

ActiveMQ JMS Event Gateway

Using the event gateway

This section covers the configuration file, the data structure passed into ColdFusion as a result of receiving a message and how to initiate messages from ColdFusion. The configuration file is covered last, since the other two pieces are much simpler.

Processing a JMS message in ColdFusion

When a message is received from the JMS server by the event gateway, the message data is passed to the `onIncomingMessage()` method in the configured gateway CFC as a single structure argument, traditionally called “event”. That structure contains the following keys:

- **data** – A structure that contains the message.
- **gatewayType** – A string that identifies the type of the event gateway: this is “ActiveMQ” for this JMS event gateway.
- **originatorID** – A string that identifies the destination from which this message was received, for example, the name of the queue or topic.

The data structure always contains the following metadata about the message:

- **msg** – The received message – either a string or a structure.
- **jmsmsg** – The raw JMS Message object.
- **properties** – Optionally, a structure that contains any message properties that were set on the message. Only present if at least one property was set by the originator. These are retrieved with `Message.getObjectProperty()` and stored in a `HashMap` that is passed in as this structure. ColdFusion dynamically translates the values to the appropriate types.
- **id** – Optionally, the JMS correlation ID of the message (a string). If no correlation ID was provided or generated, this key may be missing. You should test for its presence using the following:

```
structKeyExists(arguments.event.data, "id")
```
- **msgid** – Optionally, the JMS message ID of the message (a string). If no message ID was provided or generated, this key may be missing. You should test for its presence using the following:

```
structKeyExists(arguments.event.data, "msgid")
```
- **jmsession** – A Java object that can commit or rollback a message if transacted.
- **transacted** – A Boolean that specifies whether the message consumption was transacted or not. If transacted is true, the ColdFusion code should call the following:

```
arguments.event.data.jmsession.commit()
```


after the message has been fully processed, or call:

```
arguments.event.data.jmsession.rollback()
```


if processing failed in a way that would allow the message to be redelivered. If the ColdFusion code fails to call either of these methods, the consumer thread in the gateway times out after a specified period and automatically performs either a commit or rollback operation (which is configurable).

ActiveMQ JMS Event Gateway

The `onIncomingMessage()` method must return a structure. That structure must contain a **status** key, with the value **OK** for successful message processing, or **EXCEPTION** for failure. Alternatively, the returned structure may have a **status** equal to **SEND**, along with all the appropriate data; it is handled as if you called `sendGatewayMessage()` to send a JMS message through this gateway instance.

Message Data

If the message was sent as a `TextMessage` or a `BytesMessage`, `arguments.event.data.msg` is a ColdFusion string that contains the text of the message. A `BytesMessage` is treated as UTF-8 character data but the result is a ColdFusion string.

If the message was sent as a `MapMessage`, `arguments.event.data.msg` is a ColdFusion structure that contains the keys and values from the map. The values are raw objects, because ColdFusion can translate them to appropriate values.

Check `isSimpleValue(arguments.event.data.msg)` to see if the message is a string. Check `isStruct(arguments.event.data.msg)` to see if the message is a map.

If the message was sent in any other format, `arguments.event.data` does not have the `msg` key. In all cases, `arguments.event.data.jmsmsg` is the raw message object and can be used to retrieve the data.

Transacted Message Consumption

The event gateway architecture is inherently asynchronous. When a message is received from the JMS server, it is placed in a queue inside ColdFusion and processed by a pool of threads that invoke `onIncomingMessage()`. If the queue is full, the event gateway automatically rolls back the message consumption if it is configured to be transacted, otherwise excess messages are simply discarded. The queue size can be configured in the ColdFusion Administrator and can be extremely large (several hundred thousand events); even with non-transacted consumption, you can ensure that losing messages is an extremely rare edge case.

In the transacted consumption case, the ColdFusion code in `onIncomingMessage()` is responsible for committing or rolling back the message consumption, based on the success or failure of the operation. If the message is successfully consumed and processed, the code should commit the consumption. If the message cannot be successfully consumed and processed, the code needs to decide whether to commit the consumption or roll it back. If the message itself was valid but processing failed due to some external failure that is likely to be transient, such as a database connection error, the consumption can be rolled back so that message delivery is retried. If the message is invalid, there is no point in retrying the delivery; instead, you must take some error action (saving the message to disk, logging the error) and then commit the consumption so that the erroneous message is not redelivered.

For transacted message consumption, the event gateway creates a pool of consumers, each of which creates a single JMS session and receives one message at a time, handshaking with the ColdFusion code that performs commit or rollback operations. After a commit or rollback, the next message is received and passed to ColdFusion. The size of the consumer pool can be configured per gateway instance in the configuration file specified for that gateway instance. The default size is 10. If a message is rolled back,

ActiveMQ JMS Event Gateway

the JMS server may attempt redelivery of the message up to a configured number of times before giving up and discarding the message. With a default ActiveMQ server configuration, this is seven delivery attempts, after which the message is placed in the Dead Letter Queue. You use the **providerURL** configuration setting to change this.

