Best practices for integrating IBM WebSphere DataPower SOA Appliances with WebSphere MQ

Chintamani (Chin) Sahoo (chintam3@us.ibm.com)
Advisory Software Engineer
IBM

15 October 2014

IBM® WebSphere® DataPower SOA Appliances are purpose built computing devices that provide many integration end points for backend services. WebSphere MQ is one of the services used with DataPower to process messages in the enterprise. The integration of DataPower application with MQ requires an understanding of many configuration objects and its settings. This tutorial presents best practices for configuring these objects to integrate with MQ, improve performance in message processing, and protect it from overutilization of resources such as CPU, memory, and connections.

Introduction

IBM WebSphere DataPower SOA Appliances (hereafter called DataPower) are purpose built computing devices that provide integration end points for many services, such as WebSphere MQ, WebSphere Service Integration Bus using Java™ Messaging Service (JMS), WebSphere Service Registry and Repository (WSRR), TIBCO Enterprise Message Service (EMS), Lightweight Directory Access Protocol (LDAP), and database servers. These integration capabilities are designed to provide rapid and secure deployment of business solutions for the enterprise. Further, DataPower firmware and hardware components are optimized for faster execution of business policies and transactions using these integration service end points.

DataPower provides a programming model for developing configuration artifacts. These artifacts include controls such as service objects, processing policies, rules, actions, client side protocol handlers, and URL openers that create business applications easily using the web graphical user interface (webGUI). For example, you can configure the Multi-Protocol Gateway (MPGW) service to implement security policies by accessing a centralized LDAP service, or messages that can be processed by using MPGW services that use MQ Queue Manager (mq-qm) object and MQ front side handlers (FSH) for the backend MQ server. These DataPower objects need configurations that can provide optimal performance as well as protection from over using resources such as CPU, memory, and connections in the appliance. This tutorial discusses best practices for integrating DataPower objects with the MQ server to effectively handle various traffic patterns.
The MQ integration end points in DataPower are provided via the MQ Queue Manager Object "mq-qm" and MQ Front side handler objects. Therefore, it is necessary to present these objects and provide configuration settings for various attributes of these objects that are consistent with best practices for product integration with MQ.

**Configuring the DataPower mq-qm object**

The *mq-qm object* is the primary component that provides connectivity capabilities for MQ. It manages the connection pool size and idle timeouts. It retries the behavior when the connection is not established to the backend MQ server and sync point flow. Figure 1 presents various attributes of the mq-qm object. The required attributes are marked with an asterisk that can provide connectivity to the MQ server. However, the critical attributes are identified in red, which should *not* use the default settings. If these attributes are configured with values as suggested by the best practices, it will provide protection as well as performance improvements for the appliance. You can use the mq-qm object with the Multi-Protocol Gateway or Webservice-Proxy service to process the MQ traffic.

**Figure 1. Configuration panel of mq-qm object showing critical attributes highlighted in red for the Main tab**
The main tab provides several required fields, which must be provided to connect the mq-qm object with the backend MQ server. However, there are few attributes that need special consideration to configure this mq-qm object:

1. **Maximum Message Size**: This defines the maximum size of messages in bytes that the mq-qm object will process. This value should be equal to or less than the MaxMsgLength attribute of the channel and the queue of the MQ server. Messages that are bigger than this size are rejected. DataPower will show MQ return code of 2010 (MQRC_DATA_LENGTH_ERROR) when the message size in the queue exceeds the value of this attribute. When the request type of service is XML or SOAP, the associated XML Manager's "XML Bytes Scanned" property should be adjusted from the default value of 4194304 bytes to the value of this attribute to successfully process the message.

2. **Cache Timeout**: This specifies the number of seconds that the DataPower appliance retains (keeps alive) a connection in the connection cache. The default value is an empty string. If the default value is used in the mq-qm object, the connections are not closed. Therefore, it is important to use a value that is greater than the negotiated heartbeat interval, but less than the "KeepAlive" interval of the MQ server. If the "KeepAlive" timeout is not known, you can use a value of 60 seconds for this attribute. Note that this attribute is only related to idle connections. If the connection is being used, this value does not affect the active connections. If one finds a "connection" error for the first MQ transaction and the subsequent transactions are processed, it is mostly related to the default cache timeout value. When an inactive connection reaches a Cache Timeout threshold, the DataPower appliance removes that connection from the cache. If the cache no longer contains connections, the DataPower appliance creates new connections with the backend MQ server.

   **Note**: The Cache Timeout attribute is the only way to configure a timeout value for the DataPower appliance. No other configuration setting on the DataPower appliance can time out an MQ connection.

