Sterling Integrator®

Using JMS Services and Adapters

Version 5.1

Sterling Commerce

An IBM Company

Contents

JMS 1.1 Overview

For Sterling Integrator 5.1, the JMS 1.1 specification is supported. JMS is a messaging standard that allows application components based on the Java 2 Platform Enterprise Edition (J2EE) to create, send, and receive messages.

With JMS, you can either use the Queue or Topic method to store and deliver messages. In the service and adapters, this is called the Destination Type.

Queues are First In First Out (FIFO) storage mechanisms for messages. Messages in the queue are:

- Pushed into and pulled out of the queue.
- Bound for a single consumer.
- Available until removed from the queue.

Queue types include:

- Priority Queues - Order of consumption of messages from the queue is not order of insertion, but based upon the priority set in the header.
- Temporary Queues - Default queue setting, messages are not persisted and only kept in memory. Failure of the server will cause the messages in the queue to be lost.
- Persistent Queues - Messages are persisted and will survive a server failure.
- Exclusive verses Non-Exclusive Queues - Exclusive Queue is for one consumer pulling messages from a queue. Non Exclusive Queues allows for multiple consumers to pull from a queue.

Topics are:

- A message is delivered to potentially 0 or more consumers.
- A message is available only for as long as it takes to deliver the message to all subscribed consumers. Consumers for topics are durable subscribers. A durable subscriber has a mailbox where messages will be stored so that messages are not lost even if the consumer is not listening at the time a message is published on the topic.
- Topics have hierarchy, which includes Subject and Subscriptions.

JMS 1.1 Services and Adapters

The following JMS 1.1 services and adapters are available in Sterling Integrator 5.1:

- JMS 1.1 Acquire Connection Service - used to open a connection and session with a remote JMS server

- JMS 1.1 Send Message Service - used to send messages
- JMS 1.1 Receive Message Service - used to get messages synchronously from a queue or topic
- JMS 1.1 Request Reply Service - used in scenarios where the response from user matters to the sender and depending on this response an action needs to be taken by the sender
- JMS 1.1 Release Connection Service - used to release the connection and session
- JMS 1.1 Async Receive Adapter - used to enable clients to receive messages in async mode

What's New in Sterling Integrator 5.1

Sterling Integrator 5.1 now supports:

- SSL communication
- Request Reply pattern
- Custom Class Loading
- Message Selectors
- Durable Subscribers
- Transactions in Async mode
- Pooling of JMS connections and sessions which play a significant role in enhancing performance
- Ability to use ReplyTo header of the JMS message more efficiently when replying to received messages
- JMX Monitoring and Management Console which monitors and manages connection pools

JMS 1.1 Property File Information

The JMS 1.1 property file is named jms11.properties. You can find it in the /*install_dir*/install/properties directory. The following are the property names and definitions.

Property Name=Default Value	**Definition**
DEFAULT_TIME_TO_LIVE=300000	Specifies the default time to live per connection. Default it is 300000 milliseconds. This is the same as time to live when defining the pools. If TIME_TO_LIVE is specified at service or adapter instance level then that value will be used. This applies for max sessions, max connections, and receive timeout.
DEFAULT_MAX_SESSIONS=500	Specifies the default number of sessions per connection. This is the same as number of sessions when configuring a pool.
DEFAULT_MAX_CONNECTIONS=1	Specifies the default number of connections. This is the same as number of connections when configuring a pool.
JMS_RECEIVE_TIMEOUT=300000	Specifies the amount of time sync receive will wait for message before giving up. If no timeout value is specified in the JMS 1.1 Receive Message Service, this value is used. Default is 5 minutes (300000 milliseconds).
JMS_MSG_BUFFER_SIZE=30000	Specifies the buffer size used when creating a BufferedInputStream and BufferedOutputStream. It is used to read or write documents to message or vice versa. Buffer size is dependant on the amount of memory you have available for JVM.
JMS_REQUEST_REPLY_TIMEOUT_DEFAULT=120000	Specifies the amount of time request reply receive will wait for message before giving up. Default is 2 minutes

