IBM WebSphere MQ Integrator Message display, test and performance utilities

Version 3.23

18th November 2002

Jim MacNair
MQSeries Sales Support
IBM
Route 100
Somers, NY
USA

macnair@us.ibm.com

Take Note!

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

NinthEdition, November, 2002

This edition applies to Version 3.23 of WebSphere *MQ Integrator – Message test, display and performance utilities* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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
- MQSeries
- MQSeries Integrator
- MQSI
- IBM WebSphere
- IBM WebSphere MQSeries
- WMQ
- IBM WebSphere MQ Integrator
- IBM WebSphere MQ Integrator Broker
- IBM WebSphere Event Broker
- WMQI
- AIX
- VisualAge

The following terms are trademarks of other companies:

- Microsoft Corporation Windows NT, Windows 2000, Microsoft Visual C++

- Sun Sun, Solaris, Sparc

Acknowledgments

The author would like to thank Tim Dunn of the IBM Hursley Lab (ideas on the performance driver utility) and Neil Kolban of the Dallas Systems Center for their assistance. The author would also like to thank Phil Coxhead and Richard Brown of the IBM Hursley Lab for his help in porting and compiling the performance utilities on Solaris.

Summary of Amendments

Date	Changes
18 October 2000	Initial release
6 November 2000	Removed spurious menu items, fixed auto-sizing problem and removed message size and file limits
17 January 2001	1) Expanded code page field to 4 characters 2) Fixed bug with EBCDIC display for Both option 3) Expanded Queue names from 32 to 48 characters
1 February 2001	1) Added user id and backout count to MQMD display 2) Fixed problems when RFH was in EBCDIC or used host encoding 3) Added expiry and put date/time to MQMD display 4) Added group id and sequence number to MQMD display 5) Allowed limited modification of correlation id and group id 6) Display message, group and correlation ids in ASCII, EBCDIC or hex. 7) Changed MQMD format to always reflect the user data 8) Simplified input of encoding for binary fields in RFH 9) Added support for direct addressing of remote queues with remote queue definition 10) Added performance measurement utilities 11) Fixed problem with display of signed numbers in COBOL copybooks
1 March 2001	1) Fixed bug in MQPUT2 program where groupidx was overwriting the qname parameter.
17 April 2001	1) Added support for multiple occurs depending on clauses in a single copybook. 2) Added support for delimiters in message data files. 3) Added additional capture utilities to capture messages in queues and add them to a file.
4 July 2001	1) Added support for browse and non-destructive get, including browse next on data tab 2) Added display of queue depth after MQGet operations 3) Added support for cluster open options

Preface

The Message Display and Test Utility is a GUI based program to assist in the development and testing of WebSphere MQ Integrator applications. It can display messages in a variety of formats, including XML and COBOL copybook representations. It can read data from and write data to files as well as MQSeries queues. It can append and interpret version 1 and version 2 Rules and Formatting headers (RFH). Command line performance measurement utilities are also provided. MQSeries must be installed on the system that this SupportPac is executed on. MQSeries V5.1 with CSD 4 or above or MQSeries V5.2 is required. Separate client and server versions of the Display and Test Utility program are provided. The Display and Test Utility is provided as two executable programs, one for client environments and the other for server environments. The command line utilities are provided as both executable programs for the Windows NT and Windows 2000 environments and as ANSI C source programs, including the Microsoft Visual C++ project files. Executable versions of the command line performance utilities are also provided for the AIX environment.

Although the utilities are designed for IBM WebSphere MQ Integrator V2.1, in most cases they can be used with IBM MQSeries Integrator V2.02, V2.01 and V2.0. Similarly, the utilities are developed and tested using IBM WebSphere MQSeries V5.3, but should work with IBM MQSeries V5.2 or with IBM MQSeries V5.1 with CSD 4 applied.

Chapter 1. Introduction

To use the utility, the data portion of a message should be entered into a file. The file data is then read, and can be displayed on the data tab in several different formats. If an RFH header is desired, then the RFH tab should be displayed and the appropriate RFH header fields completed. The data can now be sent as a message, by filling in the queue manager and queue name fields on the main tab, and pressing the *write queue* button. A message can also be read from the queue and displayed in the data tab.

Once data is read from a file or message, it will be held in a data buffer in memory. The data in the data buffer will not be changed by the utility.

Version History

Changes in V1.0.1

The utility was modified to operate as a dialog application rather than a single document frame window, to fix a problem with auto-sizing that resulted in truncation of the right side of the display area in certain circumstances. The menus were removed, since they served no useful purpose. Size limits on messages and files were removed. Some care should still be taken with very large messages (e.g. > 4MB), since a considerable amount of memory can be used. It is not recommended that the display tab of this utility be used for messages or files in excess of 4MB.

Changes in V1.0.2

The code page field was expanded to 4 characters to allow for code pages with more than 3 digits, and the queue name field was expanded to 48 characters from 32. A bug was fixed which ignored the EBCDIC option when both hex and character data was displayed using the both display option.

Changes in V2.0.1

A significant number of new functions and features were added, including the following:

- Performance measurement utilities added
- Fixed problem with RFH in EBCDIC or using host encoding
- Added field for remote queue manager name
- Option to display correlation id, message id and group id in hex, ASCII or EBCDIC
- Format field now applies to user data in all cases
- Added additional fields to MQMD display (user id, expiry, backout count, etc)
- Allow limited editing of correlation id, group id and expiry
- Rearranged and cleaned up some minor aspects of the various display pages
- Improved performance and handling of XML data

Changes in V2.1

- Added support for multiple OCCURS DEPENDING ON clauses in a single COBOL copybook.
- Corrected handling of SIGN SEPARATE.
- Fixed bug in MQPUT2 program where groupidx was overwriting the qname parameter.
- Added capture utilities to allow messages in a queue to be read and written to a file.
- Added support for multiple messages in a single data file, by use of a delimiter string.
- Fixed bug in AIX version of MQPUT2 utility when using RFH headers.

Changes in V2.11

- Added support for browse and non-destructive get, including browse next on data tab
- Added display of queue depth after MQGet operations

- Fixed bug in rfhutil.exe when entering Correlation Id manually
- Added support for cluster open options
- Fixed bug where MQCapture program was ignoring msgcount parameter

Changes in V2.12

- Added mqtimes2 performance assessment utility, to circumvent problem with non-persistent messages overflowing the queue buffer and causing extraneous disk I/O.
- Added purge option to rfhutil.
- Added drop down box to queue manager edit control

Changes in V3.0

- Changed RFHUtil program to run as a frame window, with minimize and maximize buttons
- Added tooltips help
- Added limited support for display of data containing Kanji characters
- Corrected problem with purge of Alias queues
- Corrected problem with code page for variable area of RFH2
- Added support for additional RFH folders
- Split packed decimal and binary/float encoding indicators
- Added support for report options
- Added support for publish/subscribe requests
- Added support for JMS and user folders
- Added print support
- Fixed a minor bug in the mqtimes and mqtimes2 programs which sometimes counted a message in the wrong second.

