API Gateway JMS Appendices

Published: May 2013

Applies To
API Gateway 11.1.2.x

Contents

Appendix A : Message Type—API Gateway in Provider Mode

The Message Type option in the Messaging System filter enables the specification of the data type to be serialized and sent in the JMS message to the JMS provider. The option selected depends on what part of the message will be sent to the consumer. For example, to send the message body, select the option to format the body according to the rules defined in the SOAP over JMS recommendation. Alternatively, to serialize a list of name-value pairs to the JMS message, choose the option to create a `MapMessage`.

Serializations from API Gateway Message to JMS Message

When writing messages to the queue, the API Gateway extracts the content of the JMS message from a configured attribute. The following options are available:

- Use the content.body attribute to create a message in the format specified in SOAP over JMS: This results in a message placed on the queue that adheres to the SOAP over JMS recommendation (`http://www.w3.org/TR/soapjms/`). This is a `javax.jms.BytesMessage`, which is placed on the queue, and has a JMS property containing the content type (`text/xml`).
- Map a Java object stored in a message attribute on the whiteboard to the corresponding JMS message type: This results in a message placed on the queue where the JMS message type is dependent on the type of object stored in the named attribute. Use the following table as a guide:

Named attribute contains	Results in following JMS Message on the queue
`java.lang.String`	`javax.jms.TextMessage`
`byte[]`	`javax.jms.BytesMessage`
`java.util.Map`	`javax.jms.MapMessage`
`java.lang.Serializable`	`javax.jms.ObjectMessage`
`javax.jms.Message`	`javax.jms.Message`

Appendix B: Message Extraction Method—API Gateway in Consumer Mode

The Extraction Method field on the JMS Consumer screen includes the following possible options:

- Create a content.body attribute based on the SOAP over JMS draft specification: This assumes that a `BytesMessage` has been received, and one of the JMS properties contains a content-type field with a value of `text/xml`. This results in a `content.body` attribute being populated on the whiteboard of the API Gateway, and its value is a `com.vordel.mime.XMLBody`.
- Insert the JMS message directory into the attribute named below: This results in a `javax.jms.Message` being placed on the whiteboard. The expectation here is that a scripting filter is available in the policy that is called by the JMS consumer, which extracts information from the `javax.jms.Message` object.

- Populate the attribute below with the value inferred from message type to Java: This results in a Java object being placed on the whiteboard in the named attribute. The type of object placed on the whiteboard is determined by the following table:

API Gateway receives JMS Message of type	Attribute of the following type placed on whiteboard
`javax.jms.TextMessage`	`java.lang.String`
`javax.jms.BytesMessage`	`byte[]`
`javax.jms.MapMessage`	`java.util.Map`
`javax.jms.ObjectMessage`	`java.lang.Object`

When the JMS Consumer populates the whiteboard, it invokes the configured policy for processing the message. If the received JMS message contains a `ReplyTo` field, the API Gateway automatically sends the result of calling the policy to the source (queue or topic) in the `ReplyTo` field.

The JMS message sent to the ReplyTo source uses the reverse mechanism that was read from the queue. If the consumer reads a JMS message, and populates a named attribute with a value inferred from the message type to Java (`TextMessage == String`, and so on), when the policy completes, it looks for this attribute, and infers the JMS response message type based on the object type stored in the attribute.

Appendix C: Threading

Consumer Threads

By default, the API Gateway's JMS consumer has only a single thread listening for messages on the queue. If the volume of messages arriving at the queue is greater than a single thread can process, you can increase the number of threads listening on the queue by increasing the Listener Count to the required number of threads. This option is available in the JMS Session under the API Gateway process in the Policy Studio tree.

Producer Threads

When running without "Shared JMS session" option on the Messaging System filter each thread running in the API Gateway will establish a connection to the JMS server. By default there are a potential of 1024 threads that can process incoming messages, which means a possible 1024 connections to the JMS server. In order to limit the number of connections to a JMS server it can be done by reducing the number of threads that the API Gateway will use. To reduce the maximum number of threads edit the instance `service.xml` file located in the `groups/group-x/instance-y/conf` directory of the API Server installation and restarting the instance. (e.g. for 64 thread maximum)

```
    <SystemSettings tracelevel="INFO" secret="${secret}"
serviceID="${serviceID}" groupID="${groupID}"
serviceName="${serviceName}" groupName="${groupName}"
```

```
        domainID="${domainID}" title="API
Gateway"          maxThreads="64" <!-- add this
line -->      />
```

Appendix D: Session Context Cache

There is a maximum number of concurrent JMS sessions per JMS filter. By default, up to 8 sessions per filter are cached. The default can be changed by setting the environment variable V_JMS_CONTEXTCACHE_SIZE before starting the API Gateway. e.g.

Linux

```
export V_JMS_CONTEXTCACHE_SIZE=10
```

Windows `set`

```
V_JMS_CONTEXTCACHE_SIZE=10
```

Appendix E: Acknowledgement

AcknowledgeMode is set by the check box "allow duplicates" on the configured JMS session.
Example:

```
if true = Session.DUPS_OK_ACKNOWLEDGE

else Session.CLIENT_ACKNOWLEDGE)

DUPS_OK_ACKNOWLEDGE
```

This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the JMS provider fails, so it should only be used by consumers that can tolerate duplicate messages. Use of this mode can reduce session overhead by minimizing the work the session does to prevent duplicates.

```
CLIENT_ACKNOWLEDGE
```

With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method. Acknowledging a consumed message acknowledges all messages that the session has consumed.
When client acknowledgment mode is used, a client may build up a large number of unacknowledged messages while attempting to process them. A JMS provider should provide administrators with a way to limit client overrun so that clients are not driven to resource exhaustion and ensuing failure when some resource they are using is temporarily blocked.

Appendix F: Reconnection

API Gateway will only attempt to reconnect JMS during the following events

:1. At startup of the API Gateway if it fails to connect to the JMS queue
2. At deployment of a new configuration if it fails to connect to the JMS queue

Once a connection has been established (which happens in either of the above 2 events) it is then up to the JMS provider to reconnect.