Property Name=Default Value	Definition
	(120000 milliseconds). This is valid only when using permanent destination mode. If no timeout value is specified in the JMS 1.1 Request Reply Message Service, this value is used. For temporary destinations, receive will wait for an infinite time.
JMS_EXCLUDES	Specifies the list of classes to exclude. This is a semicolon separated list. Many of the JMS drivers include the javax.jms package, which is already loaded in the classloader. To avoid classloading/class conflicts, you only want to load the vendor specific jars from the vendor provided client package. To do this you need to exclude various packages which this feature provides. Ideally when using custom class loading, you should avoid loading any classes that are already present in the system's dynamic classpath (stated in dynamiclasspath.cfg). For most of the JMS vendors this means avoiding classes that start with package structure javax.jms, but in several cases there are other classes that can be present in both vendor jars and the system's dynamic classpath. To avoid class path conflicts, a generic javax is specified in the excludes list. This parameter is only valid when using the custom class loading and is per adapter instance. Be careful when changing this parameter. For example, if all the classes under the javax package need to be excluded from being loaded by custom class loader then this property should be specified by providing value as javax (JMS_EXCLUDES=javax).
JMS_REQUEST_REPLY_RETRIES=-1	Specifies the number of reply attempts made if the JMS provider has gone down. Default value is -1 (infinite) and is valid only for permanent destination mode. Minimum is 0 and maximum is infinite. It is recommended not this value is not changed.
JMS_REQUEST_REPLY_SLEEP_TIME=60000	Specifies the time between each of the retry attempts. Default is 1 minute (60000 milliseconds).

Supported Vendors and Configurations for JMS 1.1

The follow vendor configurations are supported in Sterling Integrator 5.1:

JMS Vendor	Supported Versions	JNDI	NON-JNDI	SSL	JDK Supported for SSL
ActiveMQ	5.2.0	Yes	Yes	Yes	IBM, SUN
WebLogic	9.2, 10.1	Yes	No	Yes	SUN
TIBCO EMS	4.4.1, 5.1.4	Yes	Yes	Yes	SUN
JBOSS		Yes	No	No	IBM, SUN
SonicMQ		Yes	Yes	No	IBM, SUN
WebsphereMQ	6.0, 7.0.1	Yes	No	Yes	IBM, SUN

See *System Requirements* for additional information JMS Vendor supported versions.

JMX Monitoring and Management Console

The Java Monitoring and Management Console, known as the JMX, can be used to monitor the following JMS 1.1 items:

- JMS Pools created by JMS 1.1 Acquire Connection and Session service.
- JMS Pools created by JMS 1.1 Async Receive Adapter.
- JMS Request Reply Async receivers created by JMS 1.1 Request Reply Service when used in Permanent Destination mode.
- Destinations cached.

JMX Monitoring and Management Console: Frequently Asked Questions

How can I use the JMX to monitor JMS 1.1 Pools?

You can use the JMX to monitor pools created by the Dynamic Class Loading or Custom Class Loading. The JMS 1.1 exposes the Monitor and Segmented Pool Monitor MBeans. The JMX supports the following pool operations:

Pool Operation	Description/Usage
listPools	Lists all the JMS pools created. If the same operation is selected from Segmented Pool Monitor, it lists all the pools created using Custom Class Loading.
stopPool	Used to shutdown any pool by providing the pool name. This can be used in addition to shutting down pools from JMS 1.1 Release Connection and Session service.
getPoolDetails	Used to get more information about the pool by providing the pool name.

How is JMX helpful in dealing with Request Reply when using Permanent Destination mode?

JMS 1.1 Request Reply service uses an async receiver when used in Permanent Destination mode. When used in Permanent Destination mode, the JMS 1.1 Request Reply Service exposes an MBean for monitoring various parameters. The following JMX operations can be useful in monitoring the JMS 1.1 Request Reply Service:

JMX Operations	Description/Usage
listContainers	A new async receiver is created when using JMS 1.1 Request Reply Service in Permanent Destination mode, when certain parameters are changed. These various async receivers are known as containers and can be seen from either

JMX Operations	Description/Usage
	the drop down in JMS 1.1 Release Connection and Session service or by invoking this method.
stopContainer	If you want to stop any of these async receivers, you can do this by providing the container name and invoking the stopContainer method. This can also be achieved by using the JMS 1.1 Release Connection and Session service.
pendingRequestCount	If you want to know the number of requests that are awaiting replies. If the Permanent Destination mode is the same destination used to receive replies for every request, replies are marked pending until the time a reply is received.
pendingRequestCorrelationIDs	If you want to know correlation ids for the pending requests.

How can I use the JMX to monitor JMS 1.1 Async Receive adapters?

The JMX can be used to listen for notifications sent by JMS 1.1 Async Receive adapters. If the retry is specified for JMS 1.1 Async Receive adapter and if an adapter goes down, then the JMX notifications are broadcasted which can be subscribed to.