Changes in V3.02

- Fixed bug in double-byte character display support
- Fixed bug with display using COBOL copy book and nested redefines and occurs clauses
- Added support for indent option for COBOL copy book displays
- Added Appendix C: Exploring Publish and Subscribe
- Added menu items for options when reading and writing files
- Removed Read Ascii button (replaced function with menu option)
- Browse Next and End Browse buttons now disabled on startup
- Corrected Clear Data button to only clear file data
- Fixed bug in COBOL copybook processing of lower case letters in PIC clauses
- Fixed bug in handling of quotes in parameters file in performance utilities
- Corrected bug in use of XML escape characters in data fields in pubsub items
- Corrected bug in display of other folder contents
- Corrected bug in display of pscr folder contents

Changes in V3.1

- Changed internal structure to keep queues and queue manager connections open between requests
- Added CloseQ button to close queue and disconnect from queue manager
- Changed feedback field to allow user to set the feedback field
- Message id and sequence number updated after Write Q operation
- Added Save RFH menu item for file read operations
- Added support for Solaris systems to performance utilities
- Added support for publish/subscribe and jms folders in RFH2 in performance utilities
- Added support for use of alternate user ids
- Added tutorial for message groups and segmentation

- Added support for message segmentation
- Added support for message groups

Changes in V3.1.1

- Added other field to pscr tab for fields not in response folder
- Fixed recent file list
- Allow application identity to be set on put requests (for Bill Matthews)
- Fixed bug in building of RFH2 headers where RFH encoding different than MQMD encoding

Changes in V3.2

- Added support for publish/subscribe messages using RFH1 headers
- Added support for both a version 1 and a version 2 RFH header in the same message
- Added support for additional publish and subscribe options in IBM Websphere MQ Event Broker and base MQSeries publish and subscribe (SupportPac MA0C).
- Added support for pass correlation id to report options.
- Added JMS timestamp field and support for user defined fields in JMS folder.
- Fixed bugs in generating a JMS folder.
- Automatically resize frame window when large fonts option selected in Windows.
- Added support for additional fields in publish and subscribe folder introduced with WMQ Event Broker
- Changed publish and subscriber tutorial so that topics do not start with a slash ("/") character, so that all exercises except for the subscription point exercise will now work with WMQ Event Broker
- Read file will now set the MQMD format field when a file that is read contains an RFH header
- Added support for get by correlation id
- Added support for pass correlation id to report options in MQMD

Changes in V3.21

- Corrected browse operations to not require +get authority on the queue

Changes in V3.22

- Use most recent queue name as the window title
- Added drop down menu for queue names for server version

Changes in V3.23

- Added menu item to control display of system objects in queue name drop down list

What the utility does not do

The RFHUtil utility program provided with this SupportPac will read data from files or queues, write data to files or queues and display data in a variety of formats. However, it is not an editor. While the user data portion of the message can be displayed in a variety of formats, it cannot be changed. Another program must be used to create or change the user data. The utility program can add Rules and Formatting headers (RFH) to messages or files it writes and will format these headers found in messages or files it reads. The headers can include publish and subscribe commands.

There is one exception where the data can be changed. Carriage return and line feed characters can be stripped out of a file when the file is read using the *Read Ascii File* option.

If this data is subsequently written back to a file or to a message, these characters will not be present. The second file would be different than the first.

Unlike the user data, the Rules and Formatting header (RFH) can be changed. A RFH can be removed or added, and the contents of the RFH can be changed.

Chapter 2. Installation

Supported Environments

The programs contained in this SupportPac has been written for and tested in a 32-bit Windows environment. All development was done in a Windows 2000 environment using Microsoft Visual C++ Version 6 and IBM WebSphere MQSeries V5.3, but the programs should work with IBM MQSeries V5.2. MQSeries client and/or server must be installed on the system that the utility is executed on. The GUI-based test and display utility supports 32-bit windows environments, including Windows NT V4 and Windows 2000. The performance utilities are command line based, and executables for Windows, AIX and Solaris environments are provided.

Display Requirements

The RFHUtil.exe utility program requires a display with a minimum resolution of 800x600 if small (normal) fonts are being used in Windows, and a minimum of 1024X768 if large fonts are being used in Windows. A resolution of at least 1024X768 is recommended. When used with a resolution of 800X600, the auto hide property should be selected for the task bar.

Limitations in the RFHUtil.exe utility

The RFHUtil.exe test and display utility supports most message formats and Rules and Formatting headers (RFH), there are certain limitations in what specific message types and formats are supported.

The display tabs impose certain limitations in what can be displayed or created. In particular, the following limitations should be noted:

1. If an RFH header is present, it must be the first header in the message if the header is to be displayed on the GUI panes. Other headers, such as an IMS or CICS header, must follow the RFH.
2. For pub/sub messages, a maximum of four topics can be entered or displayed.
3. For pub/sub reply messages, a maximum of two error responses can be displayed.

Installation

The provided zip file should be unzipped into the desired directory. The main windows executables will be unzipped into this directory. Two additional subdirectories will be created, namely source and AIX. The source subdirectory contains the source programs for the performance utilities. The AIX directory contains the AIX executable programs and a sample make file. The solaris directory contains the Sun Solaris executable programs and a sample make file.

Executable programs for the performance utilities are provided for the Windows, AIX and Solaris environments. The performance utilities are written in standard ANSI C and should work in most Unix environments if compiled for other Unix environments.

The main utility is provided as a single executable program (rfhutil.exe). A separate executable (rfhutilc.exe) is provided for use with MQSeries clients. The programs should be placed in an executable directory and, if desired, a program icon can be created.

Two small command line utilities are provided for the Windows environment only. The first utility (findmqm.exe) will list all queue managers found on a particular system. If the –q parameter is specified, then queue names and types for all queues defined on locally defined queue managers will also be displayed. The second utility (checkcp.exe) will display and verify the classpath environment variable. Additional options are provided to display and verify the path and lib environment variables, and to verify that the entries required for

MQSeries Java support are present in the classpath. For further options for either of these utilities, please execute the utility with an argument of -?.

In addition to the test and display utility, several command line utilities are provided to do performance measurement and testing:

- MQPUT2.exe
- MQTIMES.exe
- MQTIMES2.exe
- MQCapture.exe
- MQCapone.exe
- MQPUTS.exe
- MQPUTSC.exe

The MQPUT2.exe program is a performance driver utility. It reads message data from one or more files and then uses the data to create a test messaging workload based on the file data and a separate parameters file. The MQTimes.exe and MQTimes2.exe programs read messages from an output queue and reports the messages processed per second. The last two programs are special versions of the driver program. The first one writes all the test messages at once, without attempting to balance the input rate to the throughput of the test workload. The final version is a client version of the simplified workload driver. The MQCapture.exe and MQCapone.exe utilities will read messages from queues and store them in files. The MQCapone.exe utility reads a single message and writes it to a file. The MQCapture.exe utility will read multiple messages from a queue and will store them in a file, with a delimiter string separating each message. The author recommends that a local driver program be used if at all possible, but has provided the other two programs for cases where this is not possible. Source for these utilities is provided in the *source* sub-directory. The performance utilities should also be placed in an executable directory and program icons created if desired.

For further details on the performance utilities, please turn to the section of this document that explains their configuration and use.

Executables for the AIX and Solaris environment are also provided for the batch performance utilities. These executables, as well as a Makefile to recompile the utilities, are provided in separate subdirectories (AIX and Solaris). These executables should be uploaded to the desired AIX or Solaris system in binary mode, and if needed, the Makefile and source programs should be uploaded to the AIX or Solaris system in ASCII mode.

