WebSphere Message Broker V8.0 For Windows 64-bit Performance report

Version 1.0
April 2012

WebSphere Message Broker Development
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Before using this report be sure to read the general information under "Notices".

First Edition, April 2012.
This edition applies to WebSphere Message Broker V8.0 for Windows 64-bit and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended for Architects, Systems Programmers, Analysts and Programmers wanting to understand the performance characteristics of WebSphere Message Broker V8.0 for Windows 64-bit. It is assumed that the reader is familiar with the concepts and operation of WebSphere Message Broker V8.0.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed "as is". The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the customer to evaluate these data and project the results to their operational environment.

The performance data contained in this report was measured in a controlled environment and results obtained in other environments might vary significantly.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- WebSphere MQ
- WebSphere Message Broker
- DB2

The following terms are trademarks of other companies:

- Windows 2008 R2, Windows and Microsoft Corporation

Other company, product, and service names may be trademarks or service marks of others

Summary of Amendments

Date	Changes
April 2012	Initial Release

Feedback

This report and other tools that are produced by the performance group are produced in order to help you understand the performance characteristics of WebSphere Message Broker and to assist you with sizing.

It is important that the reports and tools are effective in what they do and it is very useful to have feedback on the content and style of the information which is produced. Your comments, both positive and negative, are therefore welcome.

Your answers to the following questions are particularly interesting:

- What are your most common performance questions?
- Do the reports provide what is needed?
- Is there any other performance information which is required to help you do your job?
- Would you like to see any other aspects of WMB performance discussed?

Please supply feedback to us at the following e-mail addresses:

- Tim Dunn (dunnt@uk.ibm.com)
- Dave Gorman(gormand@uk.ibm.com)
- Rob Convery (convery@uk.ibm.com)
- Asheesh Tiwari (ashetiwa@in.ibm.com)
- Manjunath Veerabhadraiah (manjuvee@in.ibm.com)
- Peter Prince (PETERPRI@uk.ibm.com)

or use the feedback facility on the SupportPac web page where you obtained this report.

Table of contents

Introduction

The purpose of this report is to illustrate the key processing characteristics of WebSphere Message Broker. This has been done by measuring the message throughput that is possible for a number of different types of message processing. Throughout this report, the term "message" is used in a generic sense, and can mean any request or response into or out of the broker, regardless of the transport or protocol.

This report covers multiple message formats, type, sizes and consists of three parts. These meet the following different requirements:

1. **Part I** contains the release highlights and background information to help understand the context of the results. Results are presented at a high level and are intended to help you quickly understand WebSphere Message Broker throughput capabilities. It shows:
    - The areas of improvement in performance with WebSphere Message Broker V8.0 when compared with WebSphere Message Broker V7.
    - The level of message throughput that is achievable when using WebSphere Message Broker in different ways. These tests use multiple copies of the message flow and utilise as much of the server as possible to illustrate the maximum message rate which can be sustained for the individual types of processing.

2. **Part II** contains measurement data for a wide variety of tests which examine the processing costs of individual functions using a **single** copy of the message flow. This information is provided for those who wish to understand the processing costs of different capabilities within WebSphere Message Broker. This information is intended for the more experienced WebSphere Message Broker user who is familiar with the product concepts and functions. As these tests run a single copy of the message flow, they do not utilise the whole of the server and do not therefore represent the maximum message throughput that is achievable.

3. **Appendices** that contain supplementary information. They are:
    - Appendix A - Measurement Environment
    - Appendix B - Evaluation Method
    - Appendix C - Test Messages
    - Appendix D - Use Case Descriptions
    - Appendix E - Tuning

There are a number of changes from previous performance reports. The most significant are:

1. Re-engineered tests to better reflect the processing costs which are encountered when processing messages with a WebSphere Message Broker message flow. The previous tests are deprecated and do not appear in this report.
2. Larger range of message sizes including a greater range of persistent message sizes.

The performance measurements focus on the throughput capabilities of

the broker using different message formats and processing node types. The aim of the measurements is to help you understand the rate at which messages can be processed in different situations as well as helping you to understand the relative costs of the different node types and approaches to message processing.

You should not attempt to make any direct comparisons of the test results in this report with what may appear to be similar tests in previous performance reports. This is because the contents of the test messages are significantly different as is the processing in the tests. It is not meaningful to make such comparisons. In many cases the hardware, operating system and prerequisite software are also different, making any direct comparisons invalid.

Some optimisations of the test environment and procedures have been implemented to minimise the effect of logging for example and to ensure that messages do not accumulate on output queues which has a detrimental effect on message throughput. These are detailed in Appendix E - Tuning.

In many of the tests the user logic used is minimal so the results presented represent the best throughput that can be achieved for that node type. This should be borne in mind when sizing WebSphere Message Broker.

Part I

This part contains an overview of the areas of improvement in performance with WebSphere Message Broker V8.0 when compared with WebSphere Message Broker V7.

It contains the following sections:

- Release highlights, which outlines the main differences in performance when using WebSphere Message Broker V8.0 compared with WebSphere Message Broker V7.
- Additional information, which provides links to other sources.

Release highlights

Performance Improvements over WebSphere Message Broker V7

There have been significant improvements in the performance of WebSphere Message Broker V8.0 in the following areas:

- Message parsing and serialisation
- Graphical Mapping
- Performance Analysis of message flows using Resources Statistics and Activity Log

Details of the improvements follow:

Message parsing and serialisation

V8 introduces the new DFDL (Data Format Description Language) parser and serialiser. DFDL is a new industry standard for binary, text and industry data formats, and can be exploited by all broker nodes. This new parser has excellent performance characteristics. DFDL parse and serialisation improvements have been measured up to 70% when compared to existing MRM technology, with a typical improvement of 50%. Support for DFDL includes built in facilities within the WebSphere Message Broker Toolkit to easily model data and test without the need for deployment to the runtime, reducing development costs. All existing parser technologies continue to exist in V8.

Graphical Mapping

V8 introduces a new Mapping node which allows the user to visually map and transform data from source to target. This new mapping node has excellent performance characteristics, and is a viable option for performance sensitive transformations. Some tests have been measured performing close to optimised programmatic transformations in ESQL, Java and .Net, with the typical measurement being 50%. Existing maps developed prior to V8 will continue to work as-is, but will be opened as read-only within the WebSphere Message Broker Toolkit. At the time of writing this report, automatic migration of existing maps is not possible.

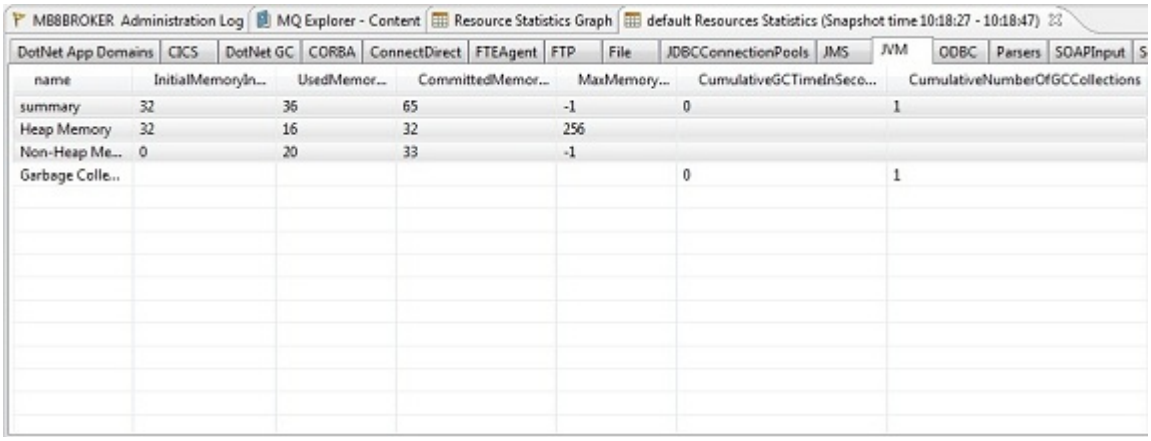
Performance Analysis of message flows using Resources Statistics and Activity Log

In WebSphere Message Broker V8.0 there have been continued enhancements to assist with understanding message flow activity and the diagnosis of runtime performance problems. Since WebSphere Message Broker V7 it has been possible to very quickly, accurately and cheaply diagnose performance problems in a running message flow using Resource Statistics, with a performance overhead that is typically no more than 3%. V8 adds many more resource managers which emit resource statistic data.
In addition, a new Activity Log has been added to help the user quickly understand what message flows are doing in the runtime, and how they are interacting with external resources. Collection is always on and overheads are negligible.

Resource Statistics

In V8 the number of resource managers emitting statistical data to report activity has been increased. The list now includes: .Net App domains, CICS, .Net GC, CORBA, ConnectDirect, FTEAgent, FTP, File, JDBCConnectionPools, JMS, JVM, ODBC, Parsers, SOAPInput, Security, Sockets, TCPIPClientNodes and TCPIPServerNodes. All of these can be visualised in the WebSphere Message Broker Explorer. The overhead of these functions is small.

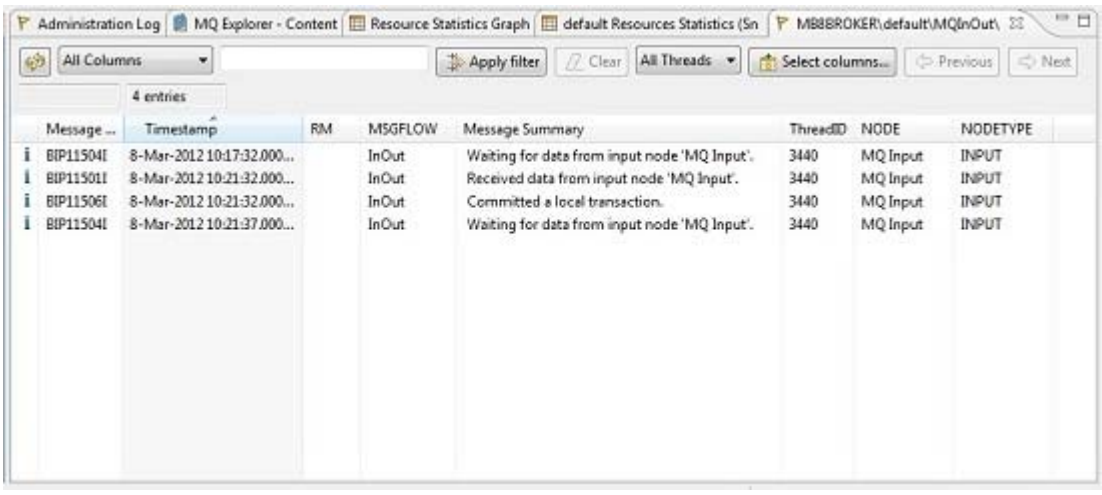
A screenshot of the resource statistics facility being used is shown below. For more information on this facility consult the product documentation.



Activity Log

V8 introduced Activity logs. These help users to understand what their message flows are doing by providing a high-level overview of how the broker runtime interacts with external resources. Similar to resource statistics, activity log data can be visualised in the WebSphere Message Broker Explorer and has no runtime overhead.

A screenshot of the activity log facility being used is shown below. For more information on this facility consult the product documentation.



Use case throughput

This section illustrates the message throughput that is possible with WebSphere Message Broker V8.0 for a number of common processing use cases. A range of processing rates will be observed. The message rate varies with the complexity of the application processing and the size and complexity of the messages as it would for any program.

Use Case Outline

This section contains a brief outline of the tests used and the results for each are presented in the tables below. For more detail about individual test cases see Appendix D - Use Case Descriptions.

- **Aggregation**

  This represents the type of processing that is required for example, when travel is booked and arrangements for a flight, hotel, car and money must be made. Requests to four different applications are made and the replies consolidated into a single reply. This test performs the processing required to split an incoming XML message and perform a four message aggregation using the Aggregation nodes that are supplied with WebSphere Message Broker.

- **Coordinated Request Reply**

  This performs the processing needed to enable two applications with different message formats to communicate with each other. One application has a message format of self-defining XML and the other uses Custom Wire Format (CWF) messages. The request and reply processing for a particular request must be coordinated so that data from the original request is restored to the reply message.

- **Data Warehouse**

  This demonstrates a scenario in which a message flow is used to perform the archiving of data, such as sales data, into a database. The data is stored for later analysis by another message flow or application.

- **Large Messaging Processing**

  This is based on the scenario of end-of-day processing of sales data. Messages representing sales for the day are batched together for transmission to the IT centre. On receipt at the IT centre the batched messages are split into their constituent parts for subsequent processing.

- **Message Routing**

  This shows how a message flow can be used to route messages to different WebSphere MQ queues based on data stored in a database table. This is a commonly used scenario which is applicable to many different industries and applications.

- **Message Transformation**

  This shows XML messages that are sent and received over the WebSphere MQ transport and tranfsormed from one format to another using ESQL.

The following table shows the message rates that were obtained for the different use cases when running on a IBM xSeries x3690 X5 with 2 x Deca-Core Intel Xeon E7-2860 2.27GHz processors with HyperThreading

turned on

Use Case	Message Size	V8.0 Msgs/sec
Aggregation	20kB	3056
Coordinated Request/Reply	2kB	9290
Data Warehouse	2kB	2226
Large Message Processing	2kB	21978
Message Routing	2kB	52666
Message Transformation	2kB	14067

NOTE: The results in this table were obtained by running sufficient copies of each message flow so that in most cases the system CPU utilisation was 80% or greater.

Note that there is a range of message rates from 2,226 to 52,666 messages per second. These rates reflect the range of complexities for the different use cases shown or, put in another way, not all use cases are of the same complexity. For example, the aggregation test case includes the processing of 10 MQ messages across 6 flows, whilst the routing cache test includes 2 MQ messages and across 1 flow. The aggregation test case requires a minimum message size which is why it is run with a message size of 20k.

When planning a system it is important to understand the complexities of the processing required so that adequate resources can be provided to meet the requirements of the particular situation.

Additional information

This section contains links to information about WebSphere Message Broker and associated products.

The Web Resources section in the development toolkit of WebSphere Message Broker V8.0 contains links to many additional pieces of information on topics such as Education, Technical Resources and SupportPacs. The Web resources section can be accessed by selecting Web Resources from the Help menu on the development toolkit menu bar.

For additional suggestions consider the following:

- See the announcement letter for IBM WebSphere Message Broker V8.0 which is available at http://www.ibm.com/support/docview.wss?uid=swg21566990
- IBM WebSphere MQ and Message Broker SupportPacs provide you with a wide range of downloadable code and documentation that complements the WebSphere MQ family of products. Additional performance reports are also available. These are available at http://www.ibm.com/software/integration/support/supportpacs
- For more information about WebSphere Message Broker V8.0, including a trial edition, go to the WebSphere Message Broker Web site. Product documentation is also available. This is available at http:/www.ibm.com/software/integration/wbimessagebroker
- For more information about WebSphere MQ V7, go to the WebSphere MQ website. Product documentation is also available. This is available at http://www.ibm.com/software/integration/wmqfamily
- For more information about business integration software from IBM go to the WebSphere Business Integration website. This is available at http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/businessint
- Get the latest WebSphere Message Broker technical resources at the WebSphere Message Broker zone. This is available at http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html
- The MQ, JMS, and HTTP transport testing which was run for this report used a tool called the Performance Harness for JMS to generate and consume messages. The tool is useful as a simple way to send and receive messages. The documentation for the tool contains examples of how to run it to send and receive messages. More information about the currently available version can be found at: http://www.alphaworks.ibm.com/tech/perfharness?open&S_TACT=105AGX21&S_CMP=AWRSS.

