IBM WebSphere MQ

This section describes how to setVirtualize up for accessing IBM MQ. Access to IBM MQ is achieved by configuring Parasoft tools.

Sections include:

IBM WebSphere MQ Java API versus the standard JMS API

This section refers to testing services over the IBM WebSphere MQ Java API, not the standard JMS API. If you intend to send and receive messages from IBM WebSphere MQ using the JMS API, select the JMS transport option and refer to JMS.

Adding Necessary Jar Files to the Classpath

To use the MQ option, you must add jars from a WebSphere MQ client or server installation. MQ Clients can be downloaded at http://www.ibm.com/software/integration/wmq/clients/.

For WebSphere MQ 9, the required jar is:

- `com.ibm.mq.allclient.jar`

For WebSphere MQ 8, the required jar is:

- `com.ibm.mq.allclient.jar`

For WebSphere MQ 7, the required jars are:

- `com.ibm.mq.jar`
- `com.ibm.mq.jmqi.jar`
- `com.ibm.mq.headers.jar`
- `com.ibm.mq.pcf.jar`
- `com.ibm.mq.commonservices.jar`
- `connector.jar`

For WebSphere MQ 6 and earlier, the required jars are:

- `com.ibm.mq.jar`
- `connector.jar`

For all versions, these jar files can be found at `[WebSphere MQ Installation directory]/java/lib`.

To add these jar files to SOAtest's or Virtualize's classpath, complete the following:

1. Choose **Parasoft> Preferences**.
2. Open the **Parasoft> System Properties** page.
3. Click the **Add JARS** button and choose and select the necessary JAR files (outlined above).

Configuring MQ Options

After selecting **Websphere MQ** from the Transport drop-down menu within the **Transport** tab of an appropriate tool, the following options display in the left panel:

MQ Address

When specifying MQ connection settings, you can choose between the following modes:

- **Default mode:** Lets you enter connection details (e.g., host, port, channel, etc.) manually.
- **CCDT mode:** Lets you specify a client channel definition table (CCDT) file that provides connection details.
- **Bindings mode**: Use this mode when the queue manager and connected applications are running on the same system. The IBM WebSphere MQ Java API connects directly to the queue manager using the Java Native Interface (JNI). To use the bindings transport, the IBM MQ classes for JMS must be run in an environment that has access to the IBM MQ Java Native Interface libraries

If you are using Default mode, complete the following fields:

Option	Description
Host	Specifies the name of the host running IBM MQ.
Port	Specifies the port where IBM MQ is running (default is 1414).
Channel	Specifies the name of the server-defined channel.
Queue manager	Specifies the queue manager's name.
Put queue	Specifies the queue that the message is sent to.
Get queue	Specifies the queue that the message is retrieved from.

If you are using CCDT mode, complete the following fields:

Option	Description
CCDT file	Specifies the location of the CCDT file (with a .tab extension).
Queue manager	Specifies the queue manager's name.
Put queue	Specifies the queue that the message is sent to.
Get queue	Specifies the queue that the message is retrieved from.

If you are using Bindings mode, complete the following fields:

Option	Description
Queue manager	Specifies the queue manager's name.
Put queue	Specifies the queue that the message is sent to.
Get queue	Specifies the queue that the message is retrieved from.

Authentication

Select the **Perform Authentication** check box and enter the **Username** and **Password** to authenticate the request. If the correct username and password are not used, the request will not be authenticated.

Message Exchange Pattern

The following options are available for **Message Exchange Pattern:**

- **Expect Synchronous Response:** Specifies whether or not SOAtest or Virtualize receives a response. If **Expect Synchronous Response** is selected, SOAtest or Virtualize sends a message and receives a response. If **Expect Synchronous Response** is not selected, SOAtest or Virtualize sends a one-way message and does not receive a response.

MQRFH2 Header

This section allows you to configure the MQRFH2 header, which is used to pass messages to and from a message broker that belongs to WebSphere Message Broker (described at http://www.ibm.com/support/knowledgecenter/SSKM8N_7.0.0/com.ibm.etools.mft.doc/aq06920_.htm).

In the **MQRFH2 Header** area, you can configure different parts of the request message header, then SOAtest or Virtualize will create header "folders" based on the specified settings.