Using the source programs

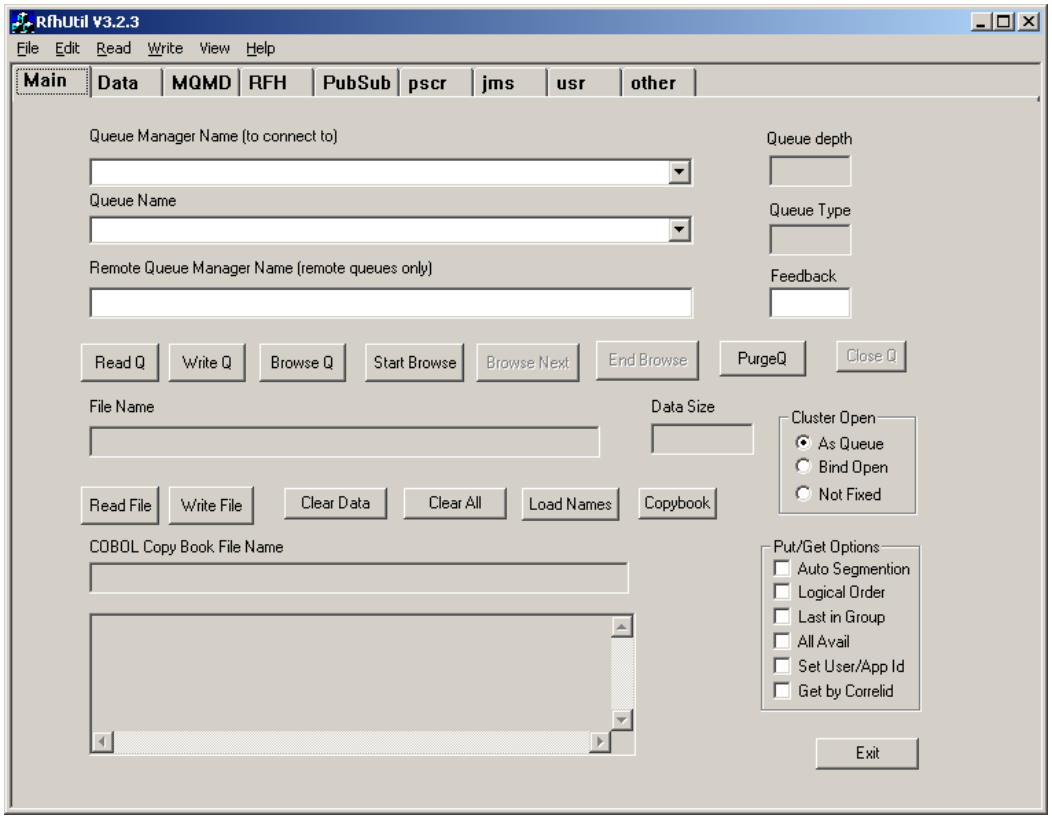
Source code for the command line based performance utilities is provided in the source subdirectory. No source code is provided for the test and display utility. The *source* subdirectory contains the source programs and related project and workspace files for Microsoft Visual C++ V6.

If the source programs are to be used in a Unix environment, such as AIX, then the individual C source files must be uploaded (in ASCII) to the desired Unix system, where they can be compiled. A sample make file for Unix systems is provided in the AIX and Solaris directories. These make files may need to be modified to reflect the actual environment of the Unix system.

The only known dependency that the performance utilities have is on the byte order of binary integers. This dependency affects the processing of Rules and Formatting Headers (RFH). The assumption in the source programs is that Windows platforms use “little-endian” binary integers and non-Windows systems use “big-endian” integers. This is correct for AIX (POWER) and Sun (SPARC) systems. For other Unix systems (such as Linux or Solaris on Intel platforms), a minor change may be necessary to the source code for the performance utilities if RFH headers are to be properly recognized and processed.

Chapter 3. Using the Display and Test Utility

General Tab



Reading and writing from/to a file or queue

Five buttons are provided for the purposes of reading or writing files or messages. The program must first read data from either a file or a queue before it can do anything else. To read data from a file, press the *Read File* button. The options available on the *Read* menu should be set prior to pressing the button. To read data from a queue, enter the name of the queue and if necessary the name of the local queue manager to connect to.

Queue manager and queue names

The server version of the utility program gets the names of the queue managers defined on the system as well as their respective queue names, and then keeps this information in memory. This information is then used when the drop down list boxes are opened to allow selection of the queue manager or queue name. This information can be reloaded by pressing the *Load Names* button. This allows queues or queue managers added since the utility program started to be accessed via the drop down list boxes.

Read Queue button

The Read Queue button will read a message from a queue. The name of the queue manager and queue should be filled in prior to pressing this button. The message will be loaded into a data buffer in memory. The length of the input message will be displayed in the length field on the general tab, and a message area on the bottom will indicate if the read was successful. If the MQGet fails, an error message will be displayed instead. The message data, MQMD and Rules and Formatting header (RFH) can now be displayed, using the appropriate tabs.

When a message is read into the data buffer, two radio buttons on the data tab will be set. If the code page of the input message is either 037 or 500, then the EBCDIC button will be selected. Otherwise, the data is assumed to be in ASCII and the ASCII button will be set. Similarly, if the numeric encoding format indicates host data, then the Host button will be set. Otherwise, the numeric encoding will be assumed to be in a PC format (e.g. reversed byte integers and packed decimal). The numeric encoding setting will only be used if the COBOL copybook option on the data tab is selected and the data is displayed as a COBOL copybook.

Write Queue button

The Write Queue button will write a message to the specified queue. The name of the queue manager and queue should be filled in before this button is pressed, and data should be read from either a file or another queue, using the Read File or Read Queue buttons respectively. If any settings in the MQMD other than defaults are desired, then these should be set before the data is written to a queue. In particular, if the data is in a code page or numeric format other than that of the PC that the program is running on, these fields should be set in the MQMD to match the actual data in the data buffer. If a Rules and Formatting header is to be added to the data, then the appropriate fields in the RFH page should be filled in as well. The data in the data buffer will be used for the user data portion of the message.

The message id and sequence number fields will be updated to reflect the value of the message that was written.

Writing to remote queues

There are two different ways to write to a remote queue. The first method is to create either a remote queue definition in the queue manager that the application connects to or to use a clustered queue and queue manager, which will create the remote queue definition automatically. The program can also write to a remote queue even if there is no remote queue definition on the queue manager that the application is connected to. In this case, the name of the queue manager that the queue resides on must be specified in the *Remote Queue Manager Name* field. In all cases, the appropriate channels and transmission queues must be defined on the local queue manager, either through clustering or direct definitions, and the channels must be working and available.

Browse operations

Two types of browse operations are supported. The Browse Q button will perform a simple non-destructive get operation rather than the destructive get operation performed by the Read Q button. A browse operation that spans multiple messages on the queue can also be performed. Each message in the queue can then be read in sequence.

Browse Queue button

The browse queue button is very similar to the read queue button. It will perform a non-destructive get operation rather than a destructive get. The message will remain on the queue after the get. If the browse queue button is invoked again, the same message will be read.

Start Browse button

The start browse button is used to initiate a browse operation against a particular queue. It will read the first message from the queue, and will then enable the browse next and end browse buttons. The browse next button can be used to read successive messages from the queue.