3. **Units of Work**: This configures the use of sync point flow when set to "1". The default value is set at zero and causes the appliance to get and put messages with no provision for roll back. Either the MQ operation succeeds or not. Undeliverable messages are silently discarded, which leaves higher level protocols with the responsibility to detect and retransmit lost packets. When set to 1, DataPower appliance uses sync point flow that allows MQ commit and roll back for each MQ message, not the entire transaction. If the sync point flow is configured, DataPower does not remove the message that it gets from a queue until it completes the put for that message. When the transaction fails and the message is left on the queue, DataPower retries the MQPUT for that message until the Backout Threshold is reached. If the Automatic Backout is set to "on", the failed message is routed to the configured Backout Queue.

   **Note**: If units-of-work is set to 1, the Automatic Backout should be configured as "on" with its associated Backout Threshold and Backout Queue attributes specified in the mq-qm object.

4. **Automatic Backout**: This is required to set as "on" when units-of-work is set at 1. The radio button enables (on) or disables (off) the automatic backout of poison messages. A poison message is any message that the receiving application does not know how to process. Usually when an application rolls back the message leaving it on the input queue, the backout count (MQMD.BackoutCount) is incremented. As the MQ FSH object continues to re-get the
message, the backout count is incremented. When the backout count exceeds the backout threshold, the mq-qm object moves the message to the backout queue. If disabled, the poison message remains on the GET queue and continue to be reprocessed by the MQ FSH until the Queue Manager that manages the GET queue removes it or the appliance reroutes the offending message. This creates a looping condition in the appliance and can cause the device to spike in memory and CPU usages. The best practice is to configure this attribute as "on" so that application can provide an exist path for the poison message.

5. **Backout Threshold**: This specifies maximum number of retry attempts per message. During processing, the count starts at 0, which accounts for the initial reception of a message. If the initial attempt fails, the backout count is set to 1. After one failed attempt to reprocess the message, the count is set to 2. Therefore, the value of 1 for this attribute specifies two attempts: the initial attempt and a single reprocessing attempt. The best practice is to use value of 1. For higher toleration of error conditions, this value can be more than 1. This causes performance degradation as reprocess attempts of the message increases.

**Note**: If the MQ FSH is configured with the concurrent MQ connections, which is greater than 1, this value should be set at a number equal to the concurrent MQ connections + 1 so as to force the roll back message to the input queue. Otherwise, the roll back message will be routed to Backout queue. If multiple devices are consuming messages from a common input queue in the MQ server and one of the devices reloads, the message that was rolled back due to this failure can be reprocessed by the MQ FSH of another active device.

6. **Backout Queue Name**: This defines the queue to contain messages that are processed due to exceeding the limit established in the backout threshold. This queue must be managed by the same MQ server as the defined input queue. The Backout queue, typically `SYSTEM.DEAD.LETTER.QUEUE`, contains messages that cannot be processed or delivered. However, the best practice is to use a local queue for this attribute.
Figure 2. Configuration panel of mq-qm object showing critical attributes highlighted in red for the Connection tab

7. **Total-connection-limit**: This defines the connection pool size of the mq-qm object. The default value is 250. For high volume MQ traffic, this default value may not be optimal to process transactions within the expected response time. If MQ transactions show a higher latency, you can tune the total-connection-limit to use larger number of connections so that the desired performance goal is achieved. This tuning should be empirically verified so that appliance can sustain the load in steady state and the backend MQ server can process the load without errors. If the load is not high, the number of connections should be reduced to smaller value based on the load factor.

   **Note**: This attribute is the only mechanism that can increase or decrease the connection pool size. While increasing this value, care should be taken to verify that the backend MQ server is provisioned to accommodate all client connections. The "MaxChannel" value should be checked in the respective MQ server to make sure the value is greater than the number of connections from all DataPower and non-DataPower clients for this MQ server.

8. **Local Address**: This defines the local address for outbound connections to a specific local interface and port. Supported formats are x.x.x.x or x.x.x.x(1414) or just (1414) to tell TCP to bind to Port 1414. For a range of ports, use (1414,1420) or x.x.x.x(1414,1420). If the port is set, the range must be greater than the total number of allowed connections. Since DataPower has multiple network interfaces (NIC) and can route traffic randomly when configured with a default route for these interfaces, the best practice is to configure one of the Network Interface IP addresses in this attribute. This configuration binds the mq-qm object
with a dedicated IP so that the routing of MQ messages can happen in a consistent and predictable manner.

**Note:** This approach is useful when more than one of the Network Interfaces is configured with default routes and the MQ Queue Manager Group (mq-qm-group) is used as the fail-over mechanism for MQ traffic in the DataPower appliance.

9. **Retry Interval:** This specifies the time interval (in seconds) between attempts to retry the failed connections to a remote host. This setting does not affect attempts to PUT or GET messages over an established connection. The default value is 1 second. If the default value is used, the appliance will aggressively retry for every one second and can spike CPU and memory usage. The **best practice** is to use a value of 10 or 15 seconds as the retry interval for failed connection.