Part II

This section contains the description and results of a series of tests that have been run to identify the processing costs of a selected range of the functions that are provided with WebSphere Message Broker.

It contains the following:

- Processing Profiles, which describes the tests and shows the results obtained when a single copy of the message flow was run.
- Resource Requirements, which provides guidance on memory use for execution groups running a variety of message flows.

Processing profiles

This section contains the results of a series of micro tests which illustrate the costs of performing different types of processing using WebSphere Message Broker such as message parsing, message streaming, and using filter nodes. These micro tests are not intended to represent applications. They are an illustration of the processing costs of specific functions.

The test results were all run using the same methodology. This was to run a single copy of the message flow (unless specified otherwise) to maximum CPU utilisation and to observe the message rate obtained. From this a CPU cost per message was calculated. This is presented in the results table for each measurement.

When comparing the costs of different functions it is recommended to compare them on the basis of CPU cost per message rather than message rate.

There are many comparisons which can be made using the data in this section which will give some insight into the relative costs of different implementations such as the relative cost of ESQL and XSLT to process the same message.

The data in this section will allow you to make a comparison only on the basis of CPU costs. Other factors such as the potential for code re-use and the operational considerations of using a particular technology are not discussed.

Messages Used in Processing

For the majority of tests the message content was common. The following different formats of the common input message content were used: XML, non-XML fixed/variable (CWF) and non-XML delimited (TDS). In these cases the input message still contained the same amount of information, it was the representation of the data that was different. The output message varied depending on the test case. The messages are described in Appendix C - Test Messages.

Note that the message size quoted is based on the size of the data in XML format. When the same data is represented as non-XML the actual size may be significantly less. The sizes for the different formats are shown in the table below:

Msg Size in XML	Msg Size in non-XML delimited	Msg Size in non-XML fixed/variable
2kB	2kB	1kB
20kB	20kB	5kB
200kB	100kB	48kB
2048kB(2MB)	995kB	4481kB
20480kB(20MB)	9941kB	4807kB

Results Presentation

Each of the tests are described below and accompanied by a table of data which has a format such as this:

Msg Size	Persistent	Message Rate (Msgs/sec)	% CPU Busy	CPU ms/msg	Instances

Msg Size	Persistent	Message Rate (Msgs/sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE				
20kB	NONE				
200kB	NONE				
2MB	NONE				
20MB	NONE				
2kB	FULL				
20kB	FULL				
200kB	FULL				
2MB	FULL				
20MB	FULL				

The data in the columns is as follows:

Msg Size: Records the approximate size of the message used as input to the test. This is the size of the XML or equivalent non-XML message payload and does not include the size of any message header. Most test cases used messages of 2kB, 20kB, 200kB, 2MB, and 20MB. In some cases a more limited range of message sizes was run where the test was not suitable for the whole range of message sizes.

Persistent: Indicates whether the messages used in the test were persistent or not. This is applicable to MQ messages only. Where message persistence doesn't apply this column will contain NONE.

Message Rate: The number of round trips or message flow invocations per second.

% CPU Busy: System busy CPU percentage on the server machine. This includes the CPU used by all processes (WebSphere Message Broker, WebSphere MQ queue manager, database manager etc) on the system under test. The rate is expressed as a percentage utilisation of all processors on the machine.

CPU ms/msg: Overall CPU cost per message, expressed as CPU milliseconds per message.

The value of CPU ms/msg is obtained using the calculation:

((Number of cores * 1000) * (%CPU/100)) / Message Rate.

This cost includes WebSphere Message Broker, WebSphere MQ, DB2, operating system costs etc. The CPU ms/msg figures reported are specific to the system on which they were obtained and if projections of message processing capacity are to be made for other systems a suitable adjustment must be made to allow for differences in the capacity of the two systems.

Instances: The total number of instances (threads) with which the message flow was tested with.

Response Times

Response time data for the message flow execution is not reported. The tests are configured to maximise message throughput and minimise CPU costs and so there are always a number of messages waiting on the input node of the message flow so that there is at least one message ready to be processed immediately after processing of the current message has

completed. This means that the processing of each message involves queuing time at the input node. Because of this it is not meaningful to report message processing times as observed by the client as it will not reflect the true execution time in the message flow. It is possible to estimate the elapsed time within a message flow in milliseconds from the results of these tests by dividing 1000 (representing the number of milliseconds in 1 second) by the message rate for the test as only a single copy of the message flow was run. For example, suppose that a test achieved a message rate of 2000 per second. The message flow average execution time is 1000 / 2000 = 0.5ms. For a message rate of 200 per second the average execution time is 1000/200 = 5ms.

These times are an estimate of the execution time in the message flow and represent the elapsed time between the message being read from the input queue and the result being placed on the output queue.

If messages are generated or consumed by remote clients an allowance must be made for network delays.

The test descriptions and results follow.

Sending and Receiving Messages over different Transports

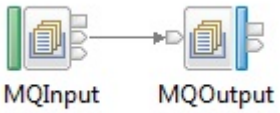
The tests in this section illustrate the processing cost of receiving and sending data over various transports supported by WebSphere Message Broker.

MQ Nodes

The tests in this section illustrate the processing cost of reading and writing MQ Messages.

Reading and writing to an MQ Queue

This test consists of:



This test shows the overhead of using message broker to move messages from one WebSphere MQ queue to another. As many of the other tests included in this report use WebSphere MQ as the transport it can be used to determine how much of the processing incurred is due to using WebSphere MQ and how much is the routing/transformation or parsing cost.

A WebSphere MQ message is placed on the Input Queue where the incoming message is then copied to the Output Queue. The message contents are treated as a BLOB and are not modified or parsed in any way.

This test identifies the cost of reading and writing a BLOB message with WebSphere MQ as the transport.

The results of running this test are given in the table below.

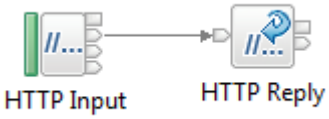
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	3940.6	5.6	0.6	1
20kB	NONE	3301.6	5.6	0.7	1
200kB	NONE	1484.8	6.1	1.6	1
2000kB	NONE	97.0	7.4	30.6	1
20000kB	NONE	5.8	4.7	323.0	1
2kB	FULL	718.1	1.5	0.9	1
20kB	FULL	555.3	2.1	1.5	1
200kB	FULL	123.4	1.6	5.0	1
2000kB	FULL	13.5	1.4	42.0	1
20000kB	FULL	1.1	1.5	515.1	1

HTTP Nodes

The tests in this section illustrate the processing cost of reading and writing HTTP messages.

Reading and Writing messages over the HTTP Transport

This test consists of:



This test shows the overhead of using WebSphere Message Broker to receive and send messages over the HTTP transport. An HTTP bytes message is written to the broker over HTTP. The incoming message is then written out unmodified back to the client. The message contents are treated as a BLOB and are not modified or parsed in anyway. Note that persistent HTTP connections were used in this test (see Tuning Section for details)
This test identifies the cost of reading and writing a BLOB message with HTTP.
The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2926.5	8.2	1.1	1
20kB	NONE	1594.8	26.7	6.7	1
200kB	NONE	298.2	39.2	52.6	1
2000kB	NONE	33.5	23.4	279.6	1

SOAP Nodes

The tests in this section illustrate the processing cost of receiving, sending, and making requests over the SOAP transport.

Receiving and sending messages over the SOAP transport

This test consists of:

The message flow is acting as a web service provider. A SOAP message is received by the broker via the SOAPInput node. A Compute Node then copies the SOAP request message to a SOAP response message which is then sent via the SOAPReply node.

Note that persistent HTTP connections were used in this test (see Tuning Section for details).

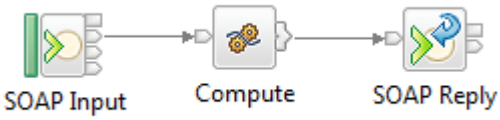
This test identifies the cost of receiving and sending a SOAP message.
The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1109.0	1.7	0.6	1
20kB	NONE	348.6	0.9	1.1	1
200kB	NONE	55.3	1.5	10.5	1

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2000kB	NONE	4.6	3.2	278.3	1
20000kB	NONE	0.3	2.7	3133.3	1

Receiving and sending messages over the SOAP transport with Validation enabled

This test consists of:



The SOAPInput and SOAPReply nodes are are used in a message flow which implements a provider web service. The SOAPInput node is configured to enable validation ("Content and value"; SOAP Parser Options select "Build tree using XML schema data types"). A SOAP message is received by the broker via the SOAPInput node. A Compute node then copies the unmodified SOAP request message across to a SOAP response message, which is then sent via the SOAPReply node.

Note that persistent HTTP connections were used in this test (see Tuning Section for details).

This test identifies the cost of receiving and sending a SOAP message where the message is validated.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	905.0	4.0	1.8	1
20kB	NONE	187.2	3.4	7.4	1
200kB	NONE	15.4	3.4	88.9	1
2000kB	NONE	1.1	3.4	1256.2	1
20000kB	NONE	0.2	2.9	7211.1	1

Receiving and sending messages over the SOAP transport using WS-Addressing

This test consists of:

The SOAPInput and SOAPReply nodes are used in a message flow which acts as a web services provider. The SOAPInput node is configured to enable WebServices Addressing (WS-Addressing). A SOAP message is received by the broker via the SOAPInput node. A Compute Node then copies the unmodified SOAP request message to a SOAP response message, which is then sent via the SOAPReply node.

Note that persistent HTTP connections were used in this test (see Tuning

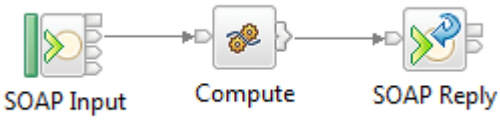
Section for details).

This test identifies the cost of receiving and sending a SOAP message where WS-Addressing is enabled. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	659.4	4.1	2.5	1
20kB	NONE	215.7	3.9	7.2	1
200kB	NONE	24.1	4.2	69.0	1
2000kB	NONE	2.0	3.7	743.6	1
20000kB	NONE	0.2	3.3	5958.6	1

Receiving and sending messages over the SOAP transport with SwA attachments

This test consists of:



The SOAPInput and SOAPReply nodes are used in a message flow which implements a provider web service. A SOAP message with an attachment is received by the broker via the SOAPInput Node. A Compute node then copies the unmodified SOAP request message to a SOAP response message, which is then sent via the SOAPReply node. The size of the SOAP message body remains constant and the size of the attachment is increased.

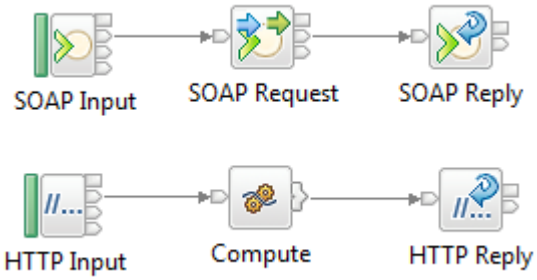
Note that persistent HTTP connections were used in this test (see Tuning Section for details).

This test identifies the cost of receiving and sending a SOAP with Attachments (SwA) message.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	656.6	1.7	1.0	1
20kB	NONE	592.4	2.6	1.8	1
200kB	NONE	177.3	3.5	8.0	1
2000kB	NONE	16.0	4.0	99.9	1

Making a SOAP Request

This test consists of:



The SOAPRequest node is used in a message flow to invoke a web service synchronously. A response must be received from the web service before the message flow continues.

A SOAP message is received by the broker via the SOAPInput node. A SOAPRequest node then issues a Web Service request. When the web service has completed processing a response is sent to the original request via the SOAPReply node.

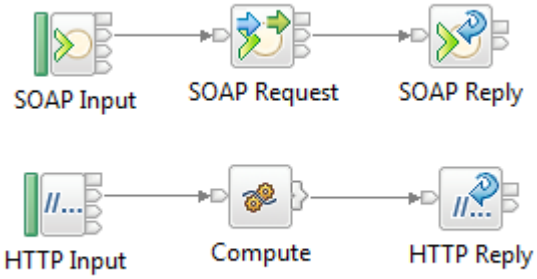
The web service that is invoked synchronously consists of an HTTP message flow, running in the same broker, which returns the request data unmodified.

This test identifies the cost of making a web service request via the SOAPRequest node. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	327.6	3.8	4.7	1
20kB	NONE	97.3	2.7	11.2	1
200kB	NONE	15.9	3.7	91.8	1
2000kB	NONE	1.8	4.6	1002.9	1
20000kB	NONE	0.2	3.9	7018.5	1

Making a SOAP Request with Validation enabled

This test consists of:



The SOAPRequest node is used in a message flow that calls a web service synchronously. This means the node sends a Web Service request and waits for the associated web service response to be received before the message flow continues.

The SOAPRequest node is configured to enable validation ("Content and

value"; SOAP Parser Options select "Build tree using XML schema data types"). A SOAP message is received by the broker via the SOAP Input Node. A SOAP Request Node makes a web service request, and the response is sent via the SOAPReply node. The request is returned, unmodified, via a HTTP flow.

This test identifies the cost of making a web service request via the SOAPRequest node with validation enabled.

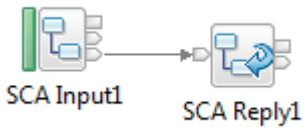
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	213.6	4.3	8.1	1
20kB	NONE	43.4	4.2	38.7	1
200kB	NONE	3.9	3.8	382.8	1
2000kB	NONE	0.7	3.8	2319.8	1
20000kB	NONE	0.1	1.4	9252.4	1

SCA Nodes

The tests in this section illustrate the processing cost of using the SCA nodes, which allow WebSphere Message Broker to interoperate with WebSphere Process Server. This supports both WebSphere Process Server to WebSphere Message Broker inbound scenarios and WebSphere Message Broker to WebSphere Process Server outbound scenarios. When designing message flows with SCA, you must select a suitable transport (also called a binding). The SCA nodes support the WebSphere MQ transport and WebSphere Broker HTTP transport with SOAP. These tests show the processing costs of SCA over each of these transports.

