PROGRESS OpenEdge

# White Paper:
# Generic JMS Adapter for Progress OpenEdge 11.5.1

PROGRESS

# Notices

**© 2015 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Business Making Progress, Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Deliver More Than Expected, Easyl, Fathom, Icenium, Kendo UI, Making Software Work Together, OpenEdge, Powered by Progress, Progress, Progress Control Tower, Progress RPM, Progress Software Business Making Progress, Progress Software Developers Network, Rollbase, RulesCloud, RulesWorld, SequeLink, SpeedScript, Stylus Studio, TeamPulse, Telerik, Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. AccelEvent, AppsAlive, AppServer, BravePoint, BusinessEdge, DataDirect Spy, DataDirect SupportLink, , Future Proof, High Performance Integration, Modulus, NativeScript, OpenAccess, Pacific, ProDataSet, Progress Arcade, Progress Pacific, Progress Profiles, Progress Results, Progress RFID, Progress Progress Software, ProVision, PSE Pro, SectorAlliance, Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

The Release Notes can be found in the OpenEdge installation directory and online at:
https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx.

For the latest documentation updates see OpenEdge Product Documentation on Progress Communities: (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx).

**May 2015**

**Last updated with new content:** Release 11.5.1

# Table of Contents


**Chapter 1: Introduction.....7**

Prerequisites.....7

**Chapter 2: Configuring the JMS Provider for OpenEdge.....9**

Updating the jmsProvider.properties File for the JMS Provider's Name.....9

Updating the jvmargs Property.....10

Locating the Connection Factory.....10

Locating the Connection Factory Using the AdminServerPlugins.properties File .....11

Locating the Connection Factory Using JNDI.....11

Updating your ABL-JMS API.....14

Non-supported API methods for Generic JMS Adapter .....14

# 1

# Introduction

Progress OpenEdge now provides a generic Java Message Service (JMS) adapter for messaging. This generic JMS adapter can operate with any JMS-compliant vendor, for example, SonicMQ, IBM WebSphereMQ, and ActiveMQ.

## Prerequisites

This white paper assumes that you have configured a JMS provider server and it is ready for use. Ensure that you have access to binaries that are required for the JMS provider.

Make sure the AdminServer and SonicAdapter Broker are installed.

You need to know the values of the following JMS provider properties:

- `PROVIDER_HOST:PROVIDER_PORT`: The host name and port number of the JMS provider server
- `PROVIDER_CONNECTION_FACTORY`: The name of the connection factory of the JMS provider
- `PROVIDER_QUEUE, PROVIDER_TOPIC`: The names of the queues and topics of the JMS provider that are to be used

# 2

# Configuring the JMS Provider for OpenEdge

To use the generic JMS adapter, you must point OpenEdge to the JMS provider that you are using. To achieve this, OpenEdge has to find the connection factory specific to your JMS provider.

**Note:** Genric JMS Adapter supports only the BrokerConnect mode. For more information on the other modes available and the installation, configuration, and the supported JMS-API for SonicMQ as the message provider, see the *OpenEdge Development: Messaging and ESB* guide.

## Updating the jmsProvider.properties File for the JMS Provider's Name

Open `%DLC%\properties\jmsProvider.properties` in a text editor, and add the JMS provider's name and the following properties with the required connection factory classes:

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

**Note:** The `jmsProvider.properties` file contains the commonly used JMS provider names. You can add your JMS provider's name and the properties in the same format if it is not listed.

Here is a sample of the `jmsProvider.properties` file that provides connection factory class names for WebSphereMQ and ActiveMQ.

```
[WebSphereMQ]
javax.jms.ConnectionFactory=com.ibm.mq.jms.MQConnectionFactory
javax.jms.QueueConnectionFactory=com.ibm.mq.jms.MQQueueConnectionFactory
javax.jms.TopicConnectionFactory=com.ibm.mq.jms.MQTopicConnectionFactory

[ActiveMQ]
javax.jms.ConnectionFactory=org.apache.activemq.ActiveMQConnectionFactory
javax.jms.QueueConnectionFactory=org.apache.activemq.ActiveMQConnectionFactory
javax.jms.TopicConnectionFactory=org.apache.activemq.ActiveMQConnectionFactory
```

**Note:** Do not modify the `javax.jms.ConnectionFactory`, `javax.jms.QueueConnectionFactory`, and `javax.jms.TopicConnectionFactory` property names. The property values for these property names must be specific to the provider.