Browse Next and End Browse buttons

The browse next and end browse buttons are normally disabled. To use the browse next or end browse buttons, a browse operation must first be initiated with the start browse button. Once a browse operation has been started and the first message has been read, the browse

next button can be used to read each successive message in the queue. The browse operation will be terminated when either the end of the queue is reached or the end browse button is used to terminate the browse operation.

Read File button

The *Read File* and button will read the data in a file into the data buffer. The *Read File* button will read the file in binary format in directly into the data buffer.

Certain menu options are provided that can change the data as it is read in. These options are located in the *Read* menu. If the *Remove CrLf* option is selected, the data will be read in binary format directly into the data buffer, and any carriage return or new line characters will then be removed. This capability allows certain types of files that contain character data only to be created and maintained in a format that allows for easier viewing with other editors, but the extra characters can then be stripped out when the data is to be used to drive an application that does not expect the extra characters.

The *Unix Text* menu option allows certain Unix formatted text files to be converted to a PC text file format. A Unix formatted text file will contain only a single new line character between lines of text, whereas a PC formatted text file expects a carriage return and new line sequence between lines of text. If a Unix formatted text file is displayed with certain PC applications (such as Notepad), then the line breaks will not be recognized properly. Once the file has been read in with this option, it can be saved as a PC formatted file by using the *Write File* button. This operation will change the file data and cannot be easily undone

The *Remove CrLf* and *Unix Text* options should not be used to read data files in a binary format (including compressed and encrypted files), or when the message format expects carriage return or line feed characters, such as Swift FIN messages.

When the *Read File* button is pressed, a pop up menu will allow the desired file to be specified.

When a file is read, the RFH header fields are normally reset to default values. If the *Save RFH* menu option is selected, the current values of the RFH header fields will not be changed unless an RFH header is found and the *Ignore RFH* option is not selected.

When a file is read, the program normally looks for the presence of a Rules and Formatting Header (RFH) at the beginning of the file, as described in the following paragraph. If this option is not desired, then the *Ignore RFH* option in the read menu should be selected and the program will not look for an RFH at the beginning of the file and the existing RFH fields will be preserved.

If the first four characters of data in a file are the characters “RFH” followed by a space character, then the utility will assume that the data is preceded by a Rules and Formatting header and this header will be stripped off and the appropriate fields filled in on the RFH tab. This check is performed for both ASCII and EBCDIC. The data buffer will only contain data that follows the Rules and Formatting header. The reason for this is to allow for message data that normally includes a Rules and Formatting header to be stored in a single file. Otherwise, the contents of the Rules and Formatting header would have to be recreated every time the data was read from the file.

If the program determines that there is an RFH header at the beginning of the file, then the code page and encoding for both the RFH and the message data will be set to match the RFH at the beginning of the file. The code page and numeric encoding of the user data will be assumed to match the RFH. If this is not correct, the data type on the data tab and the code page in the MQMD and/or RFH can be changed.

When data is read from a file, the ASCII and PC format radio buttons on the Data tab will be automatically selected. These buttons only affect the data displays and have no direct relationship to the data. When a message or file is written, the data that is written will be the same as the data that is in the data buffer.

If data is read from a file while a group or segment selection is in effect, then the MQMD selections and the current display options (e.g. ASCII/EBCDIC, etc) will not be reset. Otherwise, the MQMD fields and display options will be reset to default options. This

preserves settings such as the persistence that cannot be changed by later messages in a group or segmented message.

Write File Button

The Write File button will write the current contents of the data buffer into a file. If a Rules and Formatting header is specified, a valid RFH or RFH2 header will be written to the file ahead of the data in the data buffer. If just the message data is to be written to the file, then either make sure that the radio button under the *include RFH* on the RFH tab is set to *No* or select the *No RFH* option on the *Write* menu.

When the *Write File* button is pressed, a standard Windows file dialog is invoked to select the file name and location where the data will be written.

Clear Data and Clear All buttons

The *Clear Data* button will delete the contents of the data buffer only. It will not affect the contents of the MQMD nor change the Rules and Formatting Header (RFH) information. It is useful if a message has been published and a different type of publish/subscribe message is to be sent next (with no user data).

The *Clear All* button will clear the data buffer and Rules and Formatting header folders and will initialize all fields in the MQSeries message descriptor and Rules and Formatting header to default values.

Load Names button

The load names button will refresh the list of queue manager and queue names that are kept in memory and used with the drop down lists for queue and queue manager names. This button can be used if a new queue is added to a queue manager.

Q Depth, Queue Type, Feedback and Data Size

The Q Depth field contains the depth of the queue after the last get operation was performed. The queue type and feedback fields will be set after all MQSeries operations (such as reading a message from a queue). The feedback field can be set prior to an MQ put operation. If set it should be a numeric value. The Data Size field contains the size of the current data that was read from a file or a message. They are for information only. The Q depth parameter is only updated after a get operation is performed on an MQSeries queue.

Cluster Open

The cluster open settings will control the setting when the queue is opened. The utility currently will open the queue for each individual put, so this setting may not always produce the expected results. If there are more than one cluster queue available and there is no local copy of the queue on the queue manager that the utility is connected to, then successive messages should be sent to different queue managers.

This setting applies only to MQ put requests.

Get and Put options

Four check boxes are provided to control get and put options related to the use of message groups and the segmenting of messages.

The *Allow/Use Segmen*tation option applies to both MQSeries get and put operations. If specified for an MQSeries get (*Read Q*) operation, a complete message will be returned, even if the message has been segmented into multiple physical messages. If this option is specified for an MQSeries put (*Write Q*) operation then the MQSeries queue manager will split the message into multiple segments if necessary if the message exceeds the maximum message size specified for either the queue or the queue manager.

The *Logical Order* and *Last in Group* options apply to MQSeries message groups. When a group of messages is to be written to a queue, the *Logical Order* option should be selected on

the first message. The message can then be written to the queue. After that, subsequent messages can be read from files and written to the same queue. The *Last in Group* option should be selected before the last message in the group is written. This action will complete the group. When a group of messages are to be read, the Logical Order option should be specified. This will ensure that all messages within a group will be read (or browsed) in order, even if the messages are not in physical order on the queue or other messages are interspersed on the queue.

The *All Avail* option will issue the MQSeries get request with the necessary options such that no message will be read until all segments of a segmented message are available on the local queue and/or all messages in a message group are available on the local queue.

In most cases, the group id field in the MQMD should be left as nulls. When the group id field is null and groups or segments have been requested, the MQSeries queue manager will generate a unique group id and use it for each successive message. In a similar manner, the offset field will be automatically filled in when the MQSeries queue manager creates message segments, and the sequence number will be automatically updated when successive messages are written to the same group.

There are a number of limitations and restrictions when reading or writing a group of messages or when reading or writing a segmented message from or to a queue. A single RFHUtil session must be used to write the entire message group or segmented message. The group id should not be changed (if specified). The persistence, message format, code page and encoding value must remain the same for messages within a segment, and the persistence must be the same for all messages in a group.

Setting the user id on a get or put operation

RFHUtil supports the use of alternative user ids to get messages from a queue and the setting of a different user id when a message is written. The use of these options may require a user to have special authority options set by an administrator with the setmqaut utility.

To use a different user id than the id that RFHUtil was started under, select the *Set User ID* check box. Select the MQMD tab. The user id field should now allow input. Enter the user id to be used for the get or put operation. Return to the main tab and execute the desired MQSeries operation.