What is that Destination Monitor I see in JMX screen?

MBeans can be used to monitor all the destinations cached by various JMS 1.1 services. If you are using the JNDI lookup for destinations, then the destinations are cached to avoid the JNDI lookup every time. If needed, these destinations can be cleared from the cache using this MBean.

JMS 1.1 Pools

JMS 1.1 Pool Parameters

There are various parameters that control the JMS 1.1 pools. These include pool name, number of connections, number of sessions, time to live per connection.

Other pool parameters include URL, ContextFactory, connectionfactory, cacertid, keycertid, jmsprovider, clientid, jndi username, jndi password, jar location, SSL mode, JNDI/non JNDI mode.

The pool parameter definitions are included in the adapter and service documentation.

JMS 1.1 Pool Creation

If a business process is using the JMS1.1 connection pool and is created during the first business process run, the same pool can be used by other JMS business processes by using the same JMS 1.1 Acquire Connection and Session service.

The JMS 1.1 Acquire Connection and Session service is responsible for creating the JMS pool. To fully utilize the JMS pool, you should share the same JMS 1.1 Acquire Connection and Session service (when connecting to the same server using the same parameters).

When using JMS 1.1 Async Receive Adapter, a JMS pool gets created as soon as an adapter instance is started. All the parameters that affect the pool remain the same in the services, but in the case of an adapter, the pool has always one connection and as many sessions as the number of concurrent consumers. The pool name is the same as the name of the adapter instance_<some unique id>. If you shutdown the adapter, it does NOT shutdown the pool. The pool has to be manually shutdown. For services, where the pools can be shared by sharing the same JMS 1.1 Acquire Connection and Session service across business processes, the pools can not be shared between adapters.

As JMS pools ensure high performance, the pools are not shutdown after the business process completes. You need to manually shutdown the pools, when the connection to the JMS provider is no longer required.

JMS pools can be checked using the JMX or using the POOL_LIST or the SEGMENTED_POOL_LIST in the JMS 1.1 Release Connection service. The pools listed in the POOL_LIST were created using the Sterling

Integrator dynamic class path, while the pools listed in the SEGMENTED_POOL_LIST were created using the custom class loader.

JMS 1.1 Pool Parameters Modification

Review the following before you edit pool parameters:

- When you edit the pool parameters in the JMS 1.1 Acquire Connection Session service or the JMS 1.1 Async Adapter, a new pool gets created. This ensures that you do not end up editing an existing JMS pool which is already in use.
- A pool, in the case of services or adapters, is represented as poolName_<unique id>. If you edit a pool parameter, but not the pool name, you might end up seeing same poolname_<different unique id> multiple times. Whenever you edit pool parameters, you should also change the pool name.
- If you shut down the adapter, the pool is not shutdown. The pool has to be manually shutdown.
- Pools can not be shared between adapters.
- For services, pools can be shared by sharing the same JMS 1.1 Acquire Connection and Session service across business processes.
- For adapters, you should disable the adapter, shutdown the pool and then edit the parameters and re-enable the adapter to avoid seeing multiple pools with same name.

JMS 1.1 Pool Shutdown

The JMS1.1 pools created by the JMS 1.1 Acquire Connection Session service or the JMS Async adapter can be shut down by:

- Running a business process and specifying the pool name
- Using the JMX Monitoring and Management Console

Business Process Example: Shutdown a Pool

A simple business process containing the following services can be used to shut down JMS pool.

Start > JMS 1.1 Release Connection And Session Service > Stop

For example, select the pool from POOL_LIST or SEGMENTED_POOL_LIST (if the pool was created using custom class loading) and select the release action as SHUTDOWN POOL in JMS 1.1 Release Connection and Session service.

The shutdown in SEGMENTED_POOL_LIST can also be used if you plan to just change the custom class loading jars in the folder without configuring a new instance. Stop any adapter services using this pool, shutdown this pool and the put the new jars in the location.

JMS 1.1 Pool Best Practices

Consider the following best practice information when creating and editing your JMS pools:

- When editing the pools, you should shut down the old pool if it is not being used anymore.
- When editing the pool, you can change the pool name along with other parameters if you are not sure whether to shut down the pool.
- Set the number of connections and sessions as needed.
- When editing the pool JMS 1.1 Async Adapter parameters, always shutdown the adapter first and then shut down the pool to avoid confusing pools with same name getting created.