# Updating the jvmargs Property

In the `%DLC%\properties` path, open the `AdminServerPlugins.properties` file and update the jvmargs property in the `PluginPolicy.Progress.SonicMQ` property section as follows:

```
PluginPolicy.Progress.SonicMQ]

pluginclasspath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar

classpath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar

jvmargs=-DsonicMQExtensions=false -DjmsProvider=WebSphereMQ
```

The `sonicMQExtensions` property indicates whether to use the new adapter. You can set this property as `-DsonicMQExtensions=false` to use the generic JMS adapter as the JMS provider or set as `-DsonicMQExtensions=true` to use SonicMQ as the JMS provider.

**Note:** The value for the `-DjmsProvider` property must match the name of the JMS provider specified in the `jmsProvider.properties` file.

# Locating the Connection Factory

You can either update the `AdminServerPlugins.properties` file or use the Java Naming and Directory Interface (JNDI) to locate the connection factory.

# Locating the Connection Factory Using the AdminServerPlugins.properties File

To use a JMS provider other than SonicMQ, you have to replace the references to the Sonic client JAR files with your provider JAR files. To do this, open `%DLC%\properties\AdminServerPlugins.properties` in a text editor and add the location of the client JAR files of your JMS provider to the `pluginclasspath` and the `classpath` properties. In the following example, com.ibm.mqjms.jar is the WebSphereMQ JMS client jar that is appended to the `pluginclasspath` and the `classpath` properties.

```
PluginPolicy.Progress.SonicMQ]

pluginclasspath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar

classpath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar

jvmargs=-DsonicMQExtensions=false -DjmsProvider=WebSphereMQ
```

**Note:** `SonicMQ` is a string in the section header and `PluginPolicy.Progress.SonicMQ` is the name of the plug-in that OpenEdge provides, not the SonicMQ provider.

The provider JAR files are added to the classpath, and the run-time picks up the provider-specific connection factory objects.

# Locating the Connection Factory Using JNDI

You can locate the connection factory by using JNDI to find the JMS-administered objects in the JNDI namespace.

To prevent class conflict with the `AdminObjectFinder.java` file, the generic JMS adapter packages the class in a different package, jmsfromABL(not jmsfrom4gl). This new class provides the adapter with the configuration to find the JMS-administered objects in the JNDI namespace.

Do the following:

1. Use the following example to create a `jmsfromABL.AdminObjectFinder.java` class file.

   **AdminObjectFinder.java for IBM WebsphereMQ**

```
package jmsfromABL;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.TopicConnectionFactory;
import javax.jms.QueueConnectionFactory;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NameNotFoundException;
import javax.naming.NamingException;
import java.util.Hashtable;
```

```
public class AdminObjectFinder{

    public Context context = null;
    public AdminObjectFinder() throws Exception{

        Hashtable<String, String> env = new Hashtable<>();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.sun.jndi.fscontext.RefFSContextFactory");
        env.put(Context.PROVIDER_URL, "file:/C:/JNDI");
        env.put(Context.SECURITY_PRINCIPAL, "username");
        env.put(Context.SECURITY_CREDENTIALS, "password");
        context = new InitialContext(env);
    }

    public TopicConnectionFactory getTopicConnectionFactory(String name)
    throws Exception {
        TopicConnectionFactory factory = null;
        factory = (javax.jms.TopicConnectionFactory)context.lookup(name);
        return factory;
    }

    public QueueConnectionFactory getQueueConnectionFactory(String name)
    throws Exception {
        QueueConnectionFactory factory = null;
        factory = (javax.jms.QueueConnectionFactory)context.lookup(name);
        return factory;
    }

    public ConnectionFactory getConnectionFactory(String name)
    throws NamingException {
        return (ConnectionFactory) context.lookup(name);
    }

    public Topic getTopic(String name) throws Exception {
        Topic topic = null;
        Object object = null;
        object = context.lookup(name);
        if (object != null) {
            topic = (javax.jms.Topic) object;
        }
        return topic;
    }

    public Queue getQueue(String name) throws Exception {
        Queue queue = null;
        Object object = null;
        object = context.lookup(name);
        if (object != null) {
            queue = (javax.jms.Queue) object;
        }
        return queue;
    }

    public Destination getDestination(String name)
    throws NamingException {
        return (Destination) context.lookup(name);
    }
}
```

`env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory")`

Specifies the name of the context factory class for WebSphereMQ.

**Note:** This example assumes that JNDI .bindings file is placed at a physical location in the FileSystem and thus is referred to as RefFileSystemContextFactory. This .binding file contains attributes that are used to create a ConnectionFactory object and other JNDI resources. For WebSphereMQ, the attributes to create connection factory resides in a .bindings file and for ActiveMQ, in the jndi.properties file.

```
env.put(Context.PROVIDER_URL, "file:/C:/JNDI")
```

Specifies the location of the JNDI .bindings file. For WebSphereMQ, the location of .bindings file and for ActiveMQ, the location of jndi.properties file.

```
env.put(Context.SECURITY_PRINCIPAL, "username");
env.put(Context.SECURITY_CREDENTIALS, "password")
```

Specifies the security credentials to access the MQService

**Note:**

- The jmsfromABL.AdminObjectFinder name is mandatory in the class file.
- The class and the get...() methods must be declared as a public element .