Receiving and sending inbound messages with SCA over WebSphere MQ transport

This test consists of:

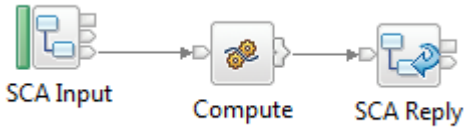


The bindings are set to MQ. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	3126.8	3.5	0.5	1
20kB	NONE	2538.6	4.2	0.7	1
200kB	NONE	1352.6	4.4	1.3	1
2000kB	NONE	54.1	5.2	38.7	1
20000kB	NONE	4.4	4.4	401.5	1

Receiving and sending inbound messages with SCA over SOAP

This test consists of:



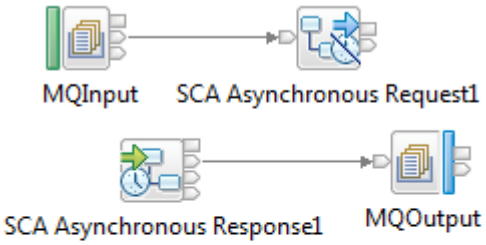
The bindings are set to Web Services and the Compute node modifies the high level tag from request to response.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	714.7	4.0	2.2	1
20kB	NONE	303.0	1.1	1.5	1
200kB	NONE	41.3	3.5	33.9	1
2000kB	NONE	3.8	3.8	400.2	1
20000kB	NONE	0.3	2.9	3656.1	1

Making an Asynchronous outbound request with SCA over WebSphere MQ transport

This test consists of:

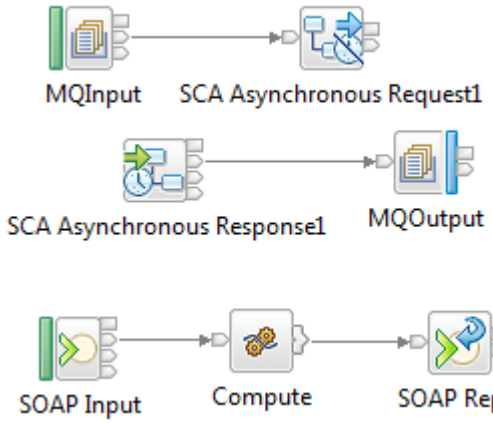


The bindings are set to MQ. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1806.5	5.6	1.2	1
20kB	NONE	1557.4	6.9	1.8	1
200kB	NONE	935.7	8.7	3.7	1
2000kB	NONE	30.7	9.5	124.2	1
20000kB	NONE	2.8	6.8	962.8	1

Making an Asynchronous outbound request with SCA over SOAP

This test consists of:

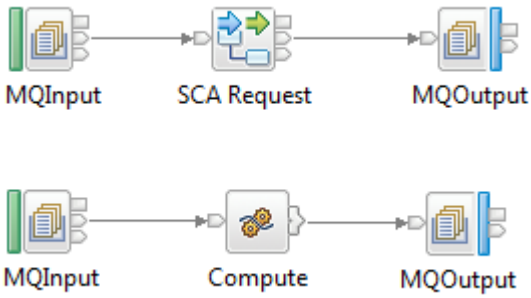


The bindings are set to Web Services.The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	243.0	6.7	11.1	1
20kB	NONE	96.4	3.0	12.3	1
200kB	NONE	13.6	7.5	221.2	1
2000kB	NONE	1.2	6.9	2304.4	1

Making a Synchronous outbound request with SCA over WebSphere MQ transport

This test consists of:



The bindings are set to MQ. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	945.2	4.5	1.9	1
20kB	NONE	802.3	4.3	2.1	1
200kB	NONE	492.5	5.3	4.3	1
2000kB	NONE	32.5	8.5	104.3	1
20000kB	NONE	3.2	6.9	862.0	1

Making a Synchronous outbound request with SCA over SOAP

This test consists of:

The bindings are set to SOAP. The results of running this test are given in the table below.

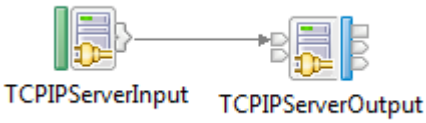
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	209.9	4.0	7.6	1
20kB	NONE	68.8	1.9	11.2	1
200kB	NONE	8.1	3.6	180.1	1
2000kB	NONE	1.0	4.1	1644.5	1
20000kB	NONE	0.1	3.6	10950.4	1

TCPIP Nodes

The tests in this section illustrate the processing cost of using the TCPIP nodes.

Receiving and sending messages over TCPIP using Fixed Length record detection

This test consists of:



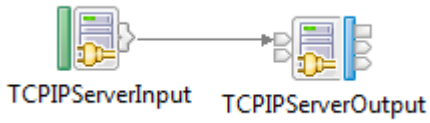
This test illustrates the cost of using the Fixed Length record detection in the TCPIPServerInput node. The incoming message is received by the TCPServerInput node and the response sent from the TCPIPServerOutput node. The messages processed are in the XMLNSC domain and record detection was set to Fixed Length on the TCPIPServerInput node.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1286.0	4.3	1.3	1
20kB	NONE	1006.2	4.6	1.8	1
200kB	NONE	465.8	6.9	6.0	1
2000kB	NONE	11.3	2.0	69.2	1

Receiving and sending messages over TCPIP using Parsed Record Sequence record detection

This test consists of:



This test illustrates the cost of using parsed record sequence detection in the TCPIPServerInput node. The incoming message is received by the TCPServerInput node and the response sent from the TCPIPServerOutput node. The messages processed were in the XMLNSC domain and record detection was set to Parsed Record Sequence on the TCPIPServerInput node.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1146.8	3.8	1.3	1
20kB	NONE	491.0	3.6	3.0	1
200kB	NONE	66.8	3.7	21.9	1
2000kB	NONE	4.9	0.6	46.6	1
20000kB	NONE	0.5	1.5	1108.0	1

Message parsing and writing

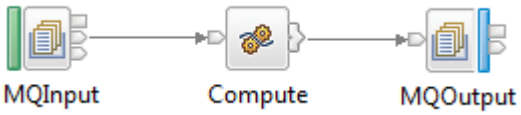
The tests in this section illustrate the cost of parsing input messages and writing output messages for different message formats.

Parsing a Message in the MRM Domain

The tests in this section illustrate the CPU processing costs of parsing different message formats in the MRM domain.
In this report only figures for TDS Fixed Length format are given. Previous measurements showed that message throughput varied little regardless of format hence the decision to report only one type.

Parsing a Tagged Delimited String, Fixed Length Input Message

This test consists of:



The input message is processed with the TDS domain.

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes the incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing a Fixed Length, Tagged Delimited String input message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1152.0	3.9	1.4	1
20kB	NONE	288.4	3.4	4.7	1
200kB	NONE	35.9	3.2	35.8	1
2000kB	NONE	3.3	3.3	400.9	1
20000kB	NONE	0.3	3.2	4397.0	1
2kB	FULL	255.8	3.6	5.6	1
20kB	FULL	155.0	3.3	8.6	1
200kB	FULL	31.3	3.3	41.7	1
2000kB	FULL	3.0	3.2	431.0	1
20000kB	FULL	0.3	3.3	4597.0	1

Parsing a Custom Wire Format Input Message

This test consists of:

The input message is processed with the CWF domain.

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes the incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing a Custom Wire Format input message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1001.2	3.7	1.5	1
20kB	NONE	356.8	3.4	3.8	1
200kB	NONE	44.1	3.3	29.8	1
2000kB	NONE	3.9	3.2	321.7	1
2kB	FULL	610.5	3.5	2.3	1
20kB	FULL	165.3	3.1	7.5	1
200kB	FULL	36.8	3.3	35.6	1
2000kB	FULL	3.9	3.2	321.6	1

Parsing a SWIFT 543 Input Message using the Tagged Delimited String Parser

This test consists of:

The input message is processed with the TDS domain.

Within the Compute node the message headers from the incoming message are copied over to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes the incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing a SWIFT MT543 message using the Tagged Delimited String format. A single implementation of this message was used which was approximately 7KB in size. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
6543	NONE	773.6	3.6	1.9	1
6543	FULL	449.7	3.6	3.2	1

Parsing and Writing a SWIFT 543 Input Message using the Tagged Delimited String Parser

This test consists of:

The input and output message are processed with the TDS domain.

Within the Compute node the Envelope within the incoming SWIFT Message is copied over to the outgoing message. This causes the incoming message to be fully parsed and the outgoing message to be serialized.

This test identifies the cost of parsing a SWIFT MT543 message and serializing it again using the Tagged Delimited String format. A single implementation of this message was used which was approximately 7KB in size. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
6543	NONE	536.5	1.0	0.7	1
6543	FULL	238.9	2.3	3.8	1

Parsing an HL7 Input Message using the Tagged Delimited String Parser

This test consists of:

The input message is processed with the TDS domain.

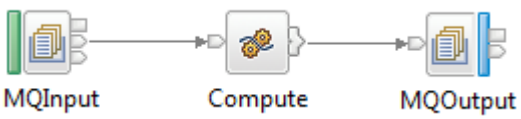
Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing an HL7 input message using the Tagged Delimited String format. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
16384	NONE	271.2	3.4	5.1	1
16384	FULL	193.7	3.4	7.1	1

Parsing and Writing a HL7 Input Message using the Tagged Delimited String Parser

This test consists of:



The input and output message are processed with the TDS domain.

Within the Compute node the Envelope within the incoming HL7 Message is copied over to the outgoing message. This causes the incoming message to be fully parsed and the outgoing message to be serialized.

This test identifies the cost of parsing a HL7 message and serializing it again using the Tagged Delimited String format. A single implementation of this message was used which was approximately 1KB in size. The results of running this test are given in the table below.

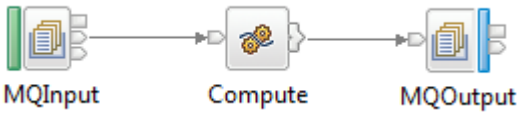
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
16384	NONE	86.6	1.1	5.1	1
16384	FULL	54.5	1.0	7.6	1

Writing a Message in the MRM Domain

The tests in this section illustrate the CPU processing costs of creating an output message with different formats in the MRM domain. This is the processing associated with taking a message tree in OutputRoot and flattening it to create a bitstream which is the output message.

Writing a Tagged Delimited String, Fixed Length Output Message

This test consists of:



The input message is processed with the XMLNSC domain. The output message is processed using the TDS domain.

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition the incoming Generic XML message is converted to a Fixed Length, Tagged Delimited String outgoing message. This causes the incoming message to be fully parsed and payload which is then written as the payload of the output message.

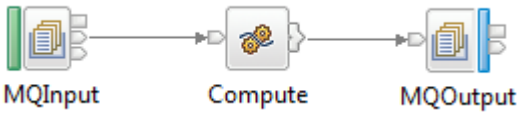
This test identifies the cost of parsing a Generic XML message and writing out a Fixed Length, Tagged Delimited String output message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1184.6	3.8	1.3	1
20kB	NONE	203.2	3.3	6.5	1
200kB	NONE	25.5	3.3	52.1	1
2000kB	NONE	2.0	3.2	657.0	1
20000kB	NONE	0.2	3.2	6683.9	1
2kB	FULL	150.7	3.2	8.6	1
20kB	FULL	107.4	3.2	12.1	1
200kB	FULL	19.1	3.2	66.7	1

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2000kB	FULL	1.4	3.3	942.8	1
20000kB	FULL	0.2	3.2	6727.2	1

Writing a Custom Wire Format Output Message

This test consists of:



The input message is processed with the XMLNSC domain. The output message is processed using the CWF domain.

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition the incoming Generic XML message is converted to a Custom Wire Format outgoing message. This causes the incoming message to be fully parsed payload which is then written as the payload of the output message.

This test identifies the cost of parsing a Generic XML message and writing out a Custom Wire Format output message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1559.0	3.8	1.0	1
20kB	NONE	302.6	3.2	4.2	1
200kB	NONE	39.4	3.3	33.6	1
2000kB	NONE	3.0	3.1	414.8	1
20000kB	NONE	0.3	3.3	4092.8	1
2kB	FULL	219.6	3.2	5.8	1
20kB	FULL	138.6	3.1	9.1	1
200kB	FULL	28.5	3.2	44.4	1
2000kB	FULL	2.0	3.2	645.4	1
20000kB	FULL	0.3	3.3	4519.7	1

Parsing Messages in the XMLNSC Domain

The tests in this section illustrate the CPU processing costs of parsing different message formats in the XMLNSC domain.

Parsing an XML Input Message

This test consists of:

Within the Compute node the message headers from the incoming

message are copied to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes the incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing an XML input message. Because there is no message body on the output message there are no writing costs. The results of running this test are given in the table below.

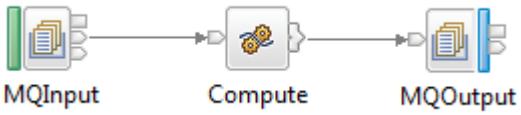
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1523.9	4.0	1.1	1
20kB	NONE	961.7	3.6	1.5	1
200kB	NONE	108.8	3.3	12.2	1
2000kB	NONE	10.8	3.4	126.6	1
20000kB	NONE	1.0	3.4	1359.4	1
2kB	FULL	520.4	3.7	2.9	1
20kB	FULL	280.3	3.5	4.9	1
200kB	FULL	93.9	3.4	14.5	1
2000kB	FULL	9.8	3.5	141.8	1
20000kB	FULL	0.9	3.3	1416.9	1

Writing a Message in the XMLNSC Domain

The test in this section illustrates the CPU processing costs of using the XMLNSC domain to create an output message. This is the processing associated with taking a message tree in OutputRoot and flattening it to create a bitstream which is the output message.

Writing a Generic XMLNSC Output Message

This test consists of:



Within the Compute node the entire incoming message is copied to the outgoing message. In addition the last element in the incoming message is modified. This causes the incoming message to be fully parsed which is then written as the payload of the output message.

This test identifies the cost of parsing a Generic XML input message and writing a modified XML output message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1412.8	2.4	0.7	1
20kB	NONE	555.8	2.5	1.8	1
200kB	NONE	54.1	1.0	7.2	1
2000kB	NONE	4.0	1.4	142.4	1
2kB	FULL	300.9	3.1	4.2	1
20kB	FULL	159.1	2.7	6.9	1

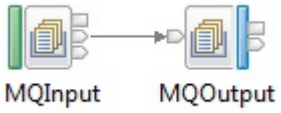
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
200kB	FULL	29.3	2.9	39.2	1
2000kB	FULL	3.1	3.2	407.5	1

Validation in the XMLNSC Domain

The test in this section illustrates the CPU processing costs of using the XMLNSC domain to validate an XML message. This is the processing associated with taking a message and validating it against an associated XML Schema.

Validating an XML Message on Input

This test consists of:



The MQInput node is set to validate the message contents and value. This causes the incoming message to be fully parsed and validated which is then written unmodified as the payload of the output message.

This test identifies the cost of validating an XML input message in the input node and writing an unmodified XML output message.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2007.0	4.2	0.8	1
20kB	NONE	830.2	3.5	1.7	1
200kB	NONE	96.3	3.4	13.9	1
2000kB	NONE	4.9	3.4	281.5	1
2kB	FULL	503.0	3.4	2.7	1
20kB	FULL	235.5	3.1	5.2	1
200kB	FULL	36.2	3.0	33.4	1
2000kB	FULL	3.9	3.0	305.9	1

Validating an XML Message mid flow

This test consists of:

The Validate node is set to validate the message contents and value. This causes the incoming message to be fully parsed and validated, which is

then written unmodified as the payload of the output message.

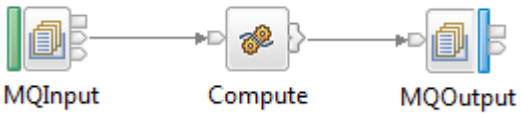
This test identifies the cost of validating an XML input message using the validate node to do this mid flow and writing an unmodified XML output message.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1235.0	4.1	1.3	1
20kB	NONE	514.4	3.6	2.8	1
200kB	NONE	53.2	3.8	28.4	1
2000kB	NONE	4.9	3.9	314.0	1
2kB	FULL	486.5	3.6	2.9	1
20kB	FULL	173.8	3.2	7.4	1
200kB	FULL	24.2	3.0	50.0	1
2000kB	FULL	2.9	3.3	445.2	1

Validating an XML Message on Output

This test consists of:



The MQOutput node is set to validate the message contents and value. This causes the incoming message to be fully parsed and validated which is then written unmodified as the payload of the output message.

This test identifies the cost of validating an XML input message using the MQOutput node on output from the flow and writing an unmodified XML output message.

The results of running this test are given in the table below.