- **Message Content Descriptor:** This panel lets you configure the <mcd> (Message content descriptor) folder. The <mcd> folder can contain elements that describe the structure of the message data in a WebSphere MQ message. They are all character strings and are case-sensitive.
- **Publish/Subscribe Command:** This panel lets you configure the <psc> (Publish/subscribe command) folder. The <psc> folder is used to convey publish/subscribe command messages to the broker. Only one psc folder is allowed in the NameValueData field. See http://www.ibm.com/support/knowledgecenter/SSKM8N_7.0.0/com.ibm.etools.mft.doc/aq06950_.htm (Command messages) for full details
- **Application (usr) Defined Properties:** This panel lets you configure the <usr> (Application [user] defined properties) folder. The content model of the <usr> folder has the following characteristics.

    - Any valid XML name that does not contain a colon can be used as an element name.
    - Only simple elements (not group) are allowed.
    - All elements take the default type of string.
    - All elements are optional, but should not occur more than once in a folder.
    - An MQRFH2 instance can contain no more than one <usr> folder.
- **Java Messaging Service:** This panel lets you configure the <jms> (Java Messaging Service) folder. The content model of the <jms> folder contains the following MQRFH2 JMS fields:
    - Dst - represents the JMSDestination header field.
    - Div - represents the JMSDeliveryMode header field.
    - Exp - represents the JMSExpiration header field.
    - Pri - represents the JMSPriority header field.
    - Tms - represents the JMSTimestamp header field.
    - Cid - represents the JMSCorrelationID header field.
    - Rto - represents the JMSReplyTo header field.

      See http://www.ibm.com/support/knowledgecenter/SSKM8N_7.0.0/com.ibm.etools.mft.doc/ac24863_.htm for more information on the content of JMS messages.
- **MQRFH2 Get Messages Options:** This panel allows you to control the **Transform MQRFH2 Header to XML format for Get Messages** option. If this option is enabled, the response message's MQRFH2 header will be transformed into readable XML format. This allows you to apply XML output tools (e.g., XML Data Bank, XML Transformer). Without this option enabled, the response message's MQRFH2 header may contain characters which are not viewable.

Put Messages

The following options are available for **Put Messages**:

- **MQPutMessageOptions.options:** Controls the action of `MQQueue.put()`. Any or none of the following values can be specified.
    - **MQPMO_SYNCPOINT:** The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.
    - **MQPMO_NO_SYNCPOINT:** The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work. If neither this option nor MQPMO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment, see MQPMO_SYNCPOINT. Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.
    - **MQPMO_NO_CONTEXT:** Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:
        - Blank, for character fields
        - Null, for byte fields
        - Zero, for numeric fields
    - **MQPMO_SET_IDENTITY_CONTEXT:** The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT.
    - **MQPMO_SET_ALL_CONTEXT:** The message is to have context information associated with it. The application specifies the identity and origin context in the MQMD structure.
    - **MQPMO_FAIL_IF_QUIESCING:** This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.
    - **MQPMO_NEW_MSG_ID:** The queue manager replaces the contents of the MsgId field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.
    - **MQPMO_NEW_CORREL_ID:** The queue manager replaces the contents of the CorrelId field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.
    - **MQPMO_LOGICAL_ORDER:** This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call. See WebSphere MQ Application Programming Reference for more information on this option.
    - **MQPMO_ALTERNATE_USER_AUTHORITY:** This indicates that the AlternateUserId field in the ObjDesc parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if this AlternateUserId is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so.
    - **MQPMO_RESOLVE_LOCAL_Q:** Use this option to fill ResolvedQName in the MQPMO structure with the name of the local queue to which the message is put, and ResolvedQMgrName with the name of the local queue manager that hosts the local queue.
- **Put Queue open options for MQQueue.access():**
    - **MQOO_OUTPUT:** Open the queue to put messages. The queue is opened for use with subsequent MQPUT calls.
    - **MQOO_PASS_IDENTITY_CONTEXT:** This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. The MQOO_OUTPUT option must be specified.
    - **MQOO_PASS_ALL_CONTEXT:** This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.
    - **MQOO_SET_ALL_CONTEXT:** This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity and origin context information contained in the MsgDesc parameter specified on the MQPUT or MQPUT1() call. The MQOO_OUTPUT option must be specified.
    - **MQOO_SET_IDENTITY_CONTEXT:** This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity context information contained in the MsgDesc parameter specified on the MQPUT() or MQPUT1 call. This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.
    - **MQOO_ALTERNATE_USER_AUTHORITY:** The AlternateUserId field in the ObjDesc parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.
    - **MQOO_FAIL_IF_QUIESCING:** The MQOPEN call fails if the queue manager is in quiescing state. This option is valid for all types of object.