2. Compile the `jmsfromABL.AdminObjectFinder.java` class file and place it in a new JAR file.
3. Add the location of this new JAR file to the `pluginclasspath` and `classpath` variables in the `AdminServerPlugins.properties` file, as in the following example:

```
[PluginPolicy.Progress.SonicMQ]

pluginclasspath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar,
C:\AdminObjectFinder.jar

classpath=C:\Progress115\OpenEdge/java/progress.jar,
C:\PROGRA~2\IBM\WEBSPH~1\java\lib\com.ibm.mqjms.jar,
C:\AdminObjectFinder.jar

jvmargs=-DsonicMQExtensions=false -DjmsProvider=WebSphereMQ
```

# Updating your ABL-JMS API

The generic JMS adapter allows you to connect to SonicMQ, ActiveMQ, and WebSphereMQ JMS providers. JMS client connects to the JMS server differently for different JMS and thus needs different inputs as follows:

- SonicMQ connection requires the SonicMQ Broker URL such as `tcp://myhost:2506`
- ActiveMQ connection requires the ActiveMQ provider URL such as `tcp://sampleHost:61616`
- WebSphereMQ connection requires the QueueManager name as it is configured in the JNDI namespace.

As generic JMS adapter behaves as a proxy client for the JMS connection, the ABL API that creates connection with the JMS provider must provide an appropriate input for the provider.

If you wish to wish to migrate your applications from SonicMQ to ActiveMQ or WebSphereMQ, you must modify your ABL-JMS API to point to the appropriate MQ Broker URL as follows:

For WebSphereMQ:

```
RUN setBrokerURL IN psSessionH (INPUT "SampleQMgr").
```

where `SampleQMgr` is the name of the QueueManager or the connection factory for the queue or the topic that you want to use.

For ActiveMQ:

```
RUN setBrokerURL IN psSessionH (INPUT "myhost:61616").
```

where `61616` is the port at which the ActiveMQ Broker URL is listening.

# Non-supported API methods for Generic JMS Adapter

The classic ABL API specifically supports all the features supported by the SonicMQ JMS provider.

---

**Note:** For more information about the classic ABL API, see the *OpenEdge Development: Messaging and ESB* guide.

---

However, the other JMS providers do not support all the capabilities and methods in this API. Here is a list of capabilities that are not supported:

- FaultTolerant
- Client Persistency
- Load Balancing
- Multi-part messages
- Message selector
- Acknowledge And Forward
- Failover

- FlowToDisk

The generic JMS adapter does not support the following methods in Session objects:

- `createChangeStateConsumer procedure`
- `createMultipartMessage procedure`
- `createRejectedMessageConsumer procedure`
- `getClientPersistence function`
- `getClientTransactionBufferSize function`
- `getConnectionURLs function`
- `getFaultTolerant function`
- `getFaultTolerantReconnectTimeout function`
- `getFlowToDisk function`
- `getInitialConnectionTimeout function`
- `getLocalStoreDirectory function`
- `getLocalStoreSize function`
- `getLocalStoreWaitTime function`
- `getReconnectInterval function`
- `getReconnectTimeout function`
- `getSelectorAtBroker function`
- `getSequential function`
- `getSingleMessageAcknowledgement function`
- `isFaultTolerant function`
- `setClientPersistence procedure`
- `setConnectionFile procedure`
- `setConnectionURLs procedure`
- `setFaultTolerant procedure`
- `setFaultTolerantReconnectTimeout procedure`
- `setFaultTolerantReconnectTimeout procedure`
- `setInitialConnectionTimeout procedure`
- `setFlowToDisk procedure`
- `setLocalStoreDirectory procedure`
- `setLocalStoreSize procedure`
- `setLocalStoreWaitTime procedure`
- `setReconnectInterval procedure`

- `setReconnectTimeout procedure`
- `setPingInterval procedure`
- `setPrefetchCount procedure`
- `setPrefetchThreshold procedure`
- `setSelectorAtBroker procedure`
- `setSequential procedure`
- `setSingleMessageAcknowledgement procedure`

The generic JMS adapter does not support the following method in Message Consumer objects :

- `acknowledgeAndForward procedure`

The generic JMS adapter does not support the following methods in Message objects:

- `addMessagePart procedure`
- `getBytesPartByID function`
- `getBytesPartByIndex function`
- `getContentType function`