# JavaScript JMS Adapter

# **Overview**

Java Message Service (JMS) is the standard messaging Java API. With a JMS adapter, you can read and write messages from any messaging provider that supports the API.

WebSphere Application Server Liberty profile included with IBM MobileFirst™ Platform Foundation does not contain the built-in Liberty JMS features. JMS is supported by the WebSphere Application Server Liberty profile V8.5 ND (Network Deployment) server. Look for "Enabling JMS" in the documentation for WebSphere Application Server.

This tutorial covers the following topics:

- Creating the adapter
- Enable JMS in Server.xml
- Connection properties
- JMS API
- Configurations for external JMS providers
- Sample application

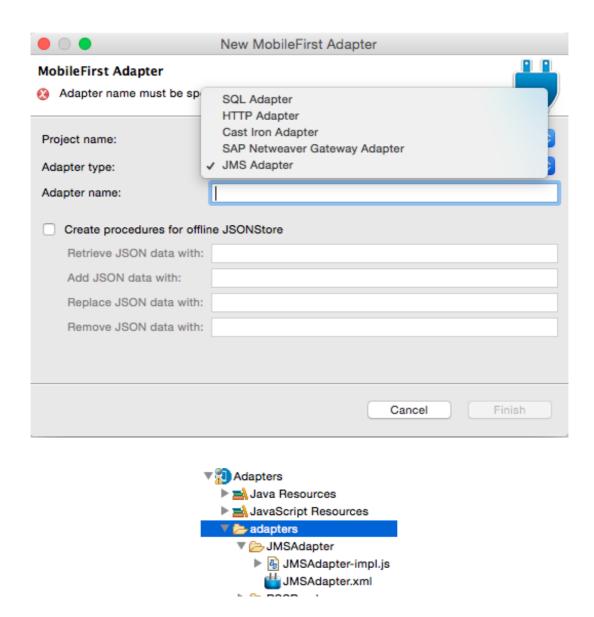
# Creating the adapter

#### CLI

From the project's directory, use mfp add adapter and follow the interactive instructions.

#### **Studio**

In MobileFirst Platform Foundation Studio, create an adapter and select the **JMS Adapter** type. A standard JMS Adapter structure is created.



Public procedures are declared in the XML file while implemented in the JavaScript file. The procedure name in the JavaScript file must match the one declared in the XML file.

#### **Procedure implementation**

Procedures are implemented in the adapter JavaScript file.

Procedure names in the JavaScript file must be the same as in the adapter XML file.

#### XML file

```
...
cprocedure name="writeMessage"/>
cprocedure name="readMessage"/>
cprocedure name="readAllMessages"/>
```

JS file

```
...
function writeMessage(messagebody) {
    ...
    ...
}
```

The destination parameter is the target for messages that are produced by the client, and the source for the messages that are used by the client.

```
function writeMessage(messagebody)
  var inputData = {
    destination: "dynamicQueues/MobileFirst",
    message:{
      body: messagebody,
      properties:{
         MY_USER_PROPERTY:123456
      }
    }
  };
  return WL.Server.writeJMSMessage(inputData)
;
}
```

#### **Enable JMS in Server.xml**

Enable JMS on your Liberty profile ND server.

```
</-- Enable features -->
<featureManager>
  <feature>jsp-2.2</feature>
  <feature>wasJmsServer-1.0</feature
>
  <feature>wasJmsClient-1.1</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

### **Connection properties**

Connection properties are configured in the adapter XML file.

- namingConnection Necessary only if you are using an external JNDI (Java™ Naming and Directory Interface) repository.
  - url The URL to the JNDI repository.
  - initialContextFactory The classname for the factory that is used for the configuration of JNDI properties.
  - user, password The credentials as set up by the JNDI administrator.
- jmsConnection

- connectionFactory The classname for the JMS connection factory that contains JMS configuration properties.
- user, password The credentials as set up by the JNDI administrator.