- **MQMD.report:** A report message about another message. This field enables SOAtest & Virtualize sending the original message to specify which report or response messages are required, whether the application message data is to be included in them, and also how the message and correlation ID in the report or reply are to be set. It comprises one or more constants from the MQC class.

  You may select one type from each of the following:

  - **Exception:**
    - **MQRO_EXCEPTION:** A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.
    - **MQRO_EXCEPTION_WITH_DATA:** This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.
    - **MQC.MQRO_EXCEPTION_WITH_FULL_DATA:** Exception reports with full data required. This is the same as MQRO_EXCEPTION, except that all the application message data from the original message is included in the report message.
  - **Expiration:**
    - **MQRO_EXPIRATION:** This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed. If this option is not set, no report message is generated if a message is discarded for this reason.
    - **MQRO_EXPIRATION_WITH_DATA:** This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.
    - **MQRO_EXPIRATION_WITH_FULL_DATA:** This is the same as MQRO_EXPIRATION, except that all the application message data from the original message is included in the report message.
  - **Confirm on arrival:**
    - **MQRO_COA:** This type of report is generated by the queue manager that owns the destination queue when the message is placed on the destination queue. Message data from the original message is not included with the report message. If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager can be retrieved only if the unit of work is committed.
    - **MQRO_COA_WITH_DATA:** This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.
    - **MQRO_COA_WITH_FULL_DATA:** This is the same as MQRO_COA, except that all the application message data from the original message is included in the report message.
  - **Confirm on delivery:**
    - **MQRO_COD:** This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message. If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.
    - **MQRO_COD_WITH_DATA:** This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.
    - **MQRO_COD_WITH_FULL_DATA:** This is the same as MQRO_COD, except that all the application message data from the original message is included in the report message.

  You can specify how the message ID is generated for the report or reply message:
  - **MQRO_NEW_MSG_ID:** This is the default action, and indicates that if a report or reply is generated as a result of this message, a new MsgId is generated for the report or reply message.
  - **MQRO_PASS_MSG_ID:** If a report or reply is generated as a result of this message, the MsgId of this message is copied to the MsgId of the report or reply message.

  You can specify one of the following to control how to set the correlation ID of the report or reply message:
  - **MQRO_COPY_MSG_ID_TO_CORREL_ID:** This is the default action, and indicates that if a report or reply is generated as a result of this message, the MsgId of this message is copied to the CorrelId of the report or reply message.
  - **MQRO_PASS_CORREL_ID:** If a report or reply is generated as a result of this message, the CorrelId of this message is copied to the CorrelId of the report or reply message.

  You can specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:
  - **MQRO_DEAD_LETTER_Q:** This is the default action, and places the message on the dead-letter queue if the message cannot be delivered to the destination queue. An exception report message is generated, if one was requested by the sender.
  - **MQRO_DISCARD_MSG:** This discards the message if it cannot be delivered to the destination queue. An exception report message is generated, if one was requested by the sender.
  - **MQRO_PASS_DISCARD_AND_EXPIRY:** If this option is set on a message, and a report or reply is generated because of it, the message descriptor of the report inherits:
    - MQRO_DISCARD_MSG if it was set.
    - The remaining expiry time of the message (if this is not an expiry report). If this is an expiry report the expiry time is set to 60 seconds.
- **Expiry:** Enter the expiry time (in tenths of a second). It is set by the application which puts the message. After a message's expiry time has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the MQC.MQRO_EXPIRATION flags, then a report is generated when the message is discarded.
- **Correlation ID:** Enter the correlation ID to use for the correlationID field in the message
- **Message sequence number:** Enter the sequence number of logical message within group.
- **Reply queue manager name:** Enter the name of the queue manager to which reply or report (response) messages can be sent.
- **Reply queue name:** Enter the name of the queue to which a reply can be sent.
- **Put application name:** Enter the name of the application that put the message.
- **Originating application data:** Enter data about the originating application. This can be used to provide additional information about the origin of the message.
- **User ID:** Enter the user ID. It is part of the identity of the message and identifies which user originated it.

Get Messages

The following options are available for **Get Messages**:

- **MQGetMessageOptions.options:** Options which control the action of MQQueue.get() that is internally invoked by SOAtest & Virtualize.

  Any or none of the following values can be specified.