10. **Retry Attempts:** This defines the number of attempts to retry the failed connections. After the number of attempts is reached, the Long Retry Interval is used instead. It should be noted that the default value of 0 (zero) means the Long Retry Interval will not be used and will retry forever using Retry Interval. The **best practice** is to configure this attribute with a value greater than zero. For an example, if the value of 6 is configured, this allows 6 attempts to retry in an interval of 10 seconds. After completing 6 attempts, the mq-qm object falls back to Long Retry Interval and continues retries if the connection to the backend is not established. This mechanism helps in reducing the impacts from retries by using a staggering approach.

11. **Long Retry Interval:** This configures the long retry interval to use after the number of retry attempts is reached. The default value is 1800 seconds and it should be greater than the Retry Interval. Otherwise, it will take no effect. While configuring this attribute, you should consider number of seconds to wait for the next retry to reconnect with the MQ server. The **best practice** is to use 600 seconds for this attribute. However, consider other use case scenarios and configure an appropriate value that is smaller than the 600 seconds.

12. **Reporting Interval:** This defines the number of seconds to report log messages at the error level when failed connections are retried. This setting filters the generation of identical error messages to MQ logging target. The **best practice** is to configure a value equal to the Retry Interval for this attribute for most use cases. However, consider a higher value for this attribute when you do not want to display log messages in the system log.

13. **Sharing Conversations** This configures the maximum number of conversations to share a single TCP/IP connection. To enable conversation sharing, use the following guidelines: WebSphere MQ V7.0 and later releases provide sharing conversations (SHARECNV) attribute on SVRCONN channel that specifies the maximum number of conversations that can share each TCP/IP channel instance. This feature can be configured in DataPower as it uses client connection with queue manager SVRCONN channel. The default setting of sharing conversations (SHARECNV) value is zero in the DataPower mq-qm object. The same default value for MQ SVRCONN channel is 10. The sharing conversations value is negotiated between MQ server and DataPower and the lower value takes effect. However, in current versions of the DataPower firmware, the sharing conversations setting of 1 is treated as 0 when negotiating with MQ server.

There are three use cases to configure sharing conversations in a DataPower mq-qm object:

- **Case 1:** The negotiated sharing conversations value is 0. The channel runs in a mode similar to WebSphere MQ V6.0 and does not use features such as:
• Administrator stop-quiesce
• Heartbeating
• Read ahead
• Client asynchronous consume

Set a value of 0 or 1 on the Sharing Conversations attribute of the mq-qm object in DataPower to disable sharing conversations, regardless of the MQ SVRCONN setting.

• **Case 2**: The negotiated sharing conversations value is 1. The channel supports MQ V7.0 and later features as outlined in Case 1, but each TCP/IP channel instance has a single conversation.
  
  Set a value of 2 on the Sharing Conversations attribute of the mq-qm object in DataPower and a value of 1 on the MQ SVRCONN setting.

• **Case 3**: The negotiated sharing conversations value is 2 or more. The channel supports MQ V7.0 and later features and each TCP/IP channel instance supports two or more conversations.
  
  Set a value of 2 or more on the Sharing Conversations attribute of the mq-qm object in DataPower and on the MQ SVRCONN channel.

  On average, the processing of messages from client applications is 15 percent slower when using SHARECNV(10) as compared to SHARECNV(0). Refer to [Performance Implications of Sharing Conversations on Client-connection Channels](#).

### 14. Dedicated XML Manager:

When multiple mq-qm objects use the same XML manager, such as the "default" in a single application domain, the configuration change in one mq-qm object can affect the states of other mq-qm objects as they share the same XML manager. The best practice is to use a dedicated XML Manager along with its associated User Agent object for each mq-qm object so that appliance can effectively manage the state changes for lower level objects.

**Configuring the DataPower MQ Front Side Handler object**

The MQ Front Side Handler (FSH) object is configured to process the client side GET and PUT operations from the MQ server queues. Figures 3 and 4 present the critical attributes highlighted in red that require user consideration while configuring this object.
Figure 3. Configuration panel of MQ FSH object showing critical general attributes highlighted in red

Figure 4. Configuration panel of MQ FSH object showing critical attributes for Properties and Headers highlighted in red
The MQ FSH polls the input queue based on the polling interval. It allows to Get and Put messages to the designated queues. This object is also known as the source MQ handler and needs consideration for configuring various parameters.

1. **The number of concurrent MQ connections**: This specifies the number of concurrent MQ connections allocated for this MQ FSH. The name "concurrent MQ connections" is a misnomer with respect to the behavior of the MQ FSH. It represents the number of active and pending MQGETs for the front side input queue (GETQ). The default setting of 1 connection can use more connections for active and pending MQGET operations if there are many messages available in the input queue. Hence, in most use cases, this default value works well. If there is need for processing more loads, the value for this attribute can be increased to a higher number. However, this number should be empirically tested to make sure that the backend MQ server can handle the load and the appliance does not keep many transactions in its memory due to latency in the backend MQ server. The behavior of MQ FSH processing messages are described below.