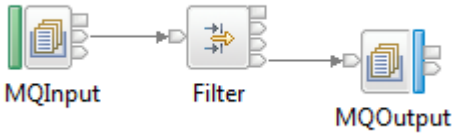
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1681.2	4.0	1.0	1
20kB	NONE	368.1	3.1	3.4	1
200kB	NONE	42.4	3.6	34.1	1
2000kB	NONE	3.0	3.5	474.0	1
2kB	FULL	252.6	3.2	5.1	1
20kB	FULL	117.8	3.2	10.9	1
200kB	FULL	20.7	3.1	59.8	1
2000kB	FULL	2.0	3.2	659.0	1

Opaque Parsing in the XMLNSC Domain

The test in this section illustrates the CPU processing costs of using the XMLNSC domain to opaquely parse an XML message. For an explanation of opaque parsing see the product documentation.

Filtering on the last element of an XML Message using Opaque Parsing on the XML Body

This test consists of:



The MQInput node is set to parse the repeating SalesList Elements of the XML message opaquely. Within the filter node the last element of the incoming message is examine. This last element is outside a sales list element. The result is always set to be true and thus the message is propagated to the MQOutput node.

This test identifies the cost of filtering on an element at the end of a message using the XMLNSC parser with opaque parsing. Almost the entire body of the message is opaquely parsed. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2512.4	4.6	0.7	1
20kB	NONE	1375.5	4.0	1.1	1
200kB	NONE	353.6	4.0	4.5	1
2000kB	NONE	17.2	4.2	97.1	1
20000kB	NONE	2.0	4.2	853.4	1
2kB	FULL	965.0	3.2	1.3	1
20kB	FULL	432.6	2.1	2.0	1
200kB	FULL	94.9	2.6	11.2	1
2000kB	FULL	10.5	2.6	100.7	1
20000kB	FULL	0.9	2.2	930.6	1

Routing and Transformation Logic

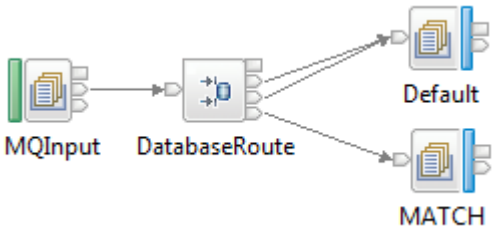
The tests in this section illustrate the processing cost of simple routing and transformation logic using a variety of routing and transformation technologies (ESQL, JavaCompute node, XML Transformation and PHPCompute). A number of the tests are performed for each of the technologies thus allowing a simple comparison of CPU processing costs to be made. In other cases a comparison is only made within a technology such as looking at the efficiency of different parsers whilst using ESQL. These tests are not a definitive statement of the relative processing costs of the different technologies. They are provided for illustrative purposes only. Message processing performance will be affected by the complexity of the messages and processing to be performed on the messages.

Using Database Route and Route Nodes

The tests in this section illustrate the processing costs of using the new Database Route and Route Nodes for routing operations.

Using Database Route Node to Route an Incoming Message Based on Data in a Database

This test consists of:



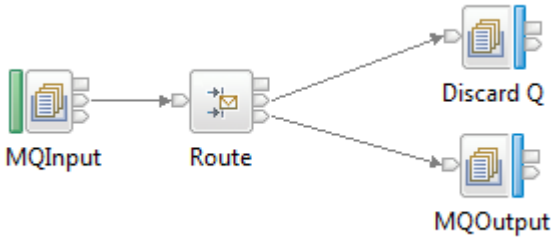
The MQ Input node is set to use the XMLNSC domain. Within the Database Route node a query is performed to obtain a single piece of data from the Database. This data is used to route the message to an output queue. The lookup result is always set to be true and thus the message is propagated to the MQ Output node.

This test identifies the cost of using the Database Route Node to route a message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1135.7	2.6	0.9	1
20kB	NONE	889.9	2.0	0.9	1
200kB	NONE	737.4	2.8	1.5	1
2000kB	NONE	68.8	5.6	32.3	1
20000kB	NONE	6.2	4.7	303.4	1
2kB	FULL	543.1	2.6	1.9	1
20kB	FULL	426.9	2.8	2.6	1
200kB	FULL	108.6	1.5	5.5	1
2000kB	FULL	12.7	1.5	48.7	1
20000kB	FULL	1.2	1.5	491.8	1

Using Route Node to Route an Incoming Message Based on Data in the Incoming Message

This test consists of:



The MQ Input node is set to use the XMLNSC domain. Within the Route node the first element of the message is queried and the message routed based on this value. The lookup result is always set to be true and thus the message is propagated to the MQ Output node.

This test identifies the cost of using the Route Node to route a message on an element at the start of a message using the XMLNSC parser. The results of running this test are given in the table below.

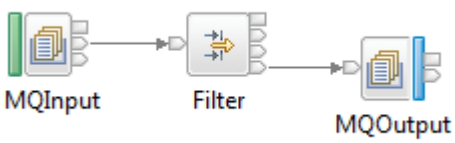
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2210.4	4.7	0.8	1
20kB	NONE	2041.3	4.8	0.9	1
200kB	NONE	1305.3	7.2	2.2	1
2000kB	NONE	108.7	8.7	32.0	1
20000kB	NONE	3.8	0.3	26.5	1
2kB	FULL	965.3	3.3	1.4	1
20kB	FULL	586.8	2.2	1.5	1
200kB	FULL	122.3	1.5	4.9	1
2000kB	FULL	13.7	1.5	43.6	1
20000kB	FULL	1.2	1.3	421.8	1

Using ESQL

The tests in this section illustrate the processing costs of using ESQL for different routing and transformation operations.

Filter an Incoming Message Based on the First Element in the Message using the XMLNSC Parser

This test consists of:



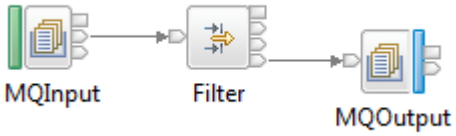
Within the filter node the first element of the incoming message is examined. The result is always set to be true and thus the message is propagated to the MQ Output node.

This test identifies the cost of filtering on an element at the start of a message using the XMLNSC parser. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2956.0	4.8	0.7	1
20kB	NONE	2542.9	5.0	0.8	1
200kB	NONE	1470.5	7.6	2.1	1
2000kB	NONE	112.2	8.8	31.5	1
20000kB	NONE	5.0	3.6	292.0	1
2kB	FULL	743.5	1.8	0.9	1
20kB	FULL	577.6	1.8	1.3	1
200kB	FULL	123.9	1.4	4.7	1
2000kB	FULL	13.5	1.4	40.8	1
20000kB	FULL	1.2	1.3	422.5	1

Filter an Incoming Message Based on the Last Element in the Message using the XMLNSC Parser

This test consists of:



Within the filter node the last element of the incoming message is examined. The result is always set to be true and thus the message is propagated to the MQ Output node.

This test identifies the cost of filtering on an element at the end of a message using the XMLNSC parser. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2133.9	4.5	0.8	1
20kB	NONE	946.2	3.5	1.5	1
200kB	NONE	109.8	3.4	12.2	1
2000kB	NONE	5.8	3.5	240.5	1
20000kB	NONE	0.9	3.5	1617.7	1
2kB	FULL	508.2	3.5	2.8	1
20kB	FULL	233.2	3.2	5.5	1
200kB	FULL	41.1	3.1	30.1	1
2000kB	FULL	4.8	3.0	249.7	1
20000kB	FULL	0.6	2.7	1763.8	1

Transformation of an Input Message using the XMLNSC Parser

This test consists of:

Within the compute node ESQL is written to significantly change the structure of the incoming message. The new structure is written as the output message

This identifies the cost of using ESQL to perform message transformation and message parsing using the XMLNSC parser. The results of running this test are given in the table below.

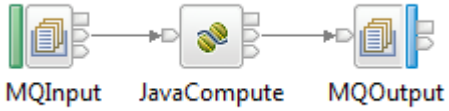
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1002.5	2.0	0.8	1
20kB	NONE	261.9	3.2	4.9	1
200kB	NONE	29.0	3.2	44.4	1
2000kB	NONE	2.6	3.3	509.3	1
20000kB	NONE	0.3	3.5	5324.0	1
2kB	FULL	296.8	3.7	5.0	1
20kB	FULL	155.3	3.3	8.4	1
200kB	FULL	17.5	3.2	72.2	1
2000kB	FULL	1.7	3.1	737.6	1
20000kB	FULL	0.2	3.1	5724.1	1

Using Java

The tests in this section illustrate the processing costs of using the JavaCompute node for different routing and transformation operations.

Filter an Incoming Message Based on the First Element in the Message using the Java Compute Nodes XPath Capability

This test consists of:



Within the Java Compute Node the first element of the incoming message is examined using the XPath capability. The result is always set to be true and thus the message is propagated to the MQ Output node.

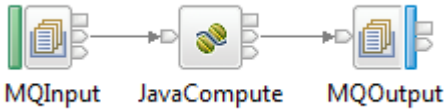
This test identifies the cost of filtering on an element at the start of a message using the Java Compute Node XPath capability. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2157.1	4.6	0.9	1
20kB	NONE	1754.3	4.2	1.0	1
200kB	NONE	1228.3	5.4	1.8	1

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2000kB	NONE	73.0	6.5	35.7	1
20000kB	NONE	6.2	5.5	353.5	1
2kB	FULL	765.4	2.6	1.4	1
20kB	FULL	544.1	2.4	1.7	1
200kB	FULL	118.6	1.6	5.4	1
2000kB	FULL	13.4	1.6	47.4	1
20000kB	FULL	1.3	1.5	474.2	1

Filter an Incoming Message Based on the Last Element in the Message using the Java Compute Nodes XPath Capability

This test consists of:



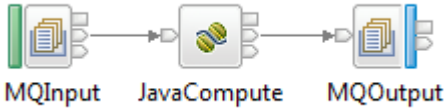
Within the Java Compute Node the last element of the incoming message is examined using the XPath capability. The result is always set to be true and thus the message is propagated to the MQOutput node

This test identifies the cost of filtering on an element at the end of a message using the Java Compute Node XPath capability. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1622.2	4.3	1.1	1
20kB	NONE	876.3	3.7	1.7	1
200kB	NONE	112.8	3.4	12.2	1
2000kB	NONE	5.5	3.5	256.1	1
20000kB	NONE	0.8	3.3	1672.3	1
2kB	FULL	450.9	3.0	2.7	1
20kB	FULL	212.6	3.2	6.0	1
200kB	FULL	40.3	3.1	30.6	1
2000kB	FULL	4.7	3.0	260.9	1
20000kB	FULL	0.6	2.9	2013.1	1

Filter an Incoming Message Based on the First Element in the Message using the Java Compute Nodes GetByPath Capability

This test consists of:



Within the Java Compute Node the first element of the incoming

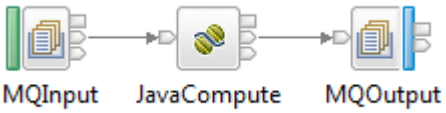
message is examined using the GetByPath capability. The result is always set to be true and thus the message is propagated to the MQOutput node.

This test identifies the cost of filtering on an element at the start of a message using the Java Compute Node GetByPath capability. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	2310.5	4.3	0.7	1
20kB	NONE	1861.7	4.4	0.9	1
200kB	NONE	1284.3	5.5	1.7	1
2000kB	NONE	97.4	7.3	30.0	1
20000kB	NONE	4.9	4.0	327.2	1
2kB	FULL	788.2	2.2	1.1	1
20kB	FULL	609.7	2.3	1.5	1
200kB	FULL	125.5	1.5	4.8	1
2000kB	FULL	13.7	1.5	43.0	1
20000kB	FULL	1.3	1.4	426.0	1

Filter an Incoming Message Based on the Last Element in the Message using the Java Compute Nodes GetByPath Capability

This test consists of:



Within the Java Compute Node the last element of the incoming message is examined using the GetByPath capability. The result is always set to be true and thus the message is propagated to the MQOutput node.

This test identifies the cost of filtering on an element at the end of a message using the Java Compute Node GetByPath capability. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1651.3	4.6	1.1	1
20kB	NONE	853.3	3.4	1.6	1
200kB	NONE	102.0	3.4	13.3	1
2000kB	NONE	8.3	3.6	172.5	1
20000kB	NONE	0.9	3.6	1689.9	1
2kB	FULL	518.2	3.8	2.9	1
20kB	FULL	227.0	3.2	5.6	1
200kB	FULL	39.5	3.1	31.0	1
2000kB	FULL	4.6	3.0	262.9	1
20000kB	FULL	0.6	2.9	1917.0	1

Parsing an XML Input Message

This test consists of:

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition a variable is declared and set to the last element in the incoming message. This causes the incoming message to be fully parsed. The output message consists of a message header only and no payload.

This test identifies the cost of parsing an XML input message. Because there is no message body on the output message there are no writing costs. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1523.9	4.0	1.1	1
20kB	NONE	961.7	3.6	1.5	1
200kB	NONE	108.8	3.3	12.2	1
2000kB	NONE	10.8	3.4	126.6	1
20000kB	NONE	1.0	3.4	1359.4	1
2kB	FULL	520.4	3.7	2.9	1
20kB	FULL	280.3	3.5	4.9	1
200kB	FULL	93.9	3.4	14.5	1
2000kB	FULL	9.8	3.5	141.8	1
20000kB	FULL	0.9	3.3	1416.9	1

Transformation of an Input Message using the Java Compute Nodes GetByPath Capability

This test consists of:

Within the Java Compute Node Java code, utilising the GetByPath capability is used to significantly change the structure of the incoming message. The new structure is written as the output message.

This test identifies the cost of using Java code and GetByPath to perform message transformation. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1258.0	4.0	1.3	1
20kB	NONE	265.5	2.9	4.3	1
200kB	NONE	29.8	3.4	45.7	1
2000kB	NONE	2.5	3.4	548.6	1
20000kB	NONE	0.3	3.2	4990.1	1

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	FULL	213.9	3.3	6.1	1
20kB	FULL	123.1	3.4	10.9	1
200kB	FULL	19.5	3.3	67.1	1
2000kB	FULL	1.9	3.2	687.4	1
20000kB	FULL	0.2	3.2	5841.4	1

Calling an External Java Procedure with One Integer Input Parameter

This test consists of:

Within the Compute node the message headers from the incoming message are copied to the outgoing message. Two thousand identical calls are made to an external Java procedure. The procedure receives one Integer parameter, and passes back zero parameters, and returns immediately.

This test identifies the cost of calling a Java procedure with one Integer parameter.

The results of running this test are given in the table below. The CPU ms/message figure has been adjusted to report a per procedure invocation cost by dividing the CPU cost obtained from the test by 2000. The results of running this test are given in the table below.

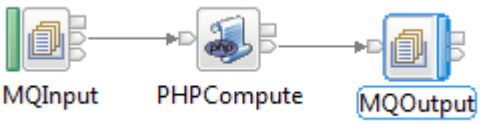
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1905.9	3.0	0.6	1
2kB	FULL	710.5	1.9	1.1	1

Using PHP

The tests in this section illustrate the processing costs of using a PHPCompute node to perform computation and transformation of an input message.

Copying a message using the PHPCompute node

This test consists of:



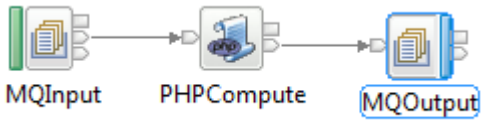
The PHPCompute node uses the MessageBrokerCopyTransform annotation which instructs the PHPCompute node to copy the input message and pass the copied message to the evaluate method. No further modifications are made to the output message.