```
<wl>adapter name="JMSAdapter" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wl="
http://www.ibm.com/mfp/integration" xmlns:jms="http://www.ibm.com/mfp/integration/jms">
 <displayName>JMSAdapter</displayName>
 <description>JMSAdapter</description>
 <connectivity>
  <connectionPolicy xsi:type="jms:JMSConnectionPolicyType">
   <namingConnection
    url="tcp://9.148.225.169:61616"
    initialContextFactory="org.apache.activemq.jndi.ActiveMQInitialContextFactory"
    user="admin"
    password="admin"
    />
   <jmsConnection
    connectionFactory="ConnectionFactory"
    user="admin"
    password="admin"
    />
  </connectionPolicy>
 </connectivity>
 cprocedure name="writeMessage" />
 rocedure name="readMessage" />
 cprocedure name="readAllMessages" />
</wl></wl></wl></l></l></l></l></l>
```

Copy the relevant external libraries to the project for it to use JMS classes.

If you use **Apache ActiveMQ**, copy the activemq-all-activemq\_version\_number.jar file to the server\lib directory.

### JMS API

- WL.Server.readSingleJMSMessage Reads a single message from the given destination.
- WL.Server.readAllJMSMessages Reads all messages from the given destination.
- WL.Server.writeJMSMessage Writes a single JMSText message to the given destination.
- WL.Server.requestReplyJMSMessage Writes a single JMSText message to the given destination and waits for the response.

### readMessage

This method gets the next message from the destination.

It waits for timeout in milliseconds and returns a JMS message that contains the body and all available properties.

```
function readMessage() {
   var result = WL.Server.readSingleJMSMessage(
   destination: "dynamicQueues/MobileFirst",
    timeout: 60
   });
   return result;
}
```

Result:

```
{
  "isSuccessful": true,
  "message": {
  "body": "Hello World",
  "properties": {
  "JMSCorrelationID": null,
  "JMSDeliveryMode": 2,
  "JMSDestination": "queue:\/\/worklightQueue",
  "JMSExpiration": 0,
  "JMSExsageID": "ID:dhcp-9-41-62-243-50565-1350319511250-1:2:1:1:1",
  "JMSPriority": 4,
  "JMSRedelivered": false,
  "JMSRedelivered": false,
  "JMSTimestamp": 1350319533527,
  "JMSType": null,
  "MY_USER_PROPERTY": 123456
  }
}
```

#### readAllJMSMessages

This method takes the same parameters as the readSingleJMSMessage method. It returns a list of JMS messages in the same format as the readSingleJMSMessage method. The result is contained in a messages object.

To use this method, use an external server, **not** the one that MobileFirst Studio uses.

Result:

```
"isSuccessful": true,
"messages": [

{
    "body": "Hello World",
    "properties": {
        "JMSCorrelationID": null,
        "JMSDeliveryMode": 2,
        "JMSDestination": "queue:\/\/worklightQueue",
        "JMSExpiration": 0,
        "JMSMessageID": "ID:dhcp-9-41-62-243-50565-1350319511250-1:4:1:1:1",
        "JMSPriority": 4,
        "JMSRedelivered": false,
        "JMSReplyTo": null,
        "JMSTimestamp": 1350319659819,
        "JMSTimestamp": 1350319659819,
        "JMSTippe": null,
        "MY_USER_PROPERTY": 123456
}
}
```

### writeMessage

This method writes a JMSText message to the destination. It features user properties that can be set. It returns the JMSMessageID of the sent message.

```
function writeMessage(messagebody) {
var inputData = {
  destination: "dynamicQueues/MobileFirst",
  message:{
   body: messagebody,
   properties:{
     MY_USER_PROPERTY:123456
   }
  }
};
return WL.Server.writeJMSMessage(inputData)
;
}
```

Result:

```
(
"ID:414d51284d59514d20202020202020202755bb4f2000b602",
"isSuccessful": true
```

#### requestReplyJMSMessage

This method:

- Accepts the same parameters as the writeJMSMessage method.
- Writes a JMSText message to the destination.
- Waits for a response on a dynamic destination.
- Is designed for services that use the replyTo destination from the originating message.
- Returns a JMS message in the same format as the readSingleJMSMessage method.

## Configurations for external JMS providers

By using IBM MobileFirst Platform, you can configure access to several JMS providers. Configurations might vary depending on the selected provider.

When you work with an external JMS provider, check its documentation to learn how to implement it. Usually, such implementation requires that you copy JAR files to the server\lib directory of your MobileFirst project. Validate the URL and port.

# Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/JavaScriptAdapters/tree/release71) the MobileFirst project.

By using the attached sample, you can send and read messages to a JMS queue called MobileFirst. To run the sample, you need an external JMS library.