The MQ FSH polls for messages with the MQ connection (hereafter referred to as C1). After the message is returned from the GETQ, DataPower starts processing the message, which may involve MQPUT. This takes additional connection if MQ FSH and the backend MQ URL use different mq-qm objects. However, the connection (C1) will be shared with the backend MQ URL if both front side and backend use the same mq-qm object with units-of-work configured.

In the meantime, the MQ FSH will continue MQGET(s), which may remain in pending states. This increases the number of concurrent connections for the MQ FSH greater than 1. After processing all the messages, this value will become 1 on the expiration of the cache timeout that is specified in the mq-qm object. Note that even setting the concurrent MQ connection as 1, MQ FSH still can use at least two MQ connections (C1 is used for transaction processing, C2 is used to poll for an input queue).

Under an extreme condition, if transaction processing takes longer than the front side time-out and there are many messages in the GETQ, then the MQ FSH can use as many connections as it is allowed by the connection pool (configured in the mq-qm object's total connection limit) to process these messages. In other words, if the request messages arrive at the GETQ with a slow rate, then MQ FSH will keep a lower number of used connections. If the request messages arrive at a higher rate, MQ FSH will use a large number of connections.

In most use cases, a default value of 1 is sufficient. Setting it to a value greater than 1 may yield some performance gain, but the difference is marginal. If message ordering is required, then it must use a value of 1 as multiple MQGETs will break the message ordering. The best practice is to use the default value of 1 for most uses cases. For high volume traffic scenarios, you can use a value of 2-5 connections for this attribute in the MQ FSH configuration.

2. **Polling Interval**: This specifies the number of seconds the MQ FSH will wait for the subsequent MQGET operation when no message is available in the input queue. Setting this value to a minimum helps to quickly detect network connectivity issues and queue manager failures. However, it increases network traffic. In most use cases, the default value of 30 seconds works well.
3. **Retrieve Backout Settings**: This control whether to retrieve a backout setting from the MQ server. When enabled, MQ FSH retrieves the Backout Threshold and Backout Queue Name settings from the MQ server and compares these values to the backout values of the mq-qm object. On retry, the handler uses the higher priority backout settings. If the MQ server does not contain backout settings, the handler uses existing backout values, either empty or populated, from the mq-qm object. If the mq-qm object does not define backout values, this capability is disabled.

   When disabled, the default does not retrieve backout settings from the MQ server. If these properties already exist in the mq-qm object, the handler uses those values. The *best practice* is to use the default value of "off". If there is a need to use this feature, care should be taken to make sure that queues in the MQ server are configured with the backout queue name and backout count threshold.

   Note: When the setting of Retrieve Backout Settings is "on", it can cause performance degradation due to additional MQ API calls for each MQGET operation. For high volume MQ traffic, the *best practice* is to use the default value of "off". Configure Automatic Backout as "on" and provide its associated Backout Threshold value and Backout Queue Name in the mq-qm object when units-of-work is enabled for handling message ROLLBACK.

4. **Parse Properties**: When enabled, it parses the properties of the incoming messages from a queue or a subscription. By default, the attribute is set as "off". When the MQ message does not contain properties, it is the *best practice* to disable this attribute.

5. **Async Put**: This allows the message to PUT asynchronously to a queue without waiting for a response from the queue manager. Do not use this option when units-of-work is configured in the associated mq-qm object because a sync point flow requires a response from the queue manager to process the roll back and commit operations for the message. The *best practice* is to use this option when the backend MQ server environment is stable to improve performance.

6. **Exclude Message Headers**: When selected, this allows the MQ FSH to strip those headers from the MQ message. If the message contains only the MQMD header, there is no need to check these boxes. If the message contains other type headers, such as MQRFH2 or MQIIH headers, the *best practice* is to select the appropriate check boxes so that these headers are excluded in the payload. However, such header values are still available in DataPower for the manipulation of these headers in processing rules.