  - **MQGMO_WAIT:** SOAtest or Virtualize waits until a suitable message arrives. The maximum waiting time is specified in WaitInterval.
  - **MQGMO_NO_WAIT:** SOAtest or Virtualize does not wait if no suitable message is available. This is the opposite of the MQGMO_WAIT option, and is defined to aid program documentation.
  - **MQGMO_SYNCPOINT:** The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.
  - **MQGMO_NO_SYNCPOINT:** The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.
  - **MQGMO_BROWSE_FIRST:** When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.
  - **MQGMO_BROWSE_NEXT:** Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to SOAtest or Virtualize, but remains on the queue. After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option.
  - **MQGMO_BROWSE_MSG_UNDER_CURSOR:** Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO. The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls have been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively. The position of the browse cursor is not changed by this call.
  - **MQGMO_MSG_UNDER_CURSOR:** Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO. The message is removed from the queue. The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.
  - **MQGMO_LOCK:** Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue.
  - **MQGMO_UNLOCK:** Unlock a message. The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQRC_NO_MSG_LOCKED.
  - **MQGMO_ACCEPT_TRUNCATED_MSG:** If the message buffer is too small to hold the complete message, allow the MQGET call to fill the buffer with as much of the message as the buffer can hold.
  - **MQGMO_FAIL_IF_QUIESCING:** Force the MQGET call to fail if the queue manager is in the quiescing state. On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.
  - **MQGMO_CONVERT:** Requests SOAtest or Virtualize data to be converted. The conversion conforms to the characterSet and encoding attributes of MQMessage, before the data is copied into the message buffer.
- **MQGetMessageOptions.matchOptions:** Selection criteria which determine which message is retrieved. The following match options can be set:
  - **MQMO_MATCH_CORREL_ID:** The message to be retrieved must have a correlation identifier that matches the value of the CorrelId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).
  - **MQMO_MATCH_GROUP_ID:** The message to be retrieved must have a group identifier that matches the value of the GroupId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).
  - **MQMO_MATCH_MSG_ID:** The message to be retrieved must have a message identifier that matches the value of the MsgId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).
  - **MQMO_MATCH_MSG_SEQ_NUMBER:** The message to be retrieved must have a message sequence number that matches the value of the MsgSeqNumber field in the MsgDesc parameter of MQGMO the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).
  - **MQMO_NONE:** Do not use matches in selecting the message to be returned. All messages on the queue are eligible for retrieval. MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.
- **Get Queue open options for MQQueue.access():**
  - **MQOO_BROWSE:** Open the queue to browse messages. The queue is opened for use with subsequent MQGET calls with one of the following options: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT and MQGMO_BROWSE_MSG_UNDER_CURSOR. This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue.
  - **MQOO_INPUT_AS_Q_DEF:** Open the queue to get messages using the queue defined default. The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the DefInputOpenOption queue attribute.
  - **MQOO_INPUT_SHARED:** Open the queue to get messages with shared access. The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.
  - **MQOO_INPUT_EXCLUSIVE:** Open the queue to get messages with exclusive access. The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type.
  - **MQOO_ALTERNATE_USER_AUTHORITY:** The AlternateUserId field in the ObjDesc parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which SOAtest or Virtualize is running is authorized to do so.
  - **MQOO_FAIL_IF_QUIESCING:** The MQOPEN call fails if the queue manager is in quiescing state. This option is valid for all types of object.
- **Wait Interval:** Enter the maximum time (in milliseconds) that an MQQueue.get() call waits for a suitable message to arrive. It is used in conjunction with MQC.MQGMO_WAIT. This has no effect if MQMO_WAIT is not selected. Value -1 is equivalent to having MQGMO_NO_WAIT selected.

Queue Manager Options

The following options are available for **Queue Manager Options**:

- **MQQueueManager binding options:** Creates a connection to the named queue manager specifying binding options.
  - **MQCNO_FASTPATH_BINDING:** This option causes SOAtest or Virtualize and the local-queue-manager agent to be part of the same unit of execution. This is in contrast to the normal method of binding, where SOAtest or Virtualize and the local-queue-manager agent run in separate units of execution.
  - **MQCNO_STANDARD_BINDING:** This connection option causes SOAtest or Virtualize and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs.
  - **MQCNO_SHARED_BINDING:** This connection option causes SOAtest or Virtualize and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager. That is, it protects the queue manager from errant programs. However, some resources are shared between SOAtest or Virtualize and the local-queue-manager agent.
  - **MQCNO_ISOLATED_BINDING:** This option causes SOAtest or Virtualize and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs. SOAtest or Virtualize process and the local-queue-manager agent are isolated from each other in that they do not share resources.