JMS 1.1 Pools Frequently Asked Questions

When should you shutdown the JMS 1.1 pools?

You should shutdown a pool:

- Shutdown the pool when it is not in use any more. If you are using JMS 1.1 services only, the business process can shutdown the pool once all the business processes have finished and you will not be run the business processes. You will want to disconnect from the JMS provider.
- If you are using the adapters, once you have shutdown the adapter instance and if you do not plan to use it any time soon and want to disconnect from the JMS provider, you can shutdown the pool.
- If you plan to edit any of the pool parameters and none of the business processes or adapters are using the pool, you can shutdown the pool before editing parameters, as this is going to result in a new pool. Or you can change the pool name. This results in creating a new pool with a different pool name rather than same pool name. You can shutdown the old pool at a later point of time.

How are JMS 1.1 adapter pools impacted when the JMS provider shuts down?

- If the JMS 1.1 adapter using a pool is enabled - If an exception occurs, the adapter and pool try to reconnect based on the retry logic. If successful, then it will re-enables or shutdown. If the pool connection is shutdown, it will need to be manually cleaned up. Otherwise, if the JMS provider is restarted and the adapter is enabled, then the pool will refresh.
- If the adapter has been shutdown - If the adapter has been shutdown and the pool is still connected (if the pool has not been manually shut down) an exception occurs or the providers goes down, then the pool will be shutdown in the back ground but will still remain active, even though in dead state. If the provider is up and the adapter is re-enabled then the pool will become active again.

How are JMS 1.1 service pools impacted when the JMS provider shuts down?

For JMS 1.1 services, if an exception occurs at provider’s side or if the provider gets shutdown, the pool is shutdown, and remains in dead state (which can be manually cleaned up). Once the JMS provider is up, you can run the business process and the pool will become active again.

JMS 1.1 SSL

Using SSL with JMS 1.1 Async Receive adapter or the JMS Acquire Connection and Session service

SSL is a feature of the JMS 1.1 Async Receive adapter or the JMS Acquire Connection and Session service. You can either choose:

- Server Authentication mode (server gets authenticated) - only one sided handshake happens. During the handshake, the server sends it's certificate and it gets authenticated against the public key already checked into the system.
- Client Authentication mode (first the server gets authenticated and then the client gets authenticated by the server) - public part of the system certificate is also checked into the trusted store of the server so that when server wants to authenticate the client, it can do so against this already present public key.

To use SSL, you will need to know the following information when configuring the adapter:

- JMS Provider name (WebLogic, Websphere, ActiveMQ, or TIBCO)
- System Certificate name
- CA Certificate name

If you are using a JMS Provider that is not listed, then you will need to go back and select the Non-SSL mode.

In addition, if you are using a business process for the SSL for send and receive sync operations, you can do so by setting the following properties:

- SSL_SETTING_ssl_option=SSL_MUST
- JmsProviderName="providername"
- SSL_SETTING_ca_cert_ids="certificatename"

JMS 1.1 SSL Configuration for ActiveMQ

ActiveMQ supports both JNDI and Non-JNDI modes of operation. SSL is also supported for both the modes. The following jars are required for ActiveMQ version 5.2.0:

- DynamicClassPath: activemq-core-5.2.0.jar

• Using Segmented ClassLoader : activemq-core-5.2.0.jar

If you need additional information on ActiveMQ as an SSL provider and configuration set up on the server side, see the vendor documentation at:

//activemq.apache.org/how-do-i-use-ssl.html

JMS 1.1 SSL Configuration for TIBCO

For TIBCO 4.2.1, you can have both JNDI and Non-JNDI. You need the following jars in the dynamic classpath in this order:

1. tibjms.jar
2. tibcrypt.jar
3. tibjmsadmin.jar
4. tibjmsapps.jar

For TIBCO 5.1, you can have both JNDI and Non-JNDI. The following jars should be in the dynamic classpath in this order:

1. tibjms.jar
2. tibcrypt.jar
3. tibjmsadmin.jar
4. tibjmsapps.jar
5. slf4j-api-1.4.2.jar
6. slf4j-simple-1.4.2.jar

You can keep the jars in a folder and use jar Location field to use this folder as segmented jar location.

You can only configure TIBCO with the SUNJSSE. IBMJSSE is not supported.

JMS 1.1 SSL Configuration for WebLogic

Sterling Integrator supports the thin client implementation of SSL for WebLogic. You must use the thin client jars of WebLogic for SSL communication.