   - CICS Bridge Header (MQCIH)
   - Dead Letter Header (MQDLH)
   - IMS Information Header (MQIIH)
   - Rules and Formatting Header (MQRFH)
   - Rules and Formatting Header (MQRFH2)
   - Work Information Header (MQWIH)

### Configuring the DataPower MQ URL opener

The MQ URL opener is used to route the message to its destination. The *best practice* is to use the "Build a MQ URL" helper available in the Multi-Protocol Gateway (MPGW) service. Figure 5 shows the highlighted fields that contain values to construct a valid MQ URL for the backend destination.
Figure 5. MQ Helper to build the MQ URL for the backend

Example of a backend static MQ URL for Datagram traffic:

- dpmq://LNX-DP1/?RequestQueue=QUEUE1 (Without units-of-work)
- dpmq://LNX-DP1/?RequestQueue=QUEUE1;Transactional=true (With units-of-work)

Example of a backend static MQ URL for Request/Reply traffic:

- dpmq://LNX-DP1/?RequestQueue=QUEUE1;ReplyQueue=QUEUE2;SetReplyTo=true (Without units-of-work)
- dpmq://LNX-DP1/?RequestQueue=QUEUE1;ReplyQueue=QUEUE2;Transactional=true;Sync=true;SetReplyTo=true (With units-of-work)

Syntax for a static URL to send a message:

dpmq://mqQueueManagerObject/URI?RequestQueue=requestQueueName;queryParameters

Syntax for a static URL to retrieve a message:

dpmq://mqQueueManagerObject/URI?ReplyQueue=replyQueueName;queryParameters

Syntax for a dynamic URL to send or retrieve a message:
To strip the headers from the back side, use the MQ URL parameter of "ParseHeaders" or its alias "ExcludeHeaders". It excludes all the headers from the payload.

**Note:** The use of the "ParseHeaders" tag in the MQ URL is useful when the reply message contains MQRFH2 headers and these extra headers need to be removed from the payload to make it a well-formed XML payload.

To overwrite the service backside timeout, use the `TimeOut=<value in milliseconds>` tag in the MQ URL. The best practice is to configure the "TimeOut" when the MQ URL contains the "replyQueue" tag. By using this "TimeOut" parameter in the MQ URL, the backend connection will wait for the response message until this timeout expires. If the reply is not received by the specified timeout, the backend MQ URL returns a code of 2033, indicating that there is no reply message available within the timeout period.

**Note:** The MQ URL "TimeOut" takes precedence over the Gateway backside timeout setting. The best practice is to configure this value lower than the configured Gateway service backside timeout value.

For more details, see the syntax of the MQ URL in this Information Center topic.

**Manipulating the MQ header in DataPower**

DataPower processes MQ headers such as MQ Message Descriptor (MQMD) and MQ Rules Formatting Headers 1 and 2 (MQRFH & MQRFH2) in the processing policy. When a request message is processed in the request rule, MQMD is not altered by the Gateway service. However, MQMD can be manipulated with a specific transformation (xform or xformbin) action in the processing rule. There are two methods to manipulate MQ headers: using the header injection or using the custom stylesheet.

**Using the header injection method**

When the MQ headers are static for request and reply messages, this method is suitable as each message type can add the necessary headers whose values do not change across multiple request or reply messages. In other words, the same MQ header values are populated for each request or reply message.

Using the Gateway service's header tab, the following examples are used to populate MQ headers:

**Adding a new header injection parameter**

- **Direction:** Back (or Backend MQPUT)
- **Header Name:** MQMD
- **Header Value:** `<MQMD><Format>MQSTR</Format><Persistence>1</Persistence></MQMD>`
Using the stylesheet method

Listing 1 shows the complete MQMD header with values.

Listing 1. Example of MQMD header with values

```xml
<MQMD>
  <StrucId>MD</StrucId>
  <Version>1</Version>
  <Report>0</Report>
  <MsgType>8</MsgType>
  <Expiry>-1</Expiry>
  <Feedback>0</Feedback>
  <Encoding>546</Encoding>
  <CodedCharSetId>819</CodedCharSetId>
  <Format>MQSTR</Format>
  <Priority>0</Priority>
  <Persistence>0</Persistence>
  <MsgId>414d5120454d42444954312020202020d724c841999900802</MsgId>
  <CorrelId>0000000000000000000000000000000000000000000000000</CorrelId>
  <BackoutCount>0</BackoutCount>
  <ReplyToQ>ADAPTERTEST.POSTFORMAT02</ReplyToQ>
  <ReplyToQMgr>EMBDIT2</ReplyToQMgr>
  <UserIdentifier>MUSR_MQADMIN</UserIdentifier>
  <AccountingToken>16010515000000271d19366f0b7216262a1345eb03000000000000000000b
  </AccountingToken>
  <AppIdentityData>
    <!--AppIdentityData-->
  </AppIdentityData>
  <PutApp1Type>28</PutApp1Type>
  <PutAppName>Websphere MQ Client for Java</PutAppName>
  <PutDate>20061005</PutDate>
  <PutTime>18401655</PutTime>
  <App1OriginData>
    <!--App1OriginData-->
  </App1OriginData>
  <GroupId>000000000000000000000000000000000000000000000000</GroupId>
  <MsgSeqNumber>1</MsgSeqNumber>
  <OriginalLength>-1</OriginalLength>
</MQMD>
```

Listing 2 shows the stylesheet that manipulates the MQMD header.