MQ Queue Manager Properties

The following option is available for **MQ Queue Manager Properties**:

- **MQC.CCSID_PROPERTY:** This configures the CCSID used by the client. It does not apply when connecting directly to WebSphere MQ in bindings mode.

  Changing this value affects the way that the queue manager you connect to translates information in the WebSphere MQ headers. All data in WebSphere MQ headers is drawn from the invariant part of the ASCII codeset, except for the data in the MQMessage.applicationIdData and MQMessage.putApplicationName fields.

  If you avoid using characters from the variant part of the ASCII codeset for these two fields, then the CCSID can be changed from 819 to any other ASCII codeset. If you change the client CCSID to be the same as that of the queue manager to which you are connecting, you gain a performance benefit at the queue manager because it does not attempt to translate the message headers. If no value is provided in the CCSID field, SOAtest or Virtualize will provide a default value of 1208 (UTF-8).

SSL

You can configure the following SSL settings.

CipherSuite

The CipherSuite is used for the SSL connection on the specified MQ channel. Refer to the IBM MQ documentation to determine which CipherSuite to select based on the CipherSpec.

You can specify a fixed CipherSuite from the drop-down menu or enter a parameterized variable to specify a CipherSuite based on traffic.

Peer Name

(Optional) Specify a peer name to verify that the certificate presented by the Queue Manager matches the criteria specified with the peer name parameter. A server certificate will match this parameter with the Distinguished Name (DN) of the certificate presented by the Queue Manager.

You can specify a fixed peer name or enter a parameterized variable to specify a peer name based on traffic.

Key Store

This setting specifies the Key Store to be used for Client-Side SSL (authentication of the client by the Queue Manager). You can configure tools to use either the key store configuration for all projects under **Parasoft> Preferences** (see Security Settings) or you can add a suite-level global settings property and configure key store settings available to all tools in the suite. SeeAdding Global Propertiesfor additional details. If the MQ Channel does not require the client to authenticate itself, then the **Key Store** does not need to be provided.

Trust Store

Specifies the Trust Store to be used for Server-Side SSL (authentication of the Queue Manager by the client). You can either configure tools to use the trust store configuration for all projects under **Parasoft> Preferences** (see Security Settings) or you can add a suite-level global settings property and configure trust store settings available to all tools in the suite. SeeAdding Global Propertiesfor additional details.

Once an SSL connection is attempted by specifying a CipherSuite and running the tool, the trust store, key store, and key store passwords cannot be changed. If they are changed, SOAtest or Virtualize needs to be restarted before changes will take effect.

If using Key Stores, you will need to download and install the Unlimited Strength Java Cryptography Extension. For details, see JCE Prerequisite.

Scripting Hook

The **Scripting Hook** options allow you to customize MQ Properties by using scripting language such as Jython, Java, Groovy, and JavaScript.

If you need more information on using the scripting utility, please refer to the Scripting section of the tutorial. For a list of scripting APIs, choose **Parasoft> Help**, then look for the book titled "Parasoft SOAtest Extensibility API" (for SOAtest) or "Parasoft Virtualize Extensibility API" (for Virtualize).

The following are scripting access keys:

- QueueManager – mqManager
- GetQueue – mqGetQue
- PutQueue – mqPutQue
- PutMessage – mqPutMessage
- GetMessage – mqGetMessage
- PutMessageOptions – mqPutMessageOptions
- GetMessageOptions – mqGetMessageOptions

For example, if you like to change the Expiry time for put message to 999:

```
from com.ibm.mq import *
def changeExpiry(context):
 putMessage = context.get("mqPutMessage")
 putMessage.expiry = 999
```

Once you are done running the tool with the above script, note the Expiry field in the Traffic header has changed to 999.

Note: Any options set in the **Add MQ Hook** tab overrides and takes precedence over any options set in the other tabs.

Interpreting WebSphere MQ Error Messages

When a failure occurs, MQ returns a reason code for a failure. SOAtest & Virtualize error messages report these same reason codes in order for users to interpret them. For a list of MQ reason codes and their meaning, please refer to the IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.tro.doc/q040710_.htm.

