adapter instance for two sample providers: SonicMQ and ActiveMQ. Each adapter works with the provider it has been configured for.



NOTE: This whitepaper assumes that you have the two JMS providers configured for use with your OpenEdge JMS Adapter.

For information on how to configure a JMS provider, see the vendor specific documentation.

How to Configure Multiple JMS Adapters for PAS for OpenEdge

There are four high-level tasks you must perform to configure multiple JMS providers to work with PAS for OpenEdge using the OpenEdge JMS Adapter's BrokerConnect mode. This section provides a brief overview of the four tasks. The next section of this document shows a step-by-step example of these tasks in practice.

High-level tasks:

- Verify that your JMS provider is listed in the `$DLC/properties/jmsProvider.properties` file
- Set up the OpenEdge JMS Adapter configuration in the `ubroker.properties` file
- Set environment variables for each JMS provider's OpenEdge JMS Adapter
- Start the OpenEdge JMS Adapter for each JMS provider

Verify that your JMS provider is listed in the `$DLC/properties/jmsProvider.properties` file

By default, the `$DLC/properties/ubroker.properties` file contains the following definition of the example `sonicMQ1` adapter:

```
[MyJMSProvider]
javax.jms.ConnectionFactory=
javax.jms.QueueConnectionFactory=
javax.jms.TopicConnectionFactory=
```

Set Up the JMS Adapter Configuration in `ubroker.properties`

By default, the `$DLC/properties/ubroker.properties` file contains the following definition of the example `sonicMQ1` adapter:

```
#  
# Sample SonicMQ adapter definition  
#  
[Adapter.sonicMQ1]  
appserviceNameList=adapter.progress.jms  
brokerLogFile=@{WorkPath}\sonicMQ1.broker.log  
controllingNameServer=NS1  
description=Sample SonicMQ Adapter broker  
portNumber=3620  
srvrLogFile=@{WorkPath}\sonicMQ1.server.log  
uuid=932.99.999.XXX:1ee77e:cf3bbe3d33:-8030
```

For every JMS adapter that you want to configure, add a similar section and customize the property values. An example of this is shown in Step 1 of the next section of this document, “Setting Up Multiple JMS Adapters for OpenEdge.”

Set Environment Variables for each JMS provider’s OpenEdge JMS Adapter

For every OpenEdge JMS Adapter that you want to start, you must set the following two environment variables in a separate `proenv` session:

For example:

```
proenv> export JMSPROVIDER=ActiveMQ  
proenv> export JMSCLIENTJAR=path/to/ActiveMQ/activemq-all-5.15.6.jar
```

The JMS provider client jar file is provided as part of your JMS provider installation.

Setting Up Multiple JMS Adapters for PAS for OpenEdge

Follow these step-by-step instructions for setting up multiple JMS adapters for a PAS for OpenEdge instance and executing a round-trip using SonicMQ, ActiveMQ and WebSphere MQ.

1. Add/validate JMS adapter sections as required in ubroker.properties. The following example shows JMS adapter definitions for ActiveMQ and WebSphere MQ, which are provided in addition to the Sonic MQ adapter definition:

```
#
# Sample ActiveMQ adapter definition
#
[Adapter.ActiveMQ1]
appserviceNameList=adapter.progress.jms
brokerLogFile=@{WorkPath}\ActiveMQ1.broker.log
controllingNameServer=NS1
description=Sample ActiveMQ Adapter broker
portNumber=3621
srvrLogFile=@{WorkPath}\ActiveMQ1.server.log
uuid=932.99.999.XXX:1ee77f:cf3bbe3d33:-8031
#
# Sample WebSphere MQ adapter definition
#
[Adapter.WebSphereMQ1]
appserviceNameList=adapter.progress.jms
brokerLogFile=@{WorkPath}\WebSphereMQ1.broker.log
controllingNameServer=NS1
description=Sample WebSphereMQ Adapter broker
portNumber=3622
srvrLogFile=@{WorkPath}\WebSphereMQ1.server.log
uuid=932.99.999. XXX:1ee77g:cf3bbe3d33:-8032
```

2. Launch two proenv windows. In one window, set JMSPROVIDER and JMSCLIENTJAR for ActiveMQ. In the other window, set JMSPROVIDER and JMSCLIENTJAR for WebSphere MQ.

proenv1 (ActiveMQ):

```
proenv>export JMSPROVIDER=ActiveMQ
proenv>export JMSCLIENTJAR=$DLC/java/ext/hawtbuf-1.11.
jar: $DLC/java/ext/slf4j-api-1.7.5.jar:$DLC/java/ext/geronimo-
j2eemanagement_1.1_spec-1.0.1.jar:$DLC/java/ext/activemq-client-
5.15.6.jar:$DLC/ java/ext/geronimo-jms_1.1_spec-1.1.1.jar
proenv> oemessaging start ActiveMQ1
```

NOTE: Here ActiveMQ1 is a broker name in ubroker.properties

proenv2 (WebSphere MQ):

```
proenv>export JMSPROVIDER=WebSphereMQ
proenv>export JMSCLIENTJAR=<IBM_install_Location>/MQShared/java/
lib/com.ibm.
mqjms.jar:$WRKDIR/AdminObjectFinder.jar
proenv>oemessaging start WebSphereMQ1
```

NOTE: Similarly, WebSphereMQ1 is a broker name in ubroker.properties

proenv3 (SonicMQ):

```
proenv>export JMSPROVIDER=SonicMQ
proenv>export JMSCLIENTJAR=<Sonic-client-jar-path>/ sonic_Client.jar
proenv>oemessaging start
```

NOTE: Releases before 12.2 defaulted to SonicMQ1 adapter and changed to GenericMQ1 adapter from 12.2 OE version onwards. We strongly recommend explicitly naming the adapter to oemessaging start.