Listing 2. Custom stylesheet method to inject the MQMD header

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  xmlns:dp=http://www.datapower.com/extensions
  extension-element-prefixes="dp"
  exclude-result-prefixes="dp">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <xsl:variable name="newMQMDStr">
      <MQMD>
        <Format>MQSTR</Format><Persistence>1</Persistence>
      </MQMD>
    </xsl:variable>
    <xsl:variable name="mqmdStr">
      <dp:serialize select="$newMQMDStr" omit-xml-decl="yes"/>
    </xsl:variable>
    <xsl:message dp:priority="debug">
      xsl:value-of select="concat('The New MQMD : ', $mqmdStr)"
    </xsl:message>
  </xsl:template>
</xsl:stylesheet>
```
When the header injection or stylesheet method does not inject a full MQMD header, DataPower adds the missing MQMD fields with default values to construct a complete MQMD. In order to populate new values for full MQMD header fields, create the structure with the respective headers and its associated values and inject it using the stylesheet method.

There are few exceptions where the above MQ header injection method does not work. In situations where the MQ URL is being used in the destination box of the Result Action, the MQMD should be injected using the "context" variable method. For example, the request rule has a need to log the MQ message with the MQMD.Expiry header being set. The message is routed using the Result Action with the MQ URL configured in the destination box. In this situation, use the "context" variable method to inject relevant MQMD headers as shown in Listing 3.

**Listing 3. Context variable method to inject the MQMD header**

```xml
<xsl:variable name="MQMDStr">
  <MQMD>
    <Expiry>8000</Expiry>
    <Priority>0</Priority>
    <Format>MQSTR</Format>
  </MQMD>
</xsl:variable>
<xsl:variable name="MQMDStr2">
  <dp:serialize select="$MQMDStr" omit-xml-decl="yes"/>
</xsl:variable>
<dp:set-variable name='\var://context/EVENTS/_extension/header/MQMD'' value='$MQMDStr2'/>
```

Make sure that the Result Action's INPUT context is set as "EVENTS", the preceding Transform (xform) Action's OUTPUT context is defined as "EVENTS", and the above code snippet is executed using the stylesheet.

**Common MQ processing flows**

There are configuration requirements for processing the MQ Datagram and Request/Reply traffic. For the Datagram traffic, the following **best practice** configurations are suggested:
1. **Datagram traffic using the MPGW Service without units-of-work:** The MPGW service is configured to use the Request rule only. There is no need for the Response rule as the Datagram traffic does not send any reply messages. This service does not need any Error rules if the error processing is not handled by the application. The "Process Backend Errors" under the Advanced tab of the MPGW service should be "off". The Request Type can be "XML", "SOAP", "non-XML", or "pass-thru". The Response Type is set as "pass-thru". During processing, the Request MQMD is not altered and the Backside MQ URL specifies the "RequestQueue" only.

2. **Datagram traffic using the MPGW Service with units-of-work:** The MPGW service is configured to use the Request rule only. There is no need for the Response rule as the Datagram traffic does not send any reply messages. This service may need an Error rule if the error processing is to be handled by the application. The "Process Backend Errors" under the Advanced tab of the MPGW service should be "off". The Request Type can be "XML", "SOAP", "non-XML", or "pass-thru". The Response Type is set as "pass-thru". During processing, the Request MQMD is not altered and the Backside MQ URL specifies the "RequestQueue" along with "Transactional=true" tags. If the Error rule is configured, the poison messages can be routed to an alternate queue by using the Result action or Transform action with the MQ URL containing the "RequestQueue" and "Sync=true" tags. Note that the MQ server has to commit the message once it is put to the alternate queue. Therefore, the "Sync=true" tag is used in the Error rule. The service variable "var://service/error-ignore" should be set as "1" in the Error rule to handle ROLLBACK when "units-of-work" is enabled in the mq-qm object.

3. **Datagram traffic using the MPGW Service with custom error processing:** The MPGW service is configured to use Request, Response, and Error rules. It must capture the response code in the Response rule using "dp:response-header('x-dp-response-code')" and execute "dp:reject" to invoke the Error rule when the "response code" is with a pattern of "2xxx", where "2xxx" is a four digit number representing the MQ return code (mqrc). The "Process Backend Errors" under the Advanced tab of the MPGW service should be "on". The Request Type can be "XML", "SOAP", "non-XML", or "pass-thru". The Response Type is set as "non-XML". During processing, the Request MQMD is not altered and the Backside MQ URL specifies the "RequestQueue" only.

   If units-of-work is enabled in the mq-qm object, the service variable "var://service/error-ignore" should be set as "1" in the Error rule to handle ROLLBACK. The additional tag of "Transactional=true" must be used in the same MQ URL.

List 4 shows the code snippet to capture the response code, and then to execute "dp:reject" if the response code is "2xxx" for the MQ traffic.

**Listing 4. Custom MQ error handling**

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp"
    version="1.0">

    <xsl:output method="xml"/>
    <xsl:template match="/">
        <xsl:variable name="mqrc" select="dp:response-header('x-dp-response-code')"/>
        <xsl:variable name="ecode" select="dp:variable('var://service/error-code')"/>
    </xsl:template>
</xsl:stylesheet>
```
4. **Request/Reply traffic using the MPGW service with units-of-work**: The MPGW service is configured to use Request, Response, and Error rules. It must capture the response code in the Response rule using "dp:response-header('x-dp-response-code')" and execute "dp:reject" to invoke Error rule when "response code" is with a pattern of "2xxx", where "2xxx" is a four digit number representing the MQ return code (mqrc). The "Process Backend Errors" under the Advanced tab of the MPGW service should be "on". The Request and Response Type can be "XML", "SOAP", or "non-XML". During processing, the Request MQMD is not altered and the Backside MQ URL specifies the "RequestQueue", "ReplyQueue", and "setReplyTo=true". When units-of-work is enabled, additional tags of "Transactional=true;Sync=true" must be specified in the same MQ URL. Note that the "Sync=true" tag is used to commit the request message so that MQ application can consume this message to prepare the reply message for the "ReplyQueue". Without this "Sync=true" tag, the message in the "RequestQueue" will not be visible as it is under the sync point flow. The service variable "var://service/error-ignore" should be set as "1" in Error rule to handle ROLLBACK.