If the ColdFusion code fails to commit or rollback a transacted message consumption, the gateway automatically performs a commit or rollback operation, depending on the configuration file settings, after a specified timeout period. This prevents threads from being blocked due to bad ColdFusion code. The default timeout is 60 seconds and the default action is to rollback to allow redelivery attempts.

Sending a JMS message from ColdFusion

Any ColdFusion page can send a JMS message using the `sendGatewayMessage()` function. This function takes two arguments:

- The full name of the configured event gateway, for example, `SampleAMQGateway`.
- A structure that contains the message and metadata about the message:
 - **status** – Must be set to `SEND`.
 - **id** – Optionally, a string that uniquely identifies the message. This value is used as the JMS correlation ID on the message sent to the server.
 - **msgid** – Optionally, a string that uniquely identifies the message. This value is used as the JMS message ID on the message sent to the server (and may be ignored by the server: ActiveMQ always generates its own message ID values).
 - **queue** or **topic** – One, and only one, of these keys must be present. A string that specifies the destination to use on the ActiveMQ server.
 - **asBytes** – Optional Boolean that specifies whether the message data should be treated as raw UTF-8 bytes. Acceptable values are: **yes**, **no**, **true**, **false**. The default is **no** – the message is a regular text string.
 - **message** – A string or structure that provides the data of the message to send. If **message** is a string, the optional **asBytes** metadata determines whether to send the data as a `TextMessage` or `BytesMessage`. If **message** is a structure, the data is sent as a `MapMessage` with all of the values in the structure converted to strings in the map. The conversion is done so that predictable types are passed over the wire, for example, you cannot send a ColdFusion date directly, so it is converted to a string. As with **properties** and **propertyTypes**, if the map keys have to have specific case-sensitive names, you must use bracket notation to create the structure instead of dot notation.
 - **properties** – Optionally, a structure that contains message property name/value pairs to add to the message before it is published. This data can be used by message selectors for consumers or within the consumer itself.
 - **propertyTypes** – Optionally, a structure that contains message property name/type pairs. If a property is specified without a matching property type, it is assumed to be a string. Valid property types are: **Boolean**, **byte**, **double**, **float**, **int**, **long**, **short**, **string**. If an invalid type

ActiveMQ JMS Event Gateway

is specified, the corresponding property is assumed to be a string. Property types are case sensitive and should be all lowercase. Property names are all uppercase if you use dot notation: `properties.someProp = 42` creates a property called SOMEPROP. To specify mixed case (case sensitive) property names, use bracket notation: `properties["someProp"] = 42`. The name of the property must exactly match in the **properties** structure and the **propertyTypes** structure, otherwise the property is assumed to be a string.

The gateway configuration file

The sample configuration files are heavily commented and should be self-explanatory, however, this section describes each setting in more detail. The sample configuration file is located in:

```
WEB-INF/cfusion/gateway/config/
```

There are some settings that apply to all gateways and some that apply only if a gateway is configured as a subscriber. (A gateway can be configured as outbound only, purely as a publisher). The generally applicable settings are described first, and then use case specific settings are described. All settings are case-sensitive; both their names and their values must be spelled exactly as shown below. Acceptable values for Boolean settings are: **yes**, **no**, **true**, **false** and must be lowercase. In general, invalid values are treated as the default value where appropriate.

providerURL – string, required. This is the URL of the JMS provider, that is, the server.

- For ActiveMQ, this is something like `tcp://localhost:61616`. ActiveMQ lets you override a number of configuration parameters using a query string, for example, **providerURL=tcp://localhost:61616?jms.redeliveryPolicy.maximumRedeliveries=20**
- For JRun, it might be just `localhost:2908`.
- For other servers, it might be a full URL such as <http://server.com:1234>.

initialContextFactory – string, required. This is the name of the class used to construct the initial JNDI context that is used to lookup the connection factory and destination.

- For ActiveMQ, it would be `org.apache.activemq.jndi.ActiveMQInitialContextFactory`.
- For JRun, it would be `jrun.naming.JRunContextFactory`.
- For FioranoMQ, it would be `fiorano.jms.runtime.naming.FioranoInitialContextFactory`.

connectionFactory – string, required. This is generally the name of the class used to construct the factory object that creates connections. For some providers, it is a c-name identifying the resource within some sort of directory, such as LDAP.

- For ActiveMQ, it is `ConnectionFactory` (a dummy name because ActiveMQ provides only a stub JNDI context).
- For JRun, it is `jrun.naming.JRunContextFactory`.
- For FioranoMQ, it is a full c-name.