This test identifies the cost of using the PHPCompute node scripting capability. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	916.3	4.2	1.8	1
20kB	NONE	856.0	4.1	1.9	1
200kB	NONE	696.0	5.5	3.2	1
2000kB	NONE	88.6	7.3	32.8	1
20000kB	NONE	7.0	5.9	337.9	1
2kB	FULL	695.1	4.1	2.3	1
20kB	FULL	483.7	3.2	2.7	1
200kB	FULL	121.5	1.6	5.1	1
2000kB	FULL	13.4	1.5	44.9	1
20000kB	FULL	1.3	1.4	430.9	1

Transformation of an Output Message using the PHPCompute node

This test consists of:



The PHPCompute node uses the MessageBrokerSimpleTransform annotation to significantly change the structure of the incoming message. The new structure is written as the output message.

This test identifies the cost of using PHPCompute node transformation (scripting) capability. The results of running this test are given in the table below.

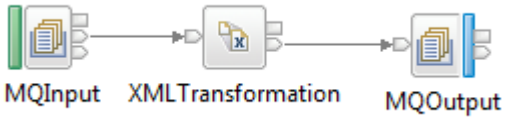
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	267.3	4.0	6.0	1
20kB	NONE	44.1	4.3	39.2	1
200kB	NONE	4.3	4.3	397.3	1
2000kB	NONE	0.3	4.3	5092.2	1
2kB	FULL	183.1	2.5	5.4	1
20kB	FULL	33.0	2.0	23.7	1
200kB	FULL	3.9	2.1	212.2	1
2000kB	FULL	0.3	2.4	2846.3	1

Using XSLT

The tests in this section illustrate the processing costs of using an XML Transformation node to perform a computation and transformation of an input message.

Transformation of an Input Message

This test consists of:



Within the XMLT Node a compiled stylesheet is used to significantly change the structure of the incoming message. The new structure is written as the output message.

This test identifies the cost of using an XSL stylesheet to perform message transformation. The results of running this test are given in the table below.

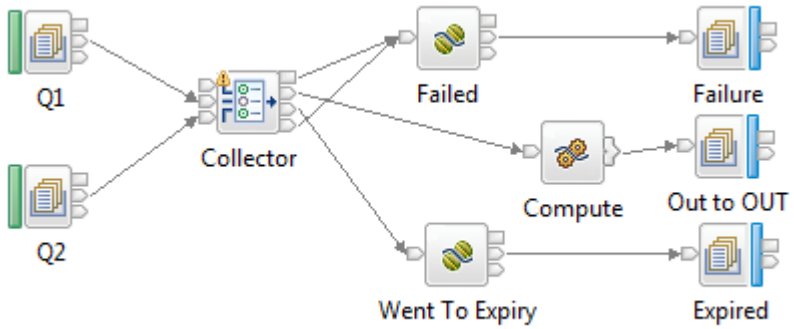
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1165.2	4.4	1.5	1
20kB	NONE	297.1	4.7	6.3	1
200kB	NONE	31.8	4.9	61.4	1
2000kB	NONE	2.9	4.6	630.1	1
2kB	FULL	561.1	3.9	2.8	1
20kB	FULL	141.2	4.1	11.7	1
200kB	FULL	17.8	4.3	96.5	1
2000kB	FULL	2.0	4.3	866.2	1

Using the Collector Node

The tests in this section illustrate the processing costs of using the Collector node for combining incoming messages. To allow for comparisons between collector tests the compute node used for all tests in this section is the same i.e. processing costs of this part in the flow is the same in all tests.

Collecting Messages from Several Inputs Based on Number of Messages

This test consists of:



The two MQ Input nodes each propagate messages to the collector node. In the collector node a collection is defined as being 1 input message from each of the two input terminals. The Collector node Persistence mode is set to Non-Persistent. Which means that the messages are

stored on the Collector node's queues Non-Persistently. Once this collection is satisfied it is propagated to the Java Compute Node which copies the entire message from the first input terminal to the output message. Then one field from the message received on the second terminal is retrieved from the input message and used to add a new field to the out going message. The message is then sent to an MQ Output node.

This test identifies the cost of using the collector node to collect 2 messages Non-Persistently from different inputs and then update one of them with a field from the other message.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	584.3	6.2	4.2	1
20kB	NONE	226.9	4.4	7.8	1
200kB	NONE	38.5	0.6	6.5	1
2000kB	NONE	3.2	1.9	242.5	1
20000kB	NONE	0.5	4.3	3407.1	1

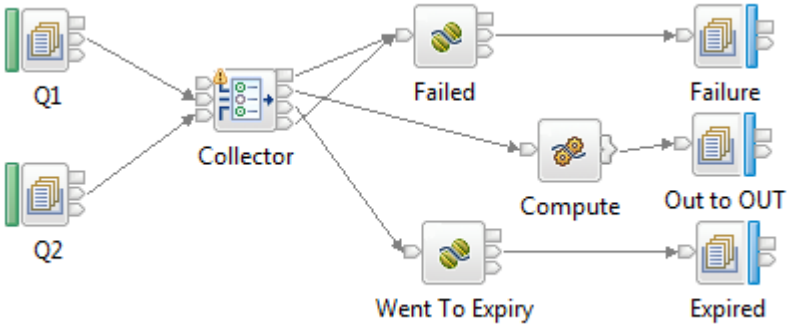
The results in the table below show the cost of running with the Collector node Persistence mode set to Persistent and using persistent transacted MQ messages to drive the flow. Running with a Collector node persistence mode set to persistent means that the messages are stored on the Collector node's queues as MQ persistent messages.

This test identifies the cost of using the collector node to collect 2 messages persistently from different inputs and then update one of them with a field from the other message. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	FULL	512.9	5.8	4.5	1
20kB	FULL	305.2	5.3	7.0	1
200kB	FULL	64.3	3.8	23.9	1
2000kB	FULL	5.0	4.4	359.0	1
20000kB	FULL	0.5	4.6	3395.3	1

Collecting Messages from Several Inputs Based on Number of Messages with a Correlation Pattern

This test consists of:



The two MQ Input nodes all propagate messages to the collector node. In

the collector node a collection is defined as being one input message from each of the two input terminals and also a correlation path to look at the first customer surname in the message. Hence messages with the same customer name are put in the collection. The Collector node Persistence mode is set to Non-Persistent. This means that the messages are stored on the Collector node's queues Non-Persistently. Once the collection is satisfied it is propagated to the Java Compute Node which copies the entire message from the first input terminal to the output message. Then one field from the message received on the second terminal is retrieved from the input message and used to add a new field to the out going message. The message is then sent to an MQ Output node. All input messages had the same matching name.

This test identifies the cost of using the collector node to collect 2 messages Non-Persistently from different inputs which have a matching surname and then update one of them with a field from the other message.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	597.4	5.3	3.6	1
20kB	NONE	270.1	4.3	6.4	1
200kB	NONE	66.7	3.6	21.6	1
2000kB	NONE	3.5	3.9	448.5	1
20000kB	NONE	0.5	3.8	3408.4	1

The results in the table below show the cost of running with the Collector node Persistence mode set to Persistent and using persistent transacted MQ messages to drive the flow.

Running with a Collector node persistence mode set to persistent means that the messages are stored on the Collector node's queues as MQ persistent messages.

This test identifies the cost of using the collector node to collect 2 messages Non-Persistently from different inputs which have a matching surname and then update one of them with a field from the other message. The results of running this test are given in the table below.

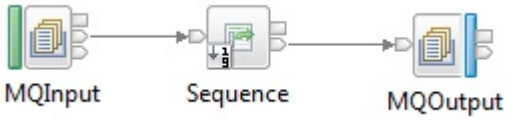
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	FULL	515.7	5.4	4.2	1
20kB	FULL	217.2	5.3	9.7	1
200kB	FULL	64.5	4.4	27.6	1
2000kB	FULL	4.7	4.5	378.6	1
20000kB	FULL	0.5	0.5	405.8	1

Using the Sequence Node

The tests in this section illustrate the processing costs of using the Sequence node to generate a sequence number.

Incrementing Sequence numbers for each Input Message

This test consists of:



The Sequence node is used to increment the sequence number for each new message (which are all part of the same sequence group) and to store the sequence number in the LocalEnvironment ($OutputLocalEnvironment/Sequence/Number) using a literal start and end of sequence definition.

This test identifies the cost of using the Sequence node to generate sequence numbers. The results of running this test are given in the table below.

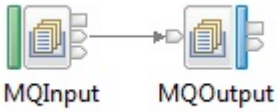
Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1653.7	4.3	1.1	1
20kB	NONE	1134.7	5.5	1.9	1
200kB	NONE	1059.5	7.3	2.8	1
2000kB	NONE	71.0	7.3	41.3	1
20000kB	NONE	5.8	7.0	482.6	1
2kB	FULL	751.8	2.9	1.5	1
20kB	FULL	462.4	3.0	2.6	1
200kB	FULL	116.8	2.5	8.7	1
2000kB	FULL	13.3	2.9	88.3	1
20000kB	FULL	1.2	1.9	639.7	1

Business-Level monitoring

The tests in this section illustrate the processing costs associated with using business-level monitoring. The message flow is configured to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Emitting one event with header information on Transaction start

This test consists of:



The MQInput node is configured to emit an event on Transaction start where the message contains information about the source of the event, the time of the event, and the reason for the event. The event does not include the message bit stream in the event payload.

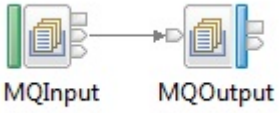
This test identifies the cost of emitting a single event with only the event header information.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1556.0	4.2	1.1	1
20kB	NONE	1462.1	4.3	1.2	1
200kB	NONE	1097.4	6.6	2.4	1
2000kB	NONE	103.0	7.2	27.9	1
20000kB	NONE	4.3	3.4	318.5	1
2kB	FULL	845.3	3.5	1.7	1
20kB	FULL	594.8	2.8	1.9	1
200kB	FULL	125.0	1.6	5.1	1
2000kB	FULL	13.8	1.4	39.2	1
20000kB	FULL	1.3	1.3	401.9	1

Emitting one event with a single selected element

This test consists of:



The MQInput node is configured to emit an event on Transaction start where the message contains information about the source of the event, the time of the event, and the reason for the event. In addition the event emitted also uses an XPath query take one field from the message tree and add it to the event pay load.

The XPath query used is the expression ($Body/Parent [1]/First[1]) thereby ensuring that a full parse of the message is not driven.

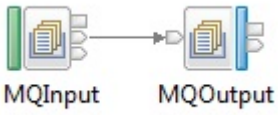
This test identifies the cost of emitting a single event with a simple XPath expression.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1309.6	4.0	1.2	1
20kB	NONE	1233.0	4.2	1.4	1
200kB	NONE	851.3	5.9	2.8	1
2000kB	NONE	104.5	8.6	32.8	1
20000kB	NONE	5.4	0.3	20.2	1
2kB	FULL	591.7	2.3	1.5	1
20kB	FULL	424.7	2.0	1.9	1
200kB	FULL	122.2	1.6	5.1	1
2000kB	FULL	13.6	1.3	39.4	1
20000kB	FULL	1.3	1.3	410.5	1

Emitting one event plus the full message tree

This test consists of:



The MQInput node is configured to emit an event on Transaction start where the message contains information about the source of the event, the time of the event, and the reason for the event. In addition the event emitted also uses an XPath query to select all the fields from the message tree and add them to the event payload.

The XPath query uses an expression ($Body/Parent) whereby all the fields in the message tree are parsed.

This test identifies the cost of emitting a single event when a message is fully parsed.

The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	907.5	3.8	1.7	1
20kB	NONE	512.4	3.8	3.0	1
200kB	NONE	53.4	4.0	29.9	1
2000kB	NONE	5.7	4.6	322.4	1
20000kB	NONE	0.4	3.8	4001.0	1
2kB	FULL	427.3	3.7	3.5	1
20kB	FULL	192.1	3.8	8.0	1

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
200kB	FULL	27.5	3.5	50.5	1
2000kB	FULL	2.9	3.8	521.9	1
20000kB	FULL	0.3	3.5	4045.5	1

External resources

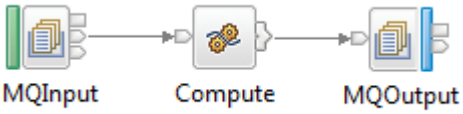
The tests in this section illustrate the processing cost of accessing resources such as a database or an external procedure.

Accessing a Database from a Message Flow

The tests in this section illustrate the processing cost of performing operations on a DB2 database.

Reading from a Database

This test consists of:



The input and output message are processed with the XMLNSC domain.

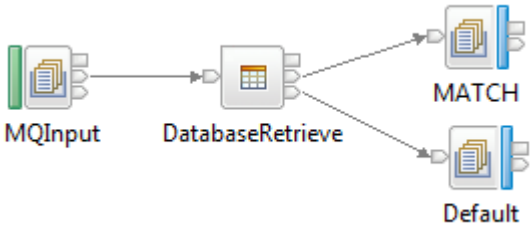
Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition a SELECT operation is performed to obtain a piece of data from the Database. This data is used to validate an element in the input message. The results are not cached in the flow, so a lookup is performed for each message. The volume of data in the database was small and so this represents the best case.

This test identifies the cost of performing a Database SELECT operation. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1016.2	3.4	1.3	1
20kB	NONE	1004.9	3.4	1.4	1
200kB	NONE	905.2	4.0	1.8	1
2000kB	NONE	140.6	5.2	14.8	1
20000kB	NONE	14.6	5.5	150.5	1
2kB	FULL	733.2	3.9	2.1	1
20kB	FULL	582.0	3.4	2.3	1
200kB	FULL	174.5	1.7	4.0	1
2000kB	FULL	25.6	1.5	22.7	1
20000kB	FULL	2.5	1.6	247.1	1

Reading from a Database using the Database Retrieve Node

This test consists of:



The input and output message are processed with the XMLNSC domain.

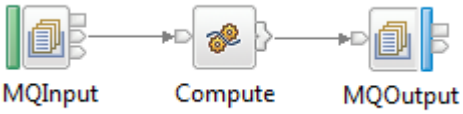
Within the DBRetrieve node the copy message box is selected so that the contents of the input message are copied to the outgoing message. The database is queried to obtain a piece of data, which is used to create an element in the output message.

This test identifies the cost of using the DatabaseRetrieve node to retrieve a piece of data from the database and insert it into the outgoing message. The test performs a similar function to reading from a database as in the case above but achieves it by using a different node and involves no programming. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	1047.5	2.2	0.8	1
20kB	NONE	964.7	2.1	0.9	1
200kB	NONE	767.4	3.0	1.5	1
2000kB	NONE	51.6	4.6	35.4	1
20000kB	NONE	4.6	0.5	40.8	1
2kB	FULL	574.4	2.9	2.0	1
20kB	FULL	436.9	2.6	2.4	1
200kB	FULL	109.4	1.6	5.7	1
2000kB	FULL	12.9	1.7	53.1	1
20000kB	FULL	1.3	1.7	523.8	1

Inserting into a Database

This test consists of:



The input and output message are processed with the XMLNSC domain.

Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition an INSERT operation is performed to populate the database with data that is obtained from an element in the input message.

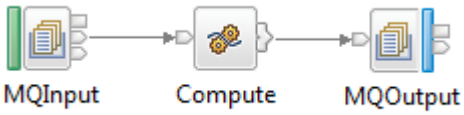
This test identifies the cost of performing a database INSERT operation. The results of running this test are given in the table below.