**Note**: Make sure that the application, which prepares the reply message, injects "Request.MQMD.Msgid in Reply.MQMD.CorrelId". Otherwise, DataPower does not fetch the reply message in its response rule.

5. **Request/Reply traffic using the MPGW service with dynamic routing**: When the request MQ message is processed, it contains the MQMD.ReplyToQ and MQMD.ReplyToQMgr values to indicate the destination path for the reply message. The reply message will be routed to the final destination based on the following two scenarios:

- **Scenario 1**: If the value of MQMD.ReplyToQMgr is the same as the qmgr name configured in the mq-qm object:
  1. Set DataPower’s internal virtual header "ReplyToQ" to an empty string using the stylesheet in the response rule: `<dp:set-response-header name="ReplyToQ" value="''/>
  2. Save MQMD.ReplyToQ to a context variable in the request rule.
  3. Save MQMD.ReplyToQMgr to a context variable in the request rule.
  4. Inject the MQOD headers with these values in the response rule for the front side client.

5. Make sure that the local MQ queue manager (qmgr) is configured to handle message routing to the remote qmgr based on the MQ Object Descriptor (MQOD).

- **Scenario 2**: If the value of MQMD.ReplyToQMgr is different from the qmgr name configured in the mq-qm object:
  
  1. Set DataPower’s internal virtual header "ReplyToQ" to an empty string using the stylesheet in the Response rule: `<dp:set-response-header name="'ReplyToQ'" value="''"/>`
  2. Set DataPower’s internal virtual header "ReplyToQM" to an empty string using the stylesheet in the Response rule: `<dp:set-response-header name="'ReplyToQM'" value="''"/>`
  3. Save MQMD.ReplyToQ to a context variable in the request rule.
  4. Save MQMD.ReplyToQMgr to a context variable in the request rule.
  5. Inject the MQOD headers with these values in the response rule for the front side client.
  6. Make sure that the local MQ queue manager (qmgr) is configured to handle message routing to the remote qmgr based on the MQ Object Descriptor (MQOD).

Listing 5 shows the stylesheet example to save the "ReplyToQ" and "ReplyToQMgr" values in the Request rule and to inject those values in the Response rule using the MQOD structure.