ActiveMQ JMS Event Gateway

debug – Boolean, optional, default **no**. If **debug=yes**, the gateway generates verbose logging. The default is to only log problems.

username – string, optional. If both **username** and **password** are present, this username is used when the connection is created (i.e., in the call to `createTopicConnection()` or `createQueueConnection()`).

password – string, optional. See **username** above.

topic – boolean, optional, default **yes**. If **topic=yes**, the gateway expects the destination name to refer to a topic. If **topic=no**, the gateway expects the destination name to refer to a queue.

cachable – Boolean, optional, default **no**. If **cachable=yes**, a single publisher connection is created and used for all outbound messages. This allows you to use vendor-specific settings that support long-lived connections, such as durable publishers. Those settings are provided via the **contextProperties** setting. If **cachable=no**, a fresh publisher connection is created for each outbound message that is sent.

contextProperties – string, optional. If present, this is a comma-separated list of additional property names within the configuration file that are read by the gateway. This is a way to get vendor-specific properties into the initial JNDI context used for looking up the connection factory and the destination. For example:

```
contextProperties=foo,bar  
foo=42  
bar=someValue
```

This causes the properties **foo** and **bar** to be added to the initial JNDI context with the specified values. You can add any number of additional properties in this manner.

Because ActiveMQ does not have a full JNDI implementation, you must use this configuration setting to populate the JNDI context for destination name lookup. For example, if you want to be able to lookup queues called “Report” and “Statistics”, you must define local queue name properties that maps to those values as follows:

```
contextProperties=queue.rpt,queue.stats  
queue.rpt=Report  
queue.stats=Statistics
```

You then specify destination names of “rpt” and “stats”. The JNDI lookup resolves those names to the real names. If you want to look up a topic name, use the prefix “topic.” instead of “queue.” in the property names. ActiveMQ also supports dynamic queues and topics. For those, you can simply specify the **destinationName** as either “dynamicQueues/Report” for a dynamic queue called “Report” or “dynamicTopics/Statistics” for a dynamic topic called “Statistics”.

outboundOnly – Boolean, optional, default **no**. If **outboundOnly=yes**, the gateway does not subscribe to a destination and the associated CFC is essentially ignored. You can use a single outbound gateway for all publishing destinations if **cachable=no**, or you can send outbound messages via any number of gateways. If **cachable=yes**, you need one gateway for topics and a separate gateway for queues and you should use just those gateways for sending messages.

If **outboundOnly=yes**, the gateway does not subscribe to a destination and the following options are ignored because they only apply to message consumption.

ActiveMQ JMS Event Gateway

destinationName – string, required for message consumption. This is the full name of a message destination to which the gateway should subscribe. This is not needed for outbound messages because the destination name is provided for each message. If the destination name is a full LDAP-style c-name, the gateway also constructs a short destination name, which is the base c-name converted to lowercase; everything after the first comma in the full c-name is discarded.

selector – string, optional. This is a valid message selector expression that is applied to the subscriber (or consumer) to filter incoming messages. Only messages that match the selector are processed by the gateway.

noLocal – Boolean, optional, default **no**. If **noLocal=yes**, try to inhibit delivery of any messages that that originate from this gateway to this gateway. It is only used for topic subscribers where a message selector is in place. For queue receivers, it is ignored. For topic subscribers with no message selector, it is ignored.

transacted – Boolean, optional, default **no**. If **transacted=yes**, a pool of consumers is created that handshake with the ColdFusion code to create JMS transactions on separate sessions within the gateway's connection. ColdFusion code must call **commit()** or **rollback()** to complete a transaction. See Transacted Message Consumption, above.

poolSize – integer, optional, default **10**. If **transacted=yes**, this determines the size of the consumer pool. See Transacted Message Consumption, above.

transactionTimeout – integer, optional, default **60**. If **transacted=yes**, this determines the number of seconds that a thread in the consumer pool waits for ColdFusion code to perform a commit or rollback operation. If a transaction times out under this mechanism, the gateway automatically performs a commit or rollback operation – see **actionOnTimeout**.

actionOnTimeout – string, optional, **rollback**. If **transacted=yes** and a transaction times out, this defines the action that the gateway automatically performs. Acceptable values are **rollback** and **commit**.

durable – Boolean, optional, default **no**. If **durable=yes**, the gateway establishes a durable connection to the JMS provider. Messages should be queued on the JMS server whenever this uniquely named consumer is not running. If **durable=no** and no consumer is present, messages are generally discarded by the provider.

subscriberName – string, optional. If **durable=yes**, this can be used to specify a unique name for this gateway's message consumer. If this setting is omitted, a default subscriber name is constructed from the short destination name, with the prefix **sub_**. If **durable=no**, this setting is ignored.

publisherName – string, optional. If you are using vendor-specific durable publishers, this can be used to specify a unique name for this gateway's message producer. If this setting is omitted, a default publisher name is constructed from the short destination name with the prefix **pub_**.