Msg	Persistent	Message Rate	% CPU	CPU	Instances

Size		(Msgs/Sec)	Busy	ms/msg	
2kB	NONE	849.8	2.9	1.4	1
20kB	NONE	832.6	3.0	1.4	1
200kB	NONE	766.7	3.6	1.9	1
2000kB	NONE	176.2	6.1	13.8	1
20000kB	NONE	4.3	0.2	17.3	1
2kB	FULL	533.1	2.7	2.0	1
20kB	FULL	467.1	2.8	2.4	1
200kB	FULL	183.0	1.9	4.2	1
2000kB	FULL	25.5	1.3	21.2	1
20000kB	FULL	2.5	1.5	236.1	1

Updating a row in a Database

This test consists of:



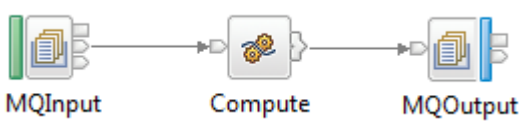
Within the Compute node the message headers from the incoming message are copied to the outgoing message. In addition an UPDATE operation is performed to update a piece of data in the database with a new value that is obtained from an element in the input message.

This test identifies the cost of performing a database UPDATE operation. The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	974.9	3.7	1.5	1
20kB	NONE	932.9	3.7	1.6	1
200kB	NONE	872.5	4.1	1.9	1
2000kB	NONE	153.4	5.7	15.0	1
20000kB	NONE	8.2	3.5	171.7	1
2kB	FULL	598.6	3.3	2.2	1
20kB	FULL	459.7	2.8	2.4	1
200kB	FULL	182.2	2.1	4.5	1
2000kB	FULL	25.8	1.5	23.9	1
20000kB	FULL	2.3	1.4	254.3	1

Calling an External Database Stored Procedure with One Integer Input Parameter

This test consists of:



Within the Compute node the message headers from the incoming message are copied to the outgoing message. Two thousand identical calls are made to an external database stored procedure. The procedure receives one parameter which is an integer, passes back zero parameters, and returns immediately.

This test identifies the cost of calling a database stored procedure with one parameter which is an integer.

The results of running this test are given in the table below. The CPU ms/message figure has been adjusted to report a per procedure invocation cost by dividing the CPU cost obtained from the test by 2000.
The results of running this test are given in the table below.

Msg Size	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg	Instances
2kB	NONE	3.9	3.1	317.2	1
2kB	FULL	3.9	3.1	315.6	1

Growing message throughput

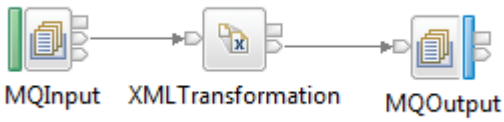
The tests in this section show the effect of using two different approaches to increase message throughput for a message flow. These are the use of additional instances and assigning one copy of the message flow to each of multiple execution groups. The increase in throughput will be limited by the number of CPUs on the machine, and whether other factors such as I/O or locking are present in the application. It is usual to see message rates start to plateau before the system utilisation reaches 100%.

Using Additional Instances

WebSphere Message Broker allows the use of additional instances of a message flow to be run. These instances map onto threads running within a broker execution group process on distributed platforms and to Task Control Blocks (TCBs) within the brokers address space on z/OS system.

A series of tests were run to show how message throughput can be increased through the use of additional instances. The XSLT Transform Sample was run with a varying number of instances of the message flow in a single execution group.

This test consists of:



The benefits observed from running multiple copies of a message flow in any given situation depends on the processing requirements of the message flow. CPU bound message flows have different scaling characteristics from those which are I/O-bound for example.
The message size used in all cases was 2kB.

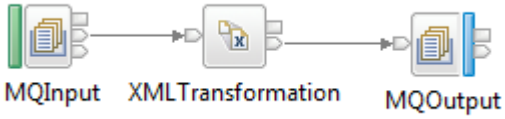
The results of running this test are given in the table below. They indicate the results of scaling with Instances with non-persistent messages.

Number of Instances	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg
1	NONE	1054.7	1.9	0.7
2	NONE	1863.9	10.1	2.2
3	NONE	3087.5	7.3	0.9
4	NONE	3350.5	22.4	2.7
5	NONE	2965.6	22.5	3.0
6	NONE	4025.8	30.0	3.0

Determine the optimum number of instances to use for each message flow individually by experimenting with a varying number of instances. These measurements show that additional instances can be an effective mechanism for increasing message throughput and allowing more of a machine to be fully utilised. In both of the tests CPU usage and message rate were grew significantly over the initial position.

Using Multiple Execution Groups

WebSphere Message Broker allows a message flow to be assigned to more than one execution group at a time and for those multiple copies to process messages concurrently. An execution group maps on-to an operating system processes on distributed platforms and to an address space on z/OS systems.
A series of tests were run to show how message throughput can be increased by using multiple execution groups. The XSLT Transform Sample was run with a varying number of execution groups each running one copy of the message flow.
This test consists of:



The benefits observed from running multiple copies of a message flow in any given situation depending on the processing requirements of the message flow. CPU bound message flows will have different scaling characteristics from those which are I/O-bound, for example.
The message size used in all cases was 2kB.
The results of running this test are given in the table below. They indicate the results of scaling using non-persistent messages.

Number of Execution Groups	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg
1	NONE	773.5	2.1	1.1
2	NONE	1547.1	9.9	2.6
3	NONE	3247.4	16.1	2.0
4	NONE	4211.3	24.3	2.3
5	NONE	5295.1	30.4	2.3
6	NONE	6039.4	37.5	2.5
7	NONE	7070.3	42.3	2.4
8	NONE	7668.4	52.1	2.7
9	NONE	8119.3	57.7	2.8
10	NONE	8836.7	60.5	2.7
11	NONE	9241.0	63.6	2.8
12	NONE	9771.5	69.0	2.8
13	NONE	10238.1	74.0	2.9
14	NONE	11321.1	76.8	2.7
15	NONE	11601.9	79.4	2.7
16	NONE	12341.1	82.1	2.7
17	NONE	12892.5	86.6	2.7
18	NONE	13147.7	87.7	2.7
19	NONE	13372.0	89.8	2.7
20	NONE	13779.7	92.0	2.7
21	NONE	13737.9	93.4	2.7
22	NONE	14305.0	94.8	2.6

Number of Execution Groups	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg
23	NONE	14569.7	95.8	2.6

To illustrate the different scaling behaviour that might be seen with non-CPU-bound flows the same test was run using persistent messages. For this test the disk becomes the overriding factor and so scaling of the CPU and message throughput is potentially more limited.

Number of Execution Groups	Persistent	Message Rate (Msgs/Sec)	% CPU Busy	CPU ms/msg
1	FULL	540.2	2.8	2.1
2	FULL	783.5	6.0	3.0
3	FULL	647.3	3.0	1.8
4	FULL	918.4	5.1	2.2
5	FULL	1122.8	6.6	2.4
6	FULL	1469.2	8.9	2.4
7	FULL	1724.6	12.6	2.9
8	FULL	1955.2	15.9	3.2
9	FULL	2208.1	18.3	3.3
10	FULL	2264.7	17.9	3.1
11	FULL	2979.2	26.0	3.5
12	FULL	3293.1	31.0	3.8
13	FULL	3800.1	37.3	3.9
14	FULL	3972.1	38.5	3.9
15	FULL	4048.3	39.1	3.9
16	FULL	4432.0	42.9	3.9
17	FULL	4597.4	43.0	3.7

With persistent messages it is again possible to increase message throughput over the rate achieved with one copy by using more copies of the message flow.
Note that the initial rate for this test is lower than the initial rate for the non-persistent messages. This pattern continues when comparing the message throughput with a given number of instances. This is the impact of the log I/O that is needed for persistent messages.
Determine the optimum number of execution groups to use by experimenting with a varying number.
These measurements show that multiple execution groups can be an effective mechanism for increasing message throughput and allowing more of a machine to be fully utilised. In both of the tests CPU usage and message rate were grew significantly over the initial position.

Resource Requirements

This section illustrates memory use for message flows.

Recommended Minimum Specification

The minimum specification required to install and run the development toolkit and the runtime can be found in the Installation Guide which can be downloaded here: http://www.ibm.com/software/integration/wbimessagebroker/requirements/#V80
These are the minimum requirements to enable the processing of simple messages with simple message transformation or routing. Situations requiring more intensive processing are likely to need greater resources.

Memory Use

The amount of memory used by a message flow running within an execution group varies depend on the complexity of the message flow, the style of processing within the message flow, and the size of the messages being processed. This is a complex subject and a detailed discussion is beyond the scope of this document. However, to assist with planning, the memory used for a variety of tests is reported.

Real memory is the amount of physical RAM allocated for the process. Memory utilisation is reported to the nearest 1MB using perfmon for Windows.

Note that the recorded memory size depends on the platform-specific memory and swap space allocation algorithms. These values vary according to platform.

The figures in the table below record the amount of real memory in MB used by an execution group for the message flow after it has processed a number of messages and the size has stabilised.

In each case a single copy of the message flow was deployed to a single execution group.

Each use case was deployed to a new execution group.

Use Case	Message Size	Real Memory Peak After Processing Messages (MB)
Aggregation	20kB	159
Coordinated Request/Reply	2kB	157
Data Warehouse	2kB	154
Large Messaging	2kB	152
Message Routing	2kB	155
XML Transformation	2kB	150

Real Memory Use for a Variety of Use Cases.

Appendix A - Measurement Environment

All throughput measurements where taken on a single server machine. The client type and machine on which they ran varied with the test. The details are given below.

Server Machine

The hardware consisted of:

- IBM xSeries x3690 X5 with 2 x Deca-Core Intel Xeon E7-2860 2.27GHz processors with HyperThreading turned on
- 146GB 15K 6.0Gbps SFF Serial SCSI / SAS Hard Drive - ST9146852SS
- 50GB SATA 3.5-INCH HOT-SWAP HIGH IOPS SOLID STATE DRIVE - 43W7701
- 16 GB RAM
- 10 GB Ethernet Card

The software consisted of:

- Microsoft Windows 2008 R2
- WebSphere MQ V7.0.1.5
- WebSphere Message Broker V8
- DB2 V9.7

Client Machine

The hardware consisted of:

- IBM xSeries x3650 M3 with 2 x Hex-Core Xeon(TM) 2.80 GHz processors
- One 135 GB SCSI hard drive formatted with NTFS
- 16 GB RAM
- 10 GB Ethernet card

The software consisted of:

- Microsoft Windows 2008 R2
- WebSphere MQ V7.0.1.0
- IBM Java v6

Network Configuration

The client and server machines were connected using a full duplex 10 Gigabit Ethernet LAN with a single hub.

Appendix B - Evaluation Method

This section outlines the software components that were used to produce the measurements which are contained in this report.
Two different configurations were used to generate and consumpe input and output messages. This is because different test cases required different types of input and output messages. The methods used were:

1. Point to Point Message Processing. This configuration tested these transports:
    1. MQ
    2. JMS
    3. HTTP
    4. SOAP

These are described in the remainder of this section.
A series of parameter configuration changes were made to improve message throughput. These are discussed in the section Tuning.

Point to Point testing

This section describes how messages were generated and consumed for the point to point messaging tests, such as the Database Read tests or Filter an Incoming Message based on the First Element in the Message. The configuration of the software components is also discussed. This approach was used for MQ, JMS, HTTP and SOAP messages.

Message Generation and Consumption

The Performance Harness for JMS, a multi-threaded WebSphere MQ Client program written in Java was used to generate input messages for the test case being run and to consume the output messages. The following PerfHarness modules were used for point to point testing:

- mqjava.Requestor . for MQ Messages
- http.Requestor . for Sending SOAP and HTTP messages
- jms.r11.Requestor . for sending and receiving JMS Messages

Differences between the transport testing are detailed below:

MQ

Both persistent and non persistent MQ Messages were generated from this program. Persistent messages were sent as part of a transaction which was committed after every message.
Sufficient threads (typically 20) were run in the multi-threaded client to ensure that there were always messages on the input queue waiting to be processed. This is important when measuring message throughput. Each thread sent a message and then immediately went to receive a reply on the output queue. Any thread within the client program was able to retrieve any message which had been processed by a message flow. No use was made of the WebSphere MQ correlation identifiers to limit consumption of a message to the thread which created it. When a thread received a reply it sent another message. The message content was the same for all threads and all messages.

JMS

The tool sent non persistent JMS Bytes messages to a JMS Destination. The connection factory for the client used the MQ Client transport to

send messages. This destination was mapped to an MQ Queue on the WebSphere Message Brokers queue manager. The JMS Input node was configured to read from this queue, the connection factory for the nodes used the MQ Bindings transport for connection. The flow then placed the reply message on another MQJMS queue on output where the client could then receive the reply.
Sufficient threads (typically 20) were run in the multi threaded client to ensure that there were always messages on the input queue waiting to be processed. This is important when measuring message throughput. Each thread sent a message and then immediately went to receive a reply on the output queue. Any thread within the client program was able to retrieve any message which had been processed by a message flow. No use was made of the JMS correlation identifiers to limit consumption of a message to the thread which created it. When a thread received a reply it sent another message. The message content was the same for all threads and all messages.

SOAP and HTTP

The tool sends predefined SOAP and HTTP Messages that it reads from files. The tool sent the messages to a broker using persistent HTTP connections, which means that each thread reused the same TCPIP socket for each request. Each client thread had its own TCPIP socket connection to send and receive data.
Sufficient threads (typically 20) were run in the multi-threaded client to ensure that there were always messages to be processed. This is important when measuring message throughput.
As per the HTTP request/reply protocol each thread sent a message and then immediately went to receive a reply on socket. When a thread received a reply it sent another message. The message content was the same for all threads and all messages.

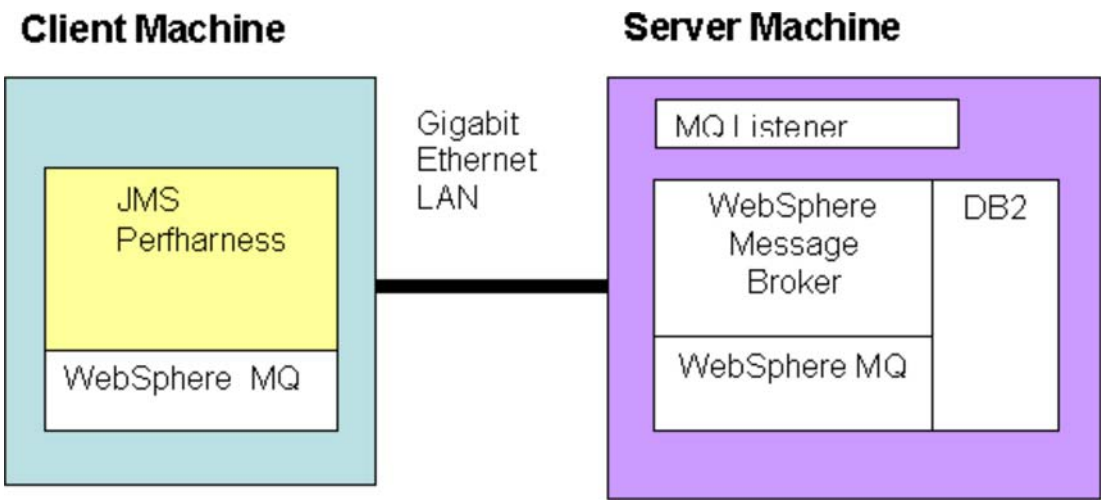
See the Performance Harness section in this report for more information on this tool.

Machine Configuration

The Performance Harness for JMS was used to generate and consume messages for the message flows and was run on a single client machine. WebSphere Message Broker, its dedicated WebSphere MQ queue manager and broker database were all located on the server machine.

There was a single client machine.

For MQ and JMS based Tests messages were transmitted from the client machine to the server machine over WebSphere MQ SVRCONN channels. The messages were received on the server by using WebSphere MQ queue manager listener process. This was run as a trusted MQ application in order to improve message throughput.
Messages were transmitted from the client machines to the server machine using the WebSphere MQ transport or SOAP/HTTP or JMS depending on the test type.
The diagram below illustrates the major components in the measurement environment and their location.



Both the client and server machine were configured with sufficient memory to ensure that no paging took place during the tests.

Reported Message Rates

For tests that did not involve publish/subscribe, the message rates reported are the number of invocations of the message flow per second. For tests involving several message flows such as the message aggregation test the rate reported is the number of complete operations or aggregations per second. Fan-out and fan-in processing is counted as one operation rather than separate operations.
For tests using the JMS nodes the message rate is the number of message flow invocations per second.
The message rates quoted are an average taken over the measurement period. This starts when the system initialisation period has completed.

Appendix C - Test Messages

This section describes the input and output messages used for the tests detailed in this report.

The messages which are in this section have been formatted for this report and as such contain white space between tags. When used in measurements all white space is removed.

Input Messages

This section details the types of input messages used in the report.

General Input Messages

An input message of the type shown below was used for the non-publish/subscribe tests in the report.

The message shown below is in generic XML format but it was also represented in a variety of other formats such as MRM XML, CWF and TDS where this was required in the test.

The different message sizes used in testing are achieved by repeating the content of the SaleList tag to give the required size. Larger messages thus result in more tags. A Perl script ensures that the names and values in the tags are different as the SaleList structure is repeated. This is to stop a limited number of strings being used in very large messages which could lead to over-optimistic results.

```
<Parent>
   <First>1</First>
   <SaleList>
      <Invoice>
         <Initial>K</Initial>
         <Initial>A</Initial>
         <Surname>Braithwaite</Surname>
         <Item>
            <Code>00</Code>
            <Code>01</Code>
            <Code>02</Code>
            <Description>Twister</Description>
            <Category>Games</Category>
            <Price>00.30</Price>
            <Quantity>01</Quantity>
         </Item>
         <Item>
            <Code>02</Code>
            <Code>03</Code>
            <Code>01</Code>
            <Description>The Times Newspaper</Description>
            <Category>Books and Media</Category>
            <Price>00.20</Price>
            <Quantity>01</Quantity>
         </Item>
         <Balance>00.50</Balance>
         <Currency>Sterling</Currency>
      </Invoice>
```

```
      <Invoice>
        <Initial>T</Initial>
        <Initial>J</Initial>
        <Surname>Dunnwin</Surname>
        <Item>
          <Code>04</Code>
          <Code>05</Code>
          <Code>01</Code>
          <Description>The Origin of Species</Description>
          <Category>Books and Media</Category>
          <Price>22.34</Price>
          <Quantity>02</Quantity>
        </Item>
        <Item>
          <Code>06</Code>
          <Code>07</Code>
          <Code>01</Code>
          <Description>Microscope</Description>
          <Category>Miscellaneous</Category>
          <Price>36.20</Price>
          <Quantity>01</Quantity>
        </Item>
        <Balance>81.84</Balance>
        <Currency>Euros</Currency>
      </Invoice>
   </SaleList>
   <Last>Test</Last>
</Parent>
```

SOAP Input Message and WSDL

Below is the input message and WSDL used for the SOAP Nodes tests:

```
<soapenv:Envelope
            xmlns:soapenv="http://schemas.xmlsoap.org
/soap/envelope/"
            xmlns:soapenc="http://schemas.xmlsoap.org
/soap/encoding/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:tns="http://WssSale.miwssoap.broker.mqst.ibm.com"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soapenv:Header>
    <wsa:Action>SummerSale</wsa:Action>
    <wsa:MessageID>uuid:515704D6-0111-4000-
E000-82267F000001</wsa:MessageID>
  </soapenv:Header>
  <soapenv:Body>
    <tns:SaleRequest>
      <SaleEnvelope>
        <Header>
          <SaleListCount>1</SaleListCount>
        </Header>
        <SaleList>
          <Invoice>
            <Initial>K</Initial>
            <Initial>A</Initial>
            <Surname>Braithwaite</Surname>
            <Item>
```

```
            <Code>00</Code>
            <Code>01</Code>
            <Code>02</Code>
            <Description>Twister</Description>
            <Category>Games</Category>
            <Price>00.30</Price>
            <Quantity>01</Quantity>
          </Item>
          <Item>
            <Code>02</Code>
            <Code>03</Code>
            <Code>01</Code>
            <Description>The Times Newspaper</Description>
            <Category>Books and Media</Category>
            <Price>00.20</Price>
            <Quantity>01</Quantity>
          </Item>
          <Balance>00.50</Balance>
          <Currency>Sterling</Currency>
        </Invoice>
        <Invoice>
          <Initial>T</Initial>
          <Initial>J</Initial>
          <Surname>Dunnwin</Surname>
          <Item>
            <Code>04</Code>
            <Code>05</Code>
            <Code>01</Code>
            <Description>The Origin of Species</Description>
            <Category>Books and Media</Category>
            <Price>22.34</Price>
            <Quantity>02</Quantity>
          </Item>
          <Item>
            <Code>06</Code>
            <Code>07</Code>
            <Code>01</Code>
            <Description>Microscope</Description>
            <Category>Miscellaneous</Category>
            <Price>36.20</Price>
            <Quantity>01</Quantity>
          </Item>
          <Balance>81.84</Balance>
          <Currency>Euros</Currency>
        </Invoice>
      </SaleList>
      <Trailer>
        <CompletionTime>12.00.00</CompletionTime>
      </Trailer>
    </SaleEnvelope>
  </tns:SaleRequest>
 </soapenv:Body>
</soapenv:Envelope>

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://WssSale.miwssoap.broker.mqst.ibm.com"
  xmlns:tns="http://WssSale.miwssoap.broker.mqst.ibm.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">
  <wsdl:types>
    <xsd:schema

targetNamespace="http://WssSale.miwssoap.broker.mqst.ibm.com"
            xmlns:tns="http://WssSale.miwssoap.broker.mqst.ibm.com"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="SaleRequest" type="tns:RootMessage"/>
      <xsd:element name="SaleResponse" type="tns:RootMessage"/>
      <xsd:complexType name="RootMessage">
        <xsd:sequence>
          <xsd:element name="SaleEnvelope">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Header" type="tns:Header"/>
                <xsd:element maxOccurs="unbounded"
name="SaleList" type="tns:SaleList"/>
                <xsd:element name="Trailer" type="tns:Trailer"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="SaleList">
        <xsd:sequence>
          <xsd:element maxOccurs="2" minOccurs="2" name="Invoice"
type="tns:Invoice"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Invoice">
        <xsd:sequence>
          <xsd:element maxOccurs="2" minOccurs="2" name="Initial"
type="xsd:string"/>
          <xsd:element name="Surname" type="xsd:string"/>
          <xsd:element maxOccurs="2" minOccurs="2" name="Item"
type="tns:Item"/>
          <xsd:element name="Balance" type="xsd:float"/>
          <xsd:element name="Currency" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Item">
        <xsd:sequence>
          <xsd:element maxOccurs="3" minOccurs="3" name="Code"
type="xsd:string"/>
          <xsd:element name="Description" type="xsd:string"/>
          <xsd:element name="Category" type="xsd:string"/>
          <xsd:element name="Price" type="xsd:float"/>
          <xsd:element name="Quantity" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Header">
        <xsd:sequence>
          <xsd:element name="SaleListCount" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="Trailer">
        <xsd:sequence>
          <xsd:element name="CompletionTime" type="xsd:string"/>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="SaleRequest">
    <wsdl:part element="tns:SaleRequest" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="SaleResponse">
    <wsdl:part element="tns:SaleResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="WssSale">
    <wsdl:operation name="Sale">
      <wsdl:input  message="tns:SaleRequest" name="SaleRequest"

wsaw:Action="http://WssSale.miwssoap.broker.mqst.ibm.com/WssSale
/services/WssSale/SaleRequest"/>
      <wsdl:output message="tns:SaleResponse" name="SaleResponse"

wsaw:Action="http://WssSale.miwssoap.broker.mqst.ibm.com/WssSale
/services/WssSale/SaleResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WssSaleSoapBinding" type="tns:WssSale">
    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Sale">
      <wsdlsoap:operation soapAction="SummerSale"/>
      <wsdl:input name="SaleRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="SaleResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="WssSaleService">
    <wsdl:port binding="tns:WssSaleSoapBinding" name="WssSale">
      <wsdlsoap:address location="http://localhost:9081/WssSale
/services/WssSale"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Output Message

For those tests that modified the message one of two message formats was used for the output messages depend on the test case. These are the Compute and Transformation messages.

Transformation Message

For the message transformation test the input message is modified and takes a different layout. For each invoice a statement is created for each customer within a SaleList.

The message layout is shown below.