For WebLogic 9.2 and WebLogic 10.1:

• Supported modes of operation: JNDI
• Jars Required in Dynamic Class Path: While using Weblogic 9.x version, you must keep the following jars and in the same sequence - wlclient.jar, wljmsclient.jar. If you try WebLogic SSL with full jars like weblogic.jar or wlfullclient.jar, it will fail. The implementation has been done for thin client which uses different classes and rely on the CORBA implementation by the JDK. If you want to keep the fullclient jar make sure it comes after the above mentioned jars.
• For WebLogic 9.x, provide path to the directory containing the two jars wlclient.jar and wljmsclient.jar if you are using the Segmented Class Loader.

For WebLogic 10.1, you need to add the following jars path to the dynamic classpath or for the Segmented Class Loader:

- wlclient.jar
- wljmsclient.jar
- wlfullclient.jar

The following are some of the WebLogic SSL limitations:

- SSL can only function on SUNJSSE. IBMJSSE is not supported.
- When using the thin client, sometimes you get CORBA exceptions in the logs. This happens when the client is connected to the server but is idle for quite sometime. You can avoid this by setting the following property in your JVM: com.sun.CORBA.transport.ORBTCPReadTimeouts. Or you can set it as an option in your run script: -Dcom.sun.CORBA.transport.ORBTCPReadTimeouts= 1000:600000:180000:2000
- WebLogic by default caches the sessions. Once the successful handshake is done, the next wrong handshake will result to success.
- When using WebLogic 10.x, change the JMS_EXCLUDES property in the /install_dir/install/properties/jms11/jms11.properties from JMS_EXCLUDES=javax. to JMS_EXCLUDES=javax.jms

In addition, for NON-SSL scenarios using WebLogic 10.x , you only need to keep the wlfullclient.jar in the classpath (using custom classsloading or not using custom classloading).

JMS 1.1 SSL Configuration for WebSphereMQ

If you want to use WebsphereMQ with SSL, you need to create a connection factory with the SSL parameters defined in it .This connection factory will be used to do a look from JMS 1.1 Adapter. If you need information on how to configure the WebsphereMQ, see the IBM Websphere documentation on how to create connection factories.

While configuring SSL for IBM WebSphereMQ 6.0 the only mode of operation is JNDI. You need to have the following jars in the classpath in this order:

1. com.ibm.mq.jar
2. com.ibm.mqjms.jar
3. fscontext.jar
4. connector.jar
5. dhbcore.jar
6. jta.jar
7. providerutil.jar

The mq.jar needs to be before mqjms jar in Dynamic Classpath.

While configuring SSL for IBM WebSphereMQ 7.0, the only mode of operation is JNDI. You need to have the following jars in the classpath in this order:

1. fscontext.jar
2. com.ibm.mqjms.jar
3. com.ibm.mq.postcard.jar
4. jms.jar
5. com.ibm.mq.jar
6. com.ibm.mq.headers.jar

7. com.ibm.mq.jmqi.jar
8. com.ibm.mq.defaultconfig.jar
9. providerutil.jar
10. com.ibm.mq.commonservices.jar
11. com.ibm.mq.pcf.jar
12. com.ibm.mq.tools.ras.jar
13. dhbcore.jar

Additional information:

- When using Websphere MQ, the com.ibm.mq.jar needs to be added to the dynamic classpath of SI using install3party script. This needs to be done for both custom class loading and the Sterling Integrator dynamic class loading.
- Also for Websphere MQ 7.X, if the SSL does not work properly, you can provide the Websphere 6.X jars.
- The same jars can be used in custom class loading or segmented classloader for WebSphereMQ 6.0 and WebSphereMQ 7.0.

JMS 1.1 SSL Frequently Asked Questions

Which JDKs are supported for Sterling Integrator for the JMS providers?

Due to the JSSE limitation from some of the providers, this SSL feature is limited to the following vendors:

- ActiveMQ: IBM JDK and SUN JDK
- WebSphereMQ: IBM JDK and SUN JDK
- WebLogic: SUN JDK
- Tibco EMS: SUN JDK

Which providers are supported?

The SSL feature is supported for four providers:

- ActiveMQ 5.2.0
- WebSphereMQ 6.0 and 7.0.1
- Weblogic-9.2 and 10.1
- TIBCO EMS 4.4.1 and 5.1.4

What happens when the server goes down ? Does it refresh the SSL handshake? How does it affect a user?

Users are not affected in any way. Information is provided in the service and adapter documentation on what happens when the server goes down.