For put operations, the queue will be opened with the MQOO_SET_IDENTITY_CONTEXT flag selected. The user id specified on the MQMD tab will then appear in the MQMD of the output message. For input operations (including purge), the queue will be opened with the MQOO_ALTERNATE_USER_AUTHORITY flag selected. The check box will be automatically reset after all input operations, since the user id on the MQMD tab may be changed by the input operation.

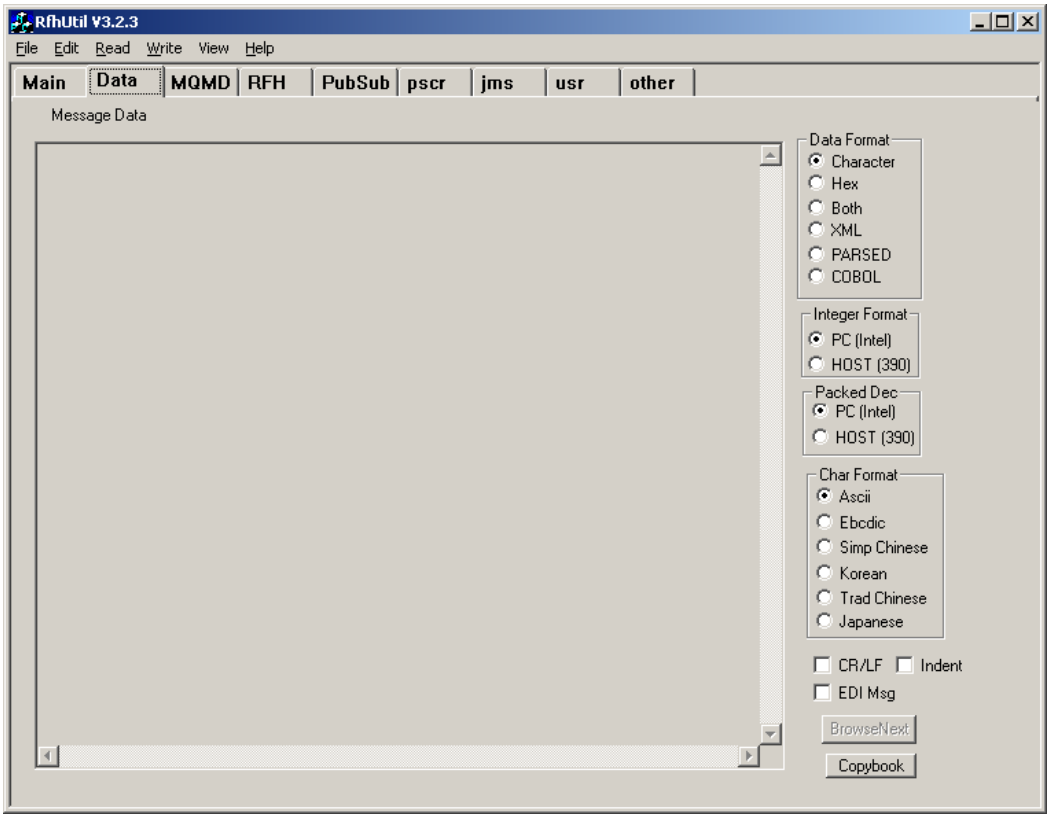
Considerations when using message groups or segmentation

When creating a group or segments of a message, there are several important considerations. The first consideration is persistence. If persistent messages are written in a group, then a unit of work must be used. This will change the behavior of RFHUtil. Normally, messages are visible on the queue as soon as they are written. However, when a group is started and either persistence is requested or the *As Queue* option is selected, RFHUtil must open a unit of work. The unit of work will remain open until a message is written with the *Last in Group* option selected, at which time the unit of work will be committed and all the messages will now appear on the queue. If any of the MQ put operations fail for any reason or if the Close Queue button is pushed, the unit of work will be rolled back and none of the messages written in the unit of work will appear on the queue.

Data tab selections

The Data tab contains a large data window and a number of radio buttons that control the format that the data will be displayed in. The radio buttons will only affect the data display

and do not change the contents of the data in the data buffer. The BrowseNext button is only enabled if a browse operation has been started.



Data Window

The data window displays the contents of the data buffer in the format selected by the radio buttons.

Data formats

The data in the data buffer can be displayed in six different formats. The six formats are as follows:

- Character
- Hex
- Both
- XML
- Parsed
- COBOL

The character, hex and both formats apply to any data format. They show the raw data in either character or hex format, or in both simultaneously. The character display will substitute periods for any non-printable characters. In all cases, an offset will be displayed on the left side, and the data will be in shown in fixed length groups, with spaces inserted periodically in the display to improve legibility. Vertical scroll bars allow for large data buffers to be displayed.

If the data in the data buffer is in a valid XML format, the XML and Parsed options can be selected. The XML format will attempt to display the data as a valid XML hierarchy, with lower level items indented. The Parsed format will display the data in a format that looks like the ESQL statements that would be used to create the given data contents. It is usually easier to see the data values when the parsed format is used rather than the XML format.

The XML and parsed formats should only be selected for data that constitutes a well-formed XML message. Attempts to use these formats for data that is not in XML format or for XML data that is not well-formed (e.g. contains XML formatting errors) will usually result in an error message, but may result in unpredictable results.

The XML and parsed formats will remove any comments and imbedded DTDs. They perform a limited validity check on the data before attempting to format the data. This checking consists of verifying that there is at least one less than sign, that the first less than sign is found before the first greater than sign and that the number of less than signs matches the number of greater than signs. Carriage returns, line feeds and tab characters found in data or attribute values will be replaced with blanks. User defined entity values found in DTDs will be ignored and the data will be displayed in its raw format.

The COBOL option should only be used to match a given data area with a COBOL copybook. To use this option, the COBOL copybook must be available on the local system. When this option is selected, a popup menu will allow the copybook to be specified. The data will then be matched against the given copybook. If a field is specified as packed decimal but the data is not in a valid packed decimal format, an error message will follow the display of the particular field. When this format is selected, character data will be assumed to be in the format specified (e.g. ASCII or EBCDIC), and if EBCDIC is specified, then the data will be translated from EBCDIC to ASCII before it is displayed. Packed decimal and integer data will be assumed to be in the format specified by the numeric format radio buttons (e.g. Host or PC/Intel). If the wrong numeric format is selected, most packed decimal fields will generate error messages. The correct numeric format button should then be selected, and the error messages should disappear.

If the ASCII button is selected, the character data will be displayed as found in the data buffer. If the EBCDIC button is selected, the data will first be translated from EBCDIC to ASCII before it is displayed. For COBOL copybook displays only, data in binary or packed fields will not be translated.

The CR/LF check box will cause the display to honor any carriage return and new line sequences in the data buffer, rather than display the data in fixed length rows. There are certain types of data that are more legible when this option is selected, such as Swift FIN messages.

If the EDI check box is selected and the standard EDI sequences are found at the front of the data area (first three characters of the data), the line termination character in the message will be used to determine the line breaks.

The indent check box is used for displays in a COBOL format. If this option is selected, then lines will be indented to reflect the hierarchy of the associated copybook.

The display data formats do not support Unicode data. The hex display can be used to view Unicode data in a raw format.