```
<Parent>
    <SaleList>
```

```
    <Statement Type="Monthly" Style="Full">
      <Customer>
        <Initials>KA</Initials>
        <Name>Braithwaite</Name>
        <Balance>00.50</Balance>
      </Customer>
      <Purchases>
        <Article>
          <Desc>Twister</Desc>
          <Cost>4.8E-1</Cost>
          <Qty>01</Qty>
        </Article>
        <Article>
          <Desc>The Times Newspaper</Desc>
          <Cost>3.2E-1</Cost>
          <Qty>01</Qty>
        </Article>
      </Purchases>
      <Amount>8E-1</Amount>
    </Statement>
    <Statement Type="Monthly" Style="Full">
      <Customer>
        <Initials>TJ</Initials>
        <Name>Dunnwin</Name>
        <Balance>81.84</Balance>
      </Customer>
      <Purchases>
        <Article>
          <Desc>The Origin of Species</Desc>
          <Cost>3.5744E+1</Cost>
          <Qty>02</Qty>
        </Article>
        <Article>
          <Desc>Microscope</Desc>
          <Cost>5.792E+1</Cost>
          <Qty>01</Qty>
        </Article>
      </Purchases>
      <Amount>1.29408E+2</Amount>
    </Statement>
   </SaleList>
</Parent>
```

Appendix D - Use case descriptions

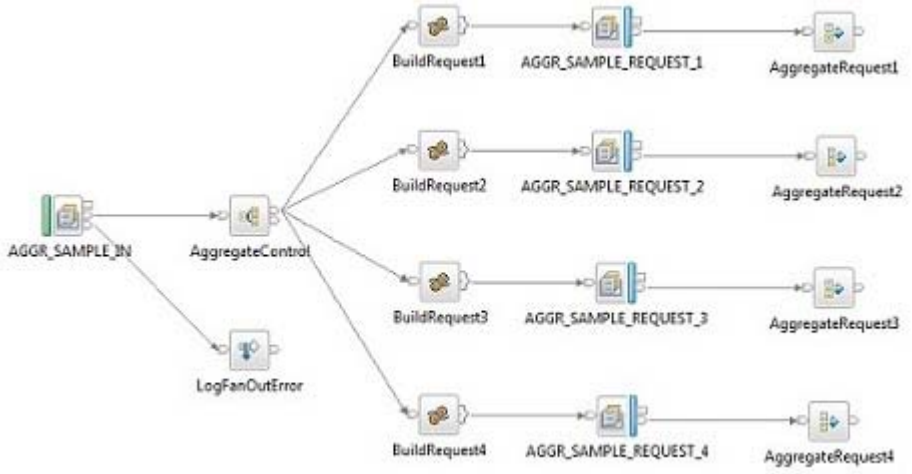
This section contains a description of the processing in each of the use cases which are used to characterise the performance of WebSphere Message Broker V8. All of these use cases are provided as samples in WebSphere Message Broker V8. See the samples gallery for more information.

Aggregation

The Aggregation use case demonstrates a simple four-way aggregation operation, using the AggregateControl, Request, and Reply nodes. It contains three message flows to implement a four-way aggregation: FanOut, RequestReplyApp, and FanIn. This is the type of processing that might be used to invoke four different applications to process a travel booking, one to organise each of the flight, hotel, car and currency.

FanOut Message Flow

This is the flow that takes the incoming request message, generates four different request messages, sends them out using a request/reply pattern, and starts the tracking of the aggregation operation:

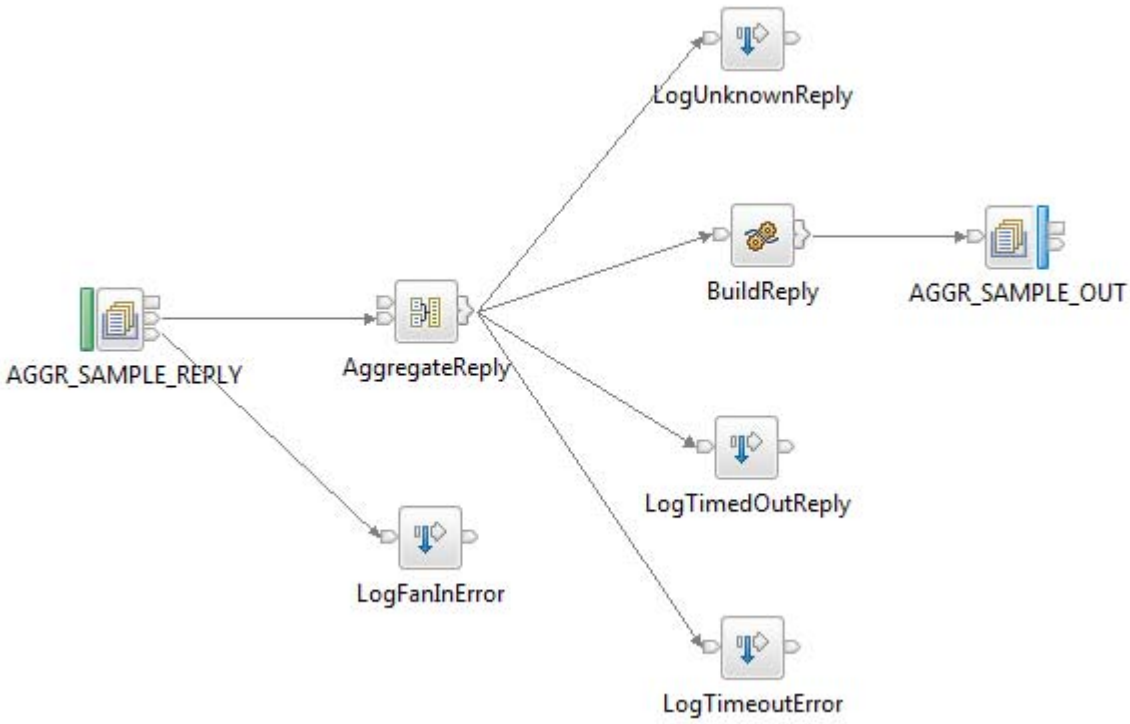


RequestReplyApp Message Flow

This message flow simulates the back-end service applications that would normally process the request messages from the aggregation operation. In a real system, these could be other message flows or existing applications. This message flow reads from the same queue that the MQOutput nodes in the FanOut flow write to, and it outputs to the queue that the input node which the FanIn flow reads from. It therefore provides a messaging bridge between the two flows. The messages are put to their reply-to queue (as set by the MQOutput nodes in the FanOut flow).

FanIn Message Flow

This flow receives all the replies from the RequestReplyApp flow, and aggregates them into a single output message. The output message from the AggregateReply node cannot be output directly by an MQOutput node without some processing so a Compute node is added to process the data into a format where it can be written to a queue.



Further information about the Aggregation sample can be found in the WebSphere Message Brokers section of the Technology samples category, which is in the samples gallery of the WebSphere Message Broker development toolkit.

Coordinated Request/Reply

The coordinated request reply sample is based on the scenario of a contemporary and established application communicating through the use of WebSphere MQ messages in a request/reply processing pattern. The contemporary application uses self-defining XML messages and issues a request message. The established application uses Custom Wire Format (CWF) messages.
The Sample receives a request message, processes it and delivers a reply message. For the applications to successfully communicate, the message formats must be transformed for both the request and reply messages.

The processing in the sample consists of three message flows and one message set. The message flows are:

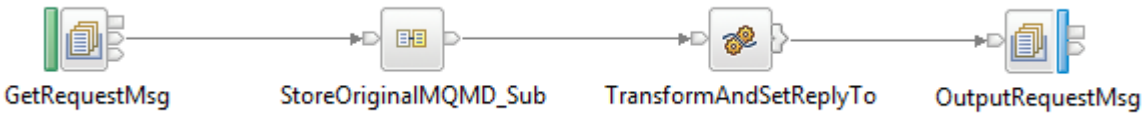
Request Message Flow

The request message flow performs the following processing:

- Reads a WebSphere MQ message containing an XML payload.
- Converts the message into the equivalent CWF format.
- Creates a WebSphere MQ message containing the transformed message.
- Saves the original ReplyToQ and ReplyToQMgr details in a separate

WebSphere MQ message for subsequent retrieval by the Reply message flow.
- Sets the ReplyToQ and ReplyToQMgr details to be the input of the Reply message flow.
- Sends the message to the Backend Reply message flow.

The Request message flow consists of the following nodes:

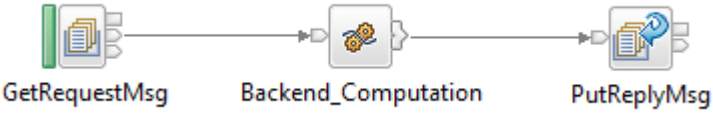


Backend Reply Message Flow

The backend reply message flows performs the following processing:

- Reads a WebSphere MQ message.
- Adds the time the message was modified to the payload of the message.
- Writes a WebSphere MQ message.

The Backend Reply message flow consists of the following nodes:



Reply Message Flow

The reply message flow performs the following processing:

- Reads a WebSphere MQ message containing a message in CWF format.
- Converts the message into the equivalent XML format.
- Obtains the ReplyToQ and ReplyToQ Mgr of the original request message by reading the WebSphere MQ message which was used to store this information in the Request message flow. This is done by using the MQGET node.
- Creates a WebSphere MQ message containing the transformed message and the retrieved ReplyToQ and ReplyToQMgr values.

The Reply message flow consists of the following nodes:

Further information about the Coordinated Request Reply sample can be found in the WebSphere Message Broker section of the Application samples category, which is in the samples gallery of the WebSphere Message Broker development toolkit.

Data Warehouse

The Data Warehouse sample demonstrates a scenario in which a message flow is used to archive data, such as sales data, into a database. The

data is stored for later analysis by another message flow or application.

Because the sales data is analyzed at a later date, the storage of the messages has been organized in a way that makes it easy to select records for specified times. The date and time at which the WebSphere MQ message containing the sales record was written are stored as separate column values when the message is inserted into the database. The database table contains four columns:

- The message data, which is the payload of the WebSphere MQ message stored as a BLOB.
- The date on which the WebSphere MQ message was created.
- The time when the WebSphere MQ message was created.
- A time stamp created by the database to record the time when the record was inserted.

By storing the data in this way it is possible to retrieve records from specific periods; for example, 9:00 a.m. to 12:00 p.m. and 12:01 p.m. and 5:00 p.m., which would allow a comparison of morning and afternoon sales to be made.

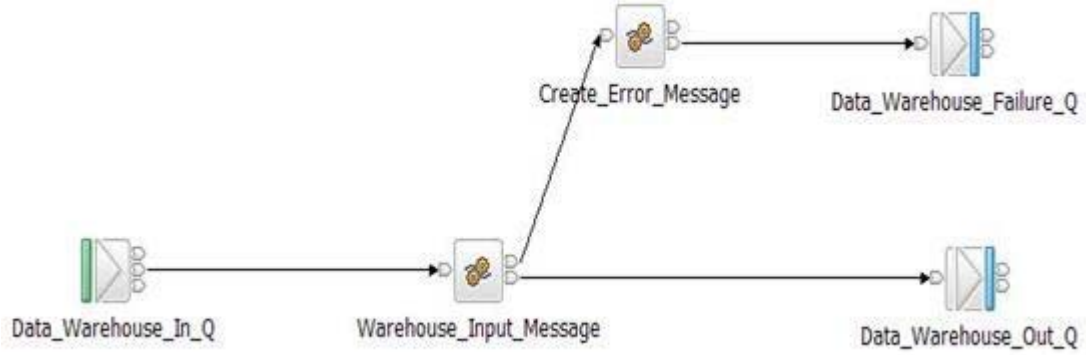
The data archiving is performed by the WarehouseData message flow. This is described below.

WarehouseData Message Flow

The WarehouseData message flow performs the following processing:

- Reads a WebSphere MQ message containing an XML payload, which is the data to be archived.
- Converts a portion of the message tree to a BLOB ready for insertion into the database.
- Inserts the message BLOB along with the date and time at which the WebSphere MQ message was written into a database.
- Sends a WebSphere MQ confirmation message to signal successful insertion of the message into the database.

The WarehouseData message flow consists of the following nodes:



Further information about the Data Warehouse sample can be found in the WebSphere Message Broker section of the Application samples category, which is in the samples gallery of the WebSphere Message Broker development toolkit.

Large Messaging

The Large Messaging sample is a sample based on the scenario of end-of-day processing of sales data. Messages recording the details of sales through the day are batched together in the store for transmission to the IT centre. On receipt at the IT centre the batched messages are split into their constituent parts for subsequent processing.

This splitting is achieved using a WebSphere Message Broker message flow. Each of the individual messages representing a sale has the same structure.

The input and output messages in this sample are implemented as self-defining XML messages for simplicity. Other message formats could be used.

Each input message consists of three parts:

- A header, which contains the number of repetitions of the repeating SaleList structure that follows.
- The body, which contains the repetitions of the repeating SaleList structure.
- The trailer, which contains the time the message was processed.

The aim of the processing in this sample is to write each instance of the SaleList structure as a separate WebSphere MQ message while minimizing overall memory requirements.

The message flow implements a memory saving technique by using a mutable message tree.

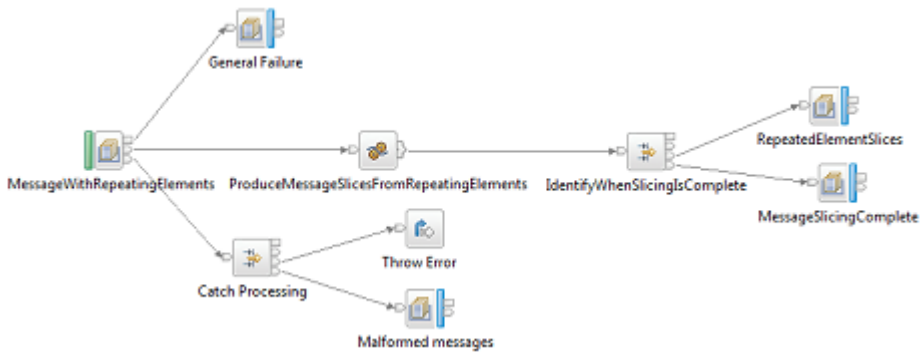
The processing in the sample consists of one message flow. The processing it performs is described below.

Large Messaging Message Flow

The large messaging message flow performs the following processing:

- Reads a WebSphere MQ message containing an XML payload under transactional control.
- Formats a WebSphere MQ message for each instance of the SaleList structure.
- Writes the WebSphere MQ messages to the output queue.
- Produces a WebSphere MQ message to signal completion of the processing when the final element has been processed.

The Large Messaging message flow consists of the following nodes:



Further information about the Large Messaging sample can be found in the WebSphere Message Broker section of the Application samples

category, which is in the samples gallery of the WebSphere Message Broker development toolkit.

Message Routing

The message routing sample shows how a database table can be used to store routing information, which a message flow can then use to route messages to WebSphere MQ queues.

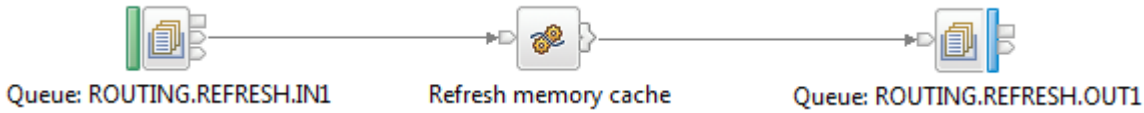
The message routing sample shows how to implement a routing table, using shared variables, to route messages in a message flow.

The processing in the message flows is described below:

Routing Using Memory Cache Message Flow

The message flow performs the following processing:

- Reads a WebSphere MQ message containing an XML payload under transactional control.
- Creates a destination list based on data that is held in shared variables.
- Produces a WebSphere MQ output message. The destination of the message is specified in the destination list.



Further information about the Message Routing sample can be found in the WebSphere Message Broker section of the Application samples category, which is in the samples gallery of the WebSphere Message Broker development toolkit.

Transformation using ESQL

The transformation using ESQL use case is based on processing sales data. At the time of a sale the customer's name, the code for the product, a description of the product, its category, the unit price, and quantity purchased are recorded. Each customer might purchase several items.

Subsequently a statement is produced for each customer and it is the production of the statement that is performed in this use case. The processing results in a restructuring of the original message.

The messages used (input and output) are self-defining XML messages. Each message with sales data consists of three parts:

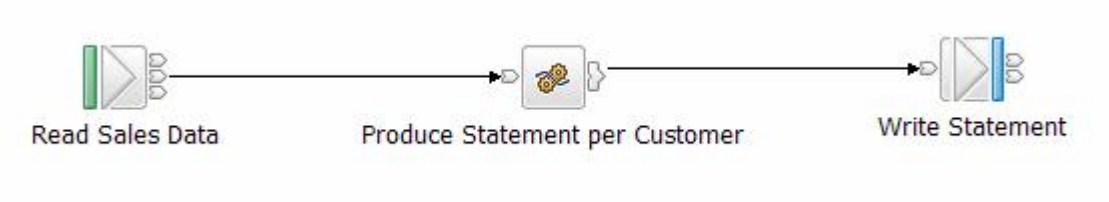
- A header, which a count of the number of repetitions of the repeating SaleList structure that follows.
- The body, which contains the repetitions of the repeating SaleList structure.
- The trailer, which contains the time the message was processed.

The production of the statement for each customer within a SaleList is achieved with a single message flow, the Transformation with ESQL Message Flow.

Transformation with ESQL Message Flow

The message flow performs the following processing:

- Reads a WebSphere MQ message containing an XML payload under transactional control.
- Parses the input message and produces an invoice for each customer. This is achieved with a single Compute node containing ESQL.
- Produces a WebSphere MQ output message containing an XML payload under transactional control.



Appendix E - Tuning

This appendix describes the tuning that was applied to WebSphere Message Broker, WebSphere MQ, and DB2.

The description of each parameter is brief; a detailed discussion is beyond the scope of this document.

WebSphere Message Broker

The WebSphere Message Broker was configured in the following ways for all tests:

1. The broker ran as a trusted WebSphere MQ application. This was achieved by use setting "-t" flag on the mqsicreatebroker command and by ensuring that the environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker was started. This does improve performance, but note there is a potential loss of integrity to the WebSphere Message Broker queue manager because the level of isolation between the WebSphere Message Broker and queue is reduced.
2. Transactional support was used when processing persistent messages . It was not used with non-persistent messages. The use of transaction control means that message processing takes place within a WebSphere MQ unit of work. This involves additional CPU and I/O processing by WebSphere MQ because the unit of work is recoverable. The result is inevitably a reduction in message throughput for persistent messages. By default the transaction parameter on the MQInput node was set to automatic. This is the prefered value to use for transaction mode unless there is a specific requirement to use a particular value, because persistent messages are processed within transactional control and non persistent messages are not.

Additional Tuning for SOAP, HTTP and SCA Tests:

- The clients sending data to the broker were configured to use persistent HTTP connections, that is MaxKeepAlives was set to 0.
- In the execution group, the persistent connections (MaxKeepAlives) value was set for the SocketConnectionManager and HTTPConnector objects.
- In the execution group and HTTP listener, the tcpNoDelay value was set to true for the HTTPConnector and SocketConnectionManager objects.
- In the execution group and HTTP listener, the maxThreads value was set to 2000 for the HTTPConnector object.
- The JVM heap was set to 1GB to allow processing of large messages.

To set these values consult the documentation for the mqsichangeproperties command.
There was no error processing and no error conditions set in any of the measurements. All messages were successfully passed from one node to another through the Out or True terminal. No messages were passed through the Failure terminal of a node.

WebSphere MQ

The following changes were made to all queue managers used in the tests:

1. The value of DefaultQBufferSize and DefaultPQBufferSize was increased to 50MB for the input and output queues used in the tests. This is the maximum supported and was used because in most tests, messages of up to 20MB were used. When using smaller messages all of the time, a smaller value is likely to be more appropriate.
2. Given the use of persistent messages in the tests, the following MQ log parameters were modified:
    - LogBufferPages was set to 4096
    - LogFilePages was set to 65535
    - LogType was set to circular
    - LogPrimaryFiles was set to 15
    - LogSecondaryFiles was set to 1
3. Circular logging was set for all WebSphere MQ queue managers used in the tests.
4. The following values were set for the TCP stanza in the queue manager .ini file:
    - SndBuffSize=70000
    - RcvBuffSize=70000
    - RcvSndBuffSize=70000
    - RcvRcvBuffSize=70000
    - Blocking=YES
5. The WebSphere Message Broker queue manager MQ listener and channels were run as trusted applications. In the queue manager qm.ini file the value MQIBindType was set to FASTPATH in the channel stanza. The environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker queue manager was started.
6. The WebSphere MQ queue manager log was located on a SAN disk with a non-volatile fast write cache used for the disk on which the log was located. Such disks are consistently capable of I/O times of 1 ms compared with a time of 6 ms for a 10,000 RPM SCSI disk. When using a disk with a fast write cache it is essential that it has a non-volatile capability because the log data is critical to the integrity of your queue manager.

For further information on WebSphere MQ tuning, see this article: http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html

TCP/IP

No specific tuning was performed for TCP/IP. All machines used the operating system default values.

Database

1. The database used in the performance tests was modified from the default in the following way:
2. The TCP/IP loopback adapter was used for the database.
3. The database data and log files were placed on a dedicated file system that was located on a SAN disk with fast write non-volatile cache. This was done to minimise I/O times and improve the capacity of the log.
4. The database was modified using these commands:
    - db2 update db cfg for userdb using logprimary 10

- db2 update db cfg for userdb using logfilsiz 250000
- db2 update db cfg for userdb using logbufsz 4096

Additional Tuning Information

To obtain the maximum message rate for your implementation, it is important that you understand the best practices for WebSphere Message Broker. These practices cover the architecture of message flow processing, the coding of message flows, the configuration, and tuning of the broker and associated components.
Such information can be found in the Business Integration Zone of WebSphere Developer Domain.
There is also a SupportPac, IP04, which covers the main design decisions when building message flows. It is available at http://www.ibm.com/support/docview.wss?uid=swg24006518