3. Example procedures to send and receive from an ABL client. Sample code for <JMS Adapter>_producer.p to send a message to a queue:

```
/* Sending a message to SampleQ1 */
DEFINE VARIABLE hMessage AS HANDLE NO-UNDO.
DEFINE VARIABLE hPTPSession AS HANDLE NO-UNDO.
/* Creates PTP session*/
RUN jms/ptpsession.p PERSISTENT SET hPTPSession ("-H localhost -S
<port> -DirectConnect -AppService AD.<MQName>").
/*Connects to the broker */
RUN setBrokerURL IN hPTPSession (INPUT "tcp//
machinename:2506").
RUN setUser IN hPTPSession (INPUT "<username>").
RUN setPassword IN hPTPSession (INPUT "<password>").
RUN beginSession IN hPTPSession.
/* Create a message */
RUN createTextMessage IN hPTPSession (OUTPUT hMessage).
RUN setText IN hMessage ("Message").
/*Send the message to "SampleQ1" */
RUN sendToQueue IN hPTPSession ("SampleQ1", hMessage, ?, ?, ?)
/* Delete message and session */
RUN deleteMessage IN hMessage.
RUN deleteSession IN hPTPSession.

Message "Sent".
```

Sample code for <JMS Adapter>_consumer.p to consume a message from a queue:

```
/* Receives a message from SampleQ1. */
DEFINE VARIABLE hConsumer AS HANDLE NO-UNDO.
DEFINE VARIABLE hPTPSession AS HANDLE NO-UNDO.
/* Creates PTP session*/
RUN jms/ptpsession.p PERSISTENT SET hPTPSession ("-H localhost -S
<port> -DirectConnect -AppService AD.<MQName>").
/*Connects to the broker */
RUN setBrokerURL IN hPTPSession (INPUT "tcp//
machinename:2506").
RUN setUser IN hPTPSession (INPUT "<username>").
```


4. Example of messaging with PAS for OpenEdge from an ABL client

```
RUN setPassword      IN hPTPSession (INPUT "<password>").
RUN beginSession IN hPTPSession.
/* Messages received from SampleQ1 are handled by the "myintproc"
procedure. */
RUN createMessageConsumer IN hPTPSession
    (THIS-PROCEDURE, "myintproc", OUTPUT hConsumer).
RUN receiveFromQueue IN hPTPSession ("SampleQ1", ?, hConsumer).
RUN startReceiveMessages IN hPTPSession.
/* Wait to receive the messages. */
AIT-FOR u1 OF THIS-PROCEDURE.
/* Delete session */
RUN deleteSession IN hPTPSession.
PROCEDURE myintproc:
    DEFINE INPUT  PARAMETER hMessage AS HANDLE NO-UNDO.
    DEFINE INPUT  PARAMETER hConsumer AS HANDLE NO-UNDO.
    DEFINE OUTPUT PARAMETER hReply   AS HANDLE NO-UNDO.
    DISPLAY DYNAMIC-FUNCTION('getText':U IN hMessage) format "x(70)".
    /* Delete message */
    RUN deleteMessage IN hMessage.
    APPLY "U1" TO THIS-PROCEDURE.

END.
```

NOTE: Create more producer and consumer programs for respective JMS Adapters by updating the sample program code mentioned above.

Example of messaging with PAS for OpenEdge from an ABL client

1. Create a PAS for OpenEdge instance. The following command creates an instance called instJMS:

```
proenv>pasman create instJMS
```

2. Deploy ABL code to the instance's /openedge directory.

```
proenv>cp Sonic_producer.p instJMS/openedge
proenv>cp Sonic_consumer.p instJMS/openedge
proenv>cp ActiveMQ_producer.p instJMS/openedge
proenv>cp ActiveMQ_consumer.p instJMS/openedge
proenv>cp WebSphereMQ_producer.p instJMS/openedge
proenv>cp WebSphereMQ_consumer.p instJMS/openedge
```

3. Start the OpenEdge instance.

```
proenv>instJMS/bin/tcman.sh start
```

4. Develop a client to connect to the OpenEdge instance and execute the deployed ABL code.

For example, the following is an ABL client program that connects to a OpenEdge instance over APSV with the request to execute Sonic_producer.p:

```
DEFINE VARIABLE UserName AS CHARACTER.
DEFINE VARIABLE Pwd AS CHARACTER.
DEFINE VARIABLE happsrv AS HANDLE NO-UNDO.
CREATE SERVER happsrv.
IF happsrv:CONNECT("-URL http://<machine name>:<port>/apsv") THEN

DO :
    RUN Sonic_producer.p ON happsrv.
    happsrv:DISCONNECT().

END.

DELETE OBJECT happsrv.
```

NOTE: You can create more client programs by replacing Sonic_producer.p with Sonic_consumer.p, ActiveMQ_producer.p, ActiveMQ_consumer.p, WebSphereMQ_producer.p and WebSphereMQ_consumer.p



About the Author

Sai Tharun Kollampally is a Senior Software Engineer at Progress. Tharun has started his career on server components and gained extensive experience on growing server technologies. He is an active participant in hackathons, a quick learner and proactive in his role.







Learn More about working with the Generic JMS Adapter

About Progress

Progress (NASDAQ: PRGS) provides the leading products to develop, deploy and manage high-impact business applications. Our comprehensive product stack is designed to make technology teams more productive and enable organizations to accelerate the creation and delivery of strategic business applications, automate the process by which apps are configured, deployed and scaled, and make critical data and content more accessible and secure—leading to competitive differentiation and business success. Learn about Progress at www.progress.com or +1-800-477-6473.

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved. Rev 2021/11 RITM0135219

 /progresssw
 /progresssw
 /progresssw
 /progress-software