**Listing 5. Custom MQ routing with the MQOD header**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" extension-element-prefixes="dp"
    exclude-result-prefixes="dp">
    <xsl:output method="xml"/>
    <xsl:template match="/">
        <!-- Request Rule only -->
        <xsl:when test="$rule-type = 'request'">
            <xsl:variable name="entries" select="dp:request-header('MQMD')"/>
            <xsl:variable name="header" select="dp:parse($entries)"/>
            <!-- save ReplyToQ and ReplyToQMgr values -->
            <dp:set-variable name="var://context/MYMQMD/ReplyToQ" value=""$header//ReplyToQ""/>
            <dp:set-variable name="var://context/MYMQMD/ReplyToQMgr" value=""$header//ReplyToQMgr""/>
            <xsl:message dp:priority="debug">
                <xsl:value-of select="concat ('Request MQMD : ', dp:request-header('MQMD'))"/>
            </xsl:message>
        </xsl:when>
        <!-- Response rule only -->
        <xsl:when test="$rule-type = 'response'">
            <xsl:variable name="custMQODStr">
                <MQOD>
                    <Version>2</Version>
                    <ObjectName>
                        <value-of select="dp:variable('var://context/MYMQMD/ReplyToQ')"/>
                    </ObjectName>
                    <ObjectQMgrName>
                        <value-of select="dp:variable('var://context/MYMQMD/ReplyToQMgr')"/>
                    </ObjectQMgrName>
                </MQOD>
            </xsl:variable>
            <xsl:variable name="mqodStr">
                <dp:serialize select="$custMQODStr" omit-xml-decl="yes"/>
            </xsl:variable>
```

**Note:** If the backend MQ server is using a cluster, the injection of MQMD.ReplyToQMgr in MQOD is not needed. However, if the work load management feature is enabled, MQMD.ReplyToQMgr should be injected so that the MQ cluster can route the message to an alternate qmgr when the designated remote qmgr is busy.

### One-phase COMMIT requirements

When the DataPower mq-qm object is enabled with units-of-work, it supports a one-phase COMMIT. In order to provide a guaranteed message delivery, the following conditions must be met:

1. The same MQ qmgr must be used in the MQ front side handlers and MQ URL openers.
2. All processing actions must be synchronous.
3. The same connection is shared across all MQ operations within a transaction that guarantees a "once-and-only-once" message delivery.
4. In all other cases, it is a "at-least-once" message delivery. However, there will be no message loss with a traffic pattern of MQ to MQ. It means both the front and back sides use the MQ protocol.

These two scenarios are:

- **Scenario 1:** When the same MQ qmgr is used in both the front side handler and back side MQ URL opener, the message from the input queue (GETQ) will be resent if the input connection fails. No duplicates are allowed in the in output queue (PUTQ).
- **Scenario 2:** When two separate MQ qmgrs are used in the front side handler and back side MQ URL opener, a message from the input queue (GETQ) will be resent if the input connection fails. Duplicate messages may appear in the output queue (PUTQ). There is no message loss.

### Conclusion

Based on best practice configurations, DataPower applications can provide improved performance for MQ message processing. It creates safe guards for overutilization of appliance resources such as CPU, memory, and connections. This tutorial provided configurations to integrate DataPower services with MQ with various traffic patterns that involve the Datagram and Request/Reply traffic. If the applications are configured with these best practices, they can help appliances from reloads if the MQ server is not available during downtime and can reduce connection errors for the backend MQ server.

### Acknowledgments

Many of our IBM colleagues in the WebSphere DataPower SQA, Development, and Support teams assisted in the preparation of this tutorial. The author would like to extend special thanks to Daniel HT Shih and John Rasmussen for their review of this tutorial.
Resources

Learn

- IBM WebSphere DataPower SOA Appliances product documentation
- IBM Redbook: WebSphere DataPower SOA Appliance Handbook
- IBM Redbook: Strategic overview of WebSphere Appliances
- IBM Redbook: DataPower SOA Appliance Administration, Deployment, and Best Practices
- IBM Redbook: DataPower SOA Appliance Service Planning, Implementation, and Best Practices
- DataPower integration with WebSphere MQ
- Resource management and analysis best practices for WebSphere DataPower
- WebSphere Technical Exchange: Enabling WebSphere MQ Traffic with WebSphere DataPower - Use Case Scenarios
- Integrating WebSphere MQ V7 Publish/Subscribe Feature with WebSphere DataPower
- developerWorks WebSphere site
- WebSphere DataPower discussion forum

Technotes

- DataPower does not inject MQMD.CorrelId in fault message in error rule when request MQMD.CorrelId contains non-zero values for the traffic pattern of MQ-to-HTTP
- How to configure DataPower dynamic MQ URL to use MQ SSL channel
- Configuring DataPower MQ client to use SSL in mutual authentication mode using Key Database (KDB) Method
- The number of concurrent MQ connections in DataPower's MQ front-side handler (FSH)
- When to enable Conversation Sharing in DataPower MQ Queue Manager (mq-qm) object
- WebSphere DataPower spikes with high CPU when WebSphere MQ connection is unavailable
- Gathering WebSphere DataPower CPU and memory information
- WebSphere DataPower SOA Appliance statistics gathering
- How does the front side timeout work?
- What IBM DataPower timeouts are used in specific configuration methods
About the author

Chintamani (Chin) Sahoo

Chin Sahoo is an Advisory Software Engineer and has worked in WebSphere DataPower Appliance and API Management product support for more than five years. He collaborates with the DataPower SQA team to review test cases and develop test plans for product integration with WebSphere MQ, JMS, WebSphere Message Broker, WebSphere Service Registry and Repository (WSRR), and B2B. He also participates in IBM sponsored DataPower forums and social media. He is the team lead and subject matter expert for the MQ components that involves DataPower integration issues with MQ, JMS, WSRR, and B2B.

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)