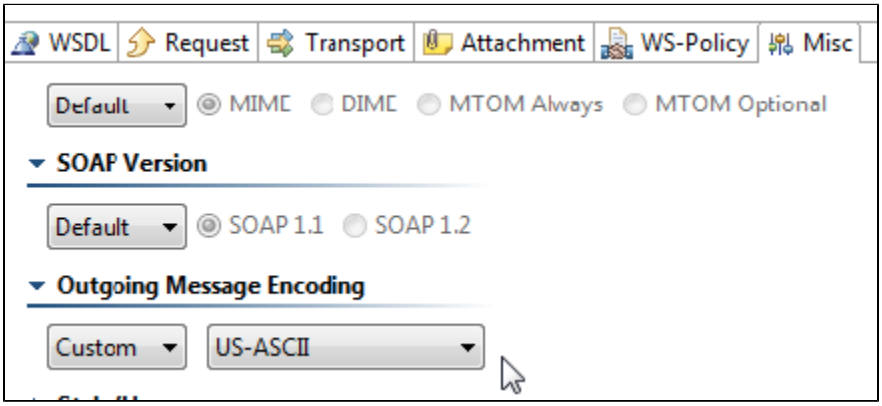
Sending and Receiving Data - Best Practices

The following are best practices for sending and receiving character data with client tools (e.g., SOAP Client, REST Client, Messaging Client, etc.)

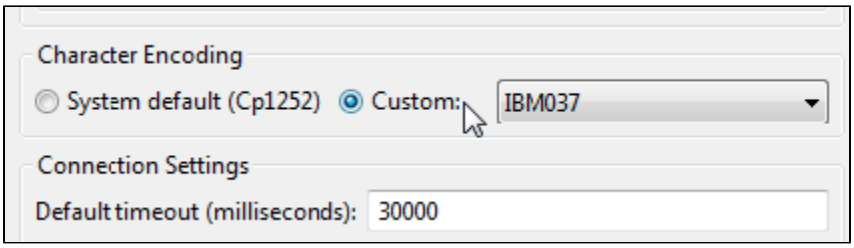
Put Messages

When sending character data such as XML, CSV, fixed-length, or plain text, the format type must be set to the value of the MQFMT_STRING constant which is MQSTR.

For the SOAP client tool, the character set used to encode the request is specified using the tool's Misc tab> Outgoing Message Encoding option.



Other applicable tools (e.g., REST Client, Messaging Client) use the character encoding configured in the product's Misc preferences.



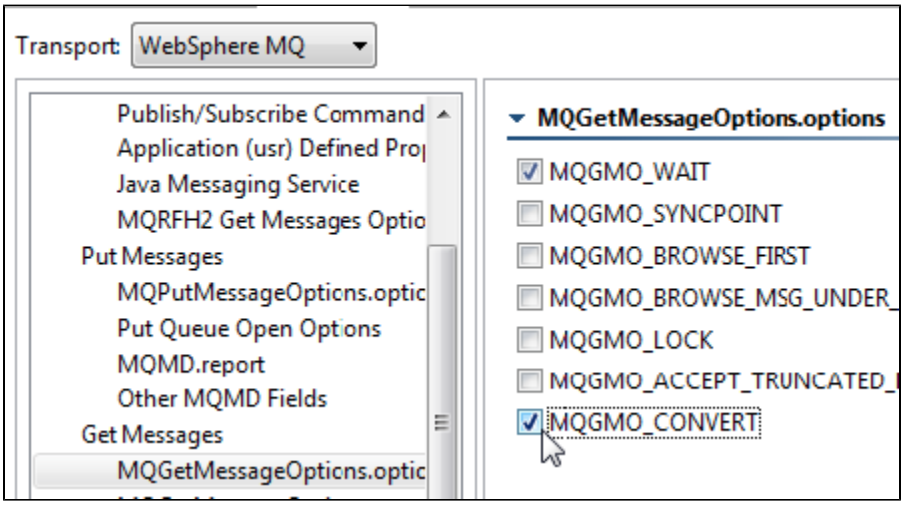
The applicable options are:

- IBM_037
- IBM_437
- ISO_8859_1
- US_ASCII
- UTF16
- UTF8.

If you specify a different encoding, then the character set of the MQ messages will be defaulted to MQCCSI_Q_MGR, which means "Character data in the message is in the queue manager's character set."

Get Messages

The MQGMO_CONVERT box should be enabled under the client tool's MQGetMessageOptions (in the **Transport** tab).



This will instruct the queue manager to convert the message to the client tool's character set. This is important if the message's original characterSet is not one of those supported by the client (IBM_037, IBM_437, etc.).

The character set used to perform the conversion is configured the same as for put messages (described above).