Numeric formats

The numeric format radio buttons are only used when data is to be displayed as a COBOL copybook. The integer buttons control the format of integer and floating point data, and the packed decimal buttons control the interpretation of packed decimal data. These selections will indicate whether binary and packed decimal numbers are in PC (Intel) reversed format or Host (390) / Unix format. This selection will not affect the data in the data buffer.

Three options are provided to handle the common encoding schemes for binary, packed and floating point numbers. The choices of PC, Host and Unix correspond to an encoding value of 544, 785 and 273 respectively. The difference between the Host and Unix selections is the format of floating point numbers (390 format vs. IEEE format). No data transformation is provided for floating point numbers. They can be displayed in hexadecimal formats only. Integer and packed decimal representations are the same.

The numeric encoding format specified in the MQSeries message descriptor should indicate the numeric encoding of the user data. If the data is all character data, then this setting has no meaning and is ignored.

Character formats

The character format radio buttons indicate whether the data in the data buffer should be translated from EBCDIC to ASCII before being displayed. It is used with all display formats except hexadecimal displays. This selection will not affect the data in the data buffer.

Four additional character formats can be used to select a font that is appropriate for the display of Chinese, Korean and Japanese characters. In each case, the appropriate font must be installed on the system, and the data must be in multi-byte Unicode format (code page 1208).

CR/LF selection

This selection is only used with the character display option. If this option is selected, then carriage return and line feed sequences will cause the subsequent characters to be displayed on the next line, rather than display a fixed number of characters on a line.

EDI selection

This selection is only used with the character display option. The data must be a valid EDI format, including X.12 and EDIFACT. The data must start with the characters “UNA”, “UNB” or “ISA”; otherwise this option is ignored.

Indent selection

This selection applies to COBOL copy book displays only. If selected, then successive levels will be indented in the display.

Data Translation

For convenience when working with EBCDIC (primarily host) data, a limited translation capabilitiy from EBCDIC characters to ASCII characters is provided. This translation is internal to the utility and does not support individual national code pages. It will translate numbers, standard letters and most special characters. This is provided to enhance the display and viewing of character data that has originated or is intended for a mainframe. The user data in a message or file is not altered by this translation. The translation of data is for display only. In addition to the data display, the header and data fields of an RFH or RFH2 header will be translated.

Binary data in either a COBOL copybook display or an RFH or RFH2 header will be transformed if the encoding parameter in the MQMD indicates that the numeric format is host (big-endian) rather than PC (little-endian) format.

Unlike MQSeries messages, files do not indicate the code page of the data. If data is read from a file on a PC, the data will be examined and the utility will make a guess if the data is ASCII or EBCDIC and will set the initial display options as a result of this determination. If this determination is not correct, the display option radio buttons should be changed to match the data. To allow EBCDIC data to be stored in PC files and recognized as such, the program will assume that any data in a file with an extension of “ebc” contains ebcdic data and the code page and display characteristics will be set for EBCDIC rather than ASCII data. In all cases, the data itself will not be transformed.

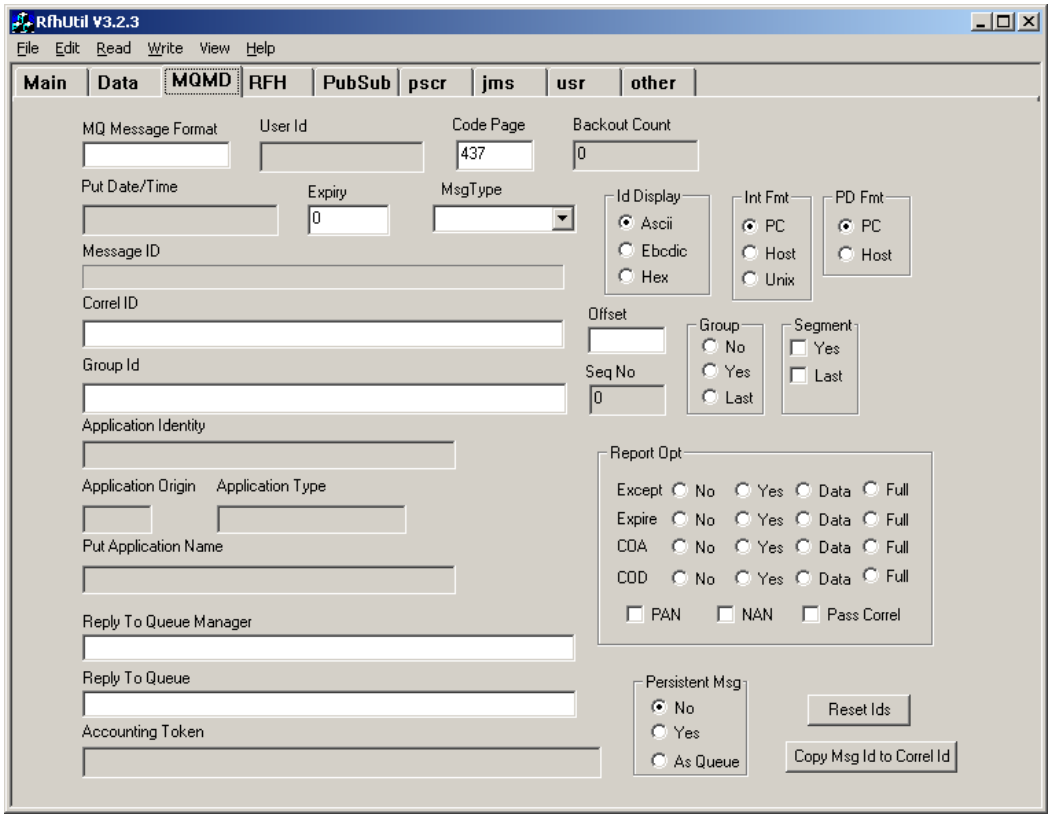
Browse Next button

The browse next button is normally disabled. It will be enabled when a browse operation has been started using the start browse button on the general tab. When enabled, the browse next button will read the next message from the queue. If there are no more messages on the queue, the browse operation will be terminated.

Copybook button

This button allows the copybook file to be changed. When the COBOL button is first selected, a popup menu is displayed that allows the copybook file to be selected. This button allows the copybook selection to be changed.

MQMD tab



The MQMD tab contains fields that display many of the commonly used fields in the MQSeries message descriptor. In some cases, the contents of the display can be changed, such as the code page. When data is to be written to an MQSeries queue, the code page should be set to match the actual data in the data buffer. If the code page or numeric formats are set incorrectly, subsequent processing of the message data may fail or generate incorrect results. Changing any of the fields in the MQMD does not change the data in the data buffer.

Before a message is written to a queue, the code page should be set to match the RFH if present or the data in the data buffer if no RFH is present. The encoding should be set to match the encoding sequence of any numeric (e.g. binary or packed decimal) data in the RFH if present or the data in the message if no RFH is present. Finally, the message persistence should be selected. The defaults for these values are for a non-persistent message with code page set to 437 (PC ASCII) and numeric encoding set to match a PC.

MQ Message Format

If no Rules and Formatting header is to be used with this message, then the format field should be set to indicate the message format. The standard MQSeries message formats should be entered into this field. If the message is not a standard format, then this field should be left blank (equivalent to MQFMT_NONE). If the data is all character data, then the format field should be set to “MQSTR ”.

In all cases, the format field should reflect the format of the user data. If an RFH or RFH2 header is included with the message when written to a file or queue, then the contents of this field will actually be found in the RFH or RFH2 header. The MQSeries message descriptor format field in this case will contain a value of MQHRF or MQHRF2 respectively. This will be done automatically by the utility.

User id, Application Type and BackoutCount

These three fields are taken directly from the MQSeries message descriptor when a message is read from a queue and are for display purposes only.

Put Date/Time and Expiry

The Put Date/Time field will display the date and time that the message was written to the queue. This field is taken from the MQSeries message descriptor when a message is read from a queue, and is for display purposes only.

The Expiry field is used to display the expiry field in the MQSeries message descriptor when a message is read from a queue. It is also used to set the message expiry when a message is written to a queue and the expiry is set to a value greater than zero.

Message Id, Correlation Id and Group Id

The message id, correlation id, group id and sequence number fields are used to display the respective fields in the MQSeries message descriptor. These fields can be displayed in character (ASCII or EBCDIC) format or in hexadecimal format, by selecting the appropriate radio button in the *id Display* group.

The message id and sequence number fields are for display purposes only. The correlation id and group id fields will be used when messages are written to a queue, if they are set to a value other than binary zeros.

Editing the correlation id and group id

Limited editing of the correlation id and group id is allowed. The editing can be confusing, since the values can be displayed in ASCII, EBCDIC or hex, and the data may contain non-printable characters, such as nulls (binary zeros). A standard windows edit box is used to display the data. If the display is in ASCII or EBCDIC, no data would normally be displayed after the first null character. To show all printable characters, any null characters are replaced with blanks, and trailing blanks are then removed. This will display all printable characters but causes problems when the data is changed.

When the data is changed, each individual character is compared to the previous value, and only characters that have different values will be updated. If characters are deleted, by using the delete key or the backspace key, some characters after the deleted character may appear to have been changed and will be updated with the values in the edit box display. For example, some null values may be changed to a space character. This problem can be avoided if all editing is done with a hex display.

Edit boxes in Windows do not support an overtype mode. For example, when the correlation id or group id is displayed as a hex value, characters must be deleted (with either the delete or backspace keys or by a mouse selection) and new values typed in. This is less of a problem with ASCII and EBCDIC displays, since trailing blanks are removed from the end of the display string and therefore some insertion is usually possible without any previous deletions.

One quirk in the translations is that the ASCII values for code points from X'30' to X'39' will be displayed as numbers when the EBCDIC display option is selected. This is due to the translation of these code points to the same values when translating from EBCDIC to ASCII. These EBCDIC code points have no valid representation in ASCII.

Report options

Six report options are supported as check boxes. When a message is read from a queue, the appropriate report option check boxes are set depending on the contents of the report field in the message. When a message is written to a queue, the report field is set to match the report option check boxes. If any report options are set, then the reply queue name field must be set. The reply queue manager field may also need to be set.

The supported report options are exception, expiration, confirm on arrival (COA), confirm on delivery (COD), positive activity notification (PAN), negative activity notification (NAN) and pass correlation id. The pass correlation id option instructs the reply application to set the correlation id to be the same at the correlation id in the request message.

When a request message is sent to the WMQI broker, it is treated as if the PAN and NAN report options had been selected.

Application Identity, Application Origin and Put Application Name fields

These three fields contain the respective fields from the MQSeries message descriptor after a message is read from an MQSeries queue. These fields are for display purposes only. If the *Set User/App Id* option is selected on the main tab, the *Application Identity* field can be changed and the value of this field will be used for any *Write Q* operations.

Reply to queue and queue manager

These two fields contain the name of the reply to queue manager and queue, as specified in the MQSeries message descriptor when a message is read from a queue. These fields can also be used to set these two fields when a message is written to an MQSeries queue. Both names can be a maximum of 48 characters.

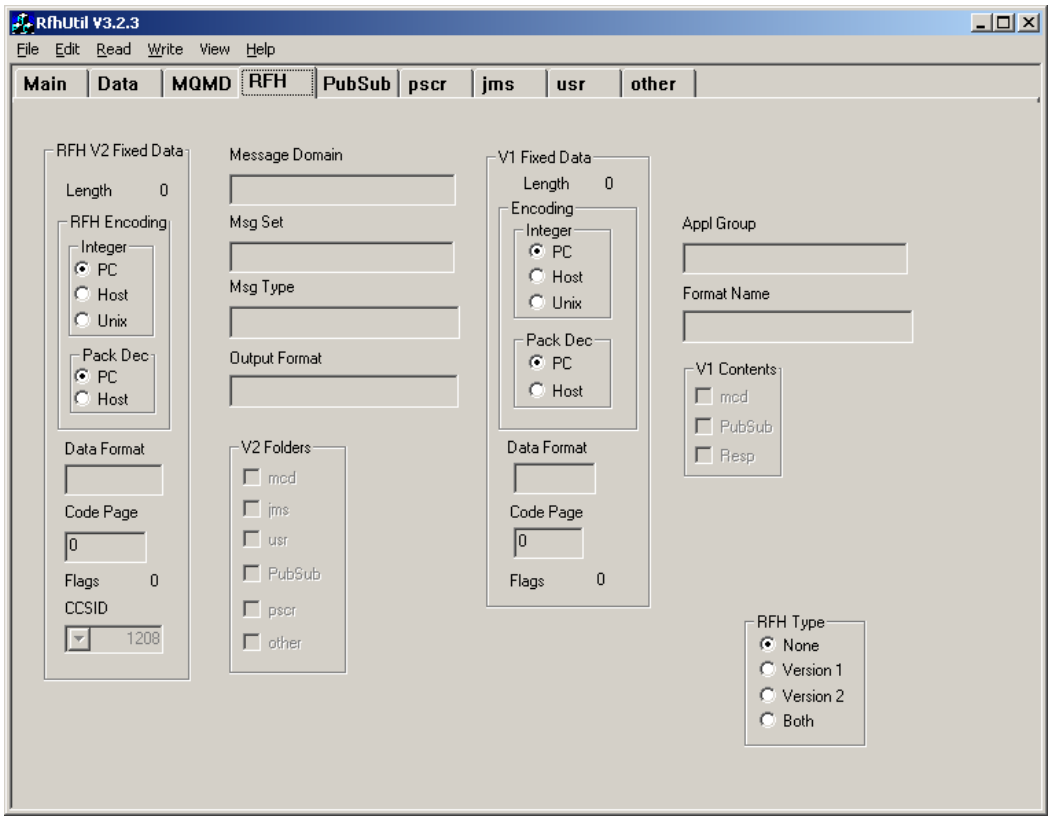
Reset Ids button

This button will clear the message id, correlation id and group id fields and set the values to nulls (binary zeros).

Copy msgid to correlid button

This button will copy the message id field to the correlation id field. This button is particularly useful when used with the *Get by Correlid* option on the main tab.

RFH tab selections



A Rules and Formatting header is an optional data header that gives the message broker information about the format of the message data. The Rules and Formatting header is contained at the front of the user data in a message.

There are two versions of the Rules and Formatting header (version 1 and version 2). Each header contains a fixed portion followed by an optional variable portion. The fixed portion identifies and describes the Rules and Formatting header itself. The variable part contains optional data fields. For a version one header, the fixed portion of the header is thirty-two bytes long, while the fixed portion of a version two header is thirty-six bytes long.

The variable portion of a version one header contains fields that can define the format of the user data (the application group and format name) and fields that are related to publish and subscribe requests.

The variable portion of a version two rules and formatting header (RFH) is divided into a number of XML folders. The mcd folder contains up to four fields that define the data type and format, namely the message domain, set, type and format. The domain indicates the parser that will process the message data. The parser determines the meaning of the other three fields. The variable portion can also contain folders related to publish and subscribe, JMS, user data and other miscellaneous user-defined folders. All folders are optional, and more than one folder may be present in the same RFH header. The desired folders can be selected using the check boxes on the RFH tab. These check boxes will be automatically set if a message or file is read that contains an RFH.

If a Rules and Formatting header is present, the format fields in the MQSeries message descriptor will be set to the literal “MQHRF “ or “MQHRF2 ”. This will be done automatically when the RFH type radio button is set to Version 1 or Version 2. The format field, encoding and code page fields of the Rules and Formatting header should be set to the format of the data that follows the RFH (which can be user data or another MQSeries header).

Multiple RFH headers in a single message

A single MQSeries message can contain both a version 1 and a version 2 Rules and Formatting header. For example, most publish and subscribe messages generated by the JMS support provided with IBM WebSphere MQSeries contain both headers. The option to have both headers is selected with the Both radio button. In this case, the RFH version 1 header is normally the first header in the message, followed by the RFH version 2. The MQMD format field should identify the first RFH header, and the format field in the first RFH header should identify the second RFH header. The format field in the second RFH header should identity the type of data that follows the second RFH header. In a similar vein, the code page and encoding settings for the MQMD and RFH headers should follow a similar pattern (the setting in the MQMD should match the first header and the setting in each RFH should match the settings of the data that immediately follows the particular RFH).

RFH fixed data

The version, length and flags fields in the Rules and Formatting headers will be set automatically, based on the radio button that is selected. The code page field and the RFH encoding radio buttons should be used to select the desired byte ordering of the binary fields in the data that immediately follows the Rules and Formatting header. The CCSID field is only used for a Rules and Formatting header that is in a version 2 format. This field denotes the code page used for the variable portion of the Rules and Formatting header. It is limited to one of four values, and should be set to code page 1208. The utility does not support variable data in a wide character format.

RFH variable data

If the Version 1 radio button is selected, then the application group and message format fields should be filled in to match the user data in the message. Similarly, if the Version 2 radio button is selected, one or more of the four fields in the message format header should be selected. Data for other folders is contained on other tabs. Publish and subscribe fields are

located on the *PubSub* and *pscr* tabs. The *jms*, *usr* and *other* tabs can only be used with a version two header.

Include RFH options

These options indicate whether a Rules and Formatting header should be included with the message or file data when it is written, and if so, what type of header should be included.

If the “None” option is selected, then no Rules and Formatting header (RFH) is included with the data when it is written to either a queue or a file. If the “Version 1” option is selected, then the application group and format name fields can be entered. When the message is subsequently written to a file or queue, a version 1 Rules and Formatting header is included with the data. Similarly, if the “Version 2” option is selected, then the appropriate fields can be entered and a Rules and Formatting header will be included with the data when it is subsequently written to a file or queue. The both option indicates that both a version one and a version two header are to be included with any messages or files that are written.

When a message is read from either a queue or a file, the appropriate option is automatically set.

Include Folders

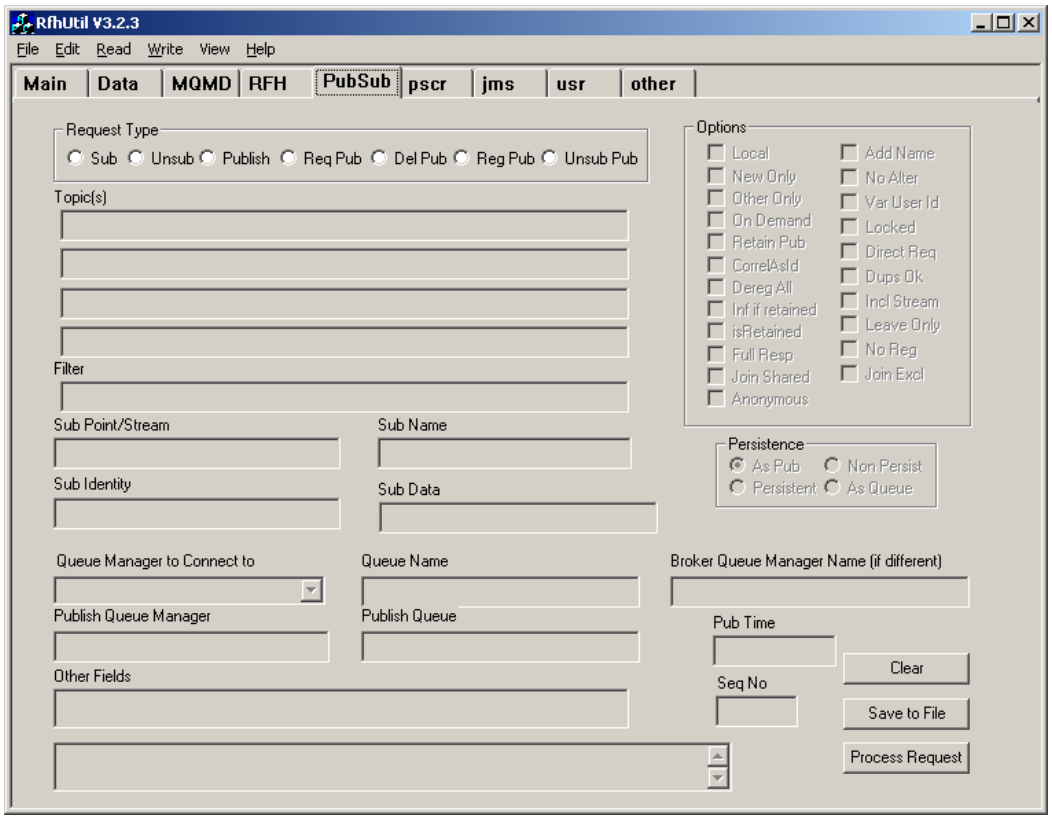
The check boxes indicate which folders are to be included with an RFH2 header. They determine the contents of the variable portion of the RFH2 header. For example, if the mcd box is checked, a mcd folder will be included with the RFH2 header. The appropriate fields on the RFH tab should be filled in for the requested mcd folder.

Separate check boxes apply to an RFH version 1 header. A version 1 header can contain application group and format name fields, as well as fields associated with publish and subscribe requests and responses.

The same data can be included in both an RFH version 1 and an RFH version 2. For example, it is possible to include publish and subscribe information in both headers. While the RFHUtil program will allow this, it is very likely to cause problems with the receiving application and is considered an error. The mcd, publish and subscribe and pscr (publish and subscribe reply) information should not be selected in both headers for a single message.

For examples of most of the fields in the publish and subscribe (psc) folder, see the tutorial in the appendices.

PubSub tab



The PubSub tab is used to generate publish and subscribe requests. This tab will allow most but not all publish/subscribe commands to be entered or displayed. Please be aware that different publish and subscribe brokers will not support all of the options that are available.

The RFHUtil program is intended to support three publish and subscribe environments, namely WebSphere MQ Integrator (WMQI), WebSphere MQ Event Broker (WMQEB) and base MQSeries publish and subscribe (available as SupportPac MA0C). Some options are unique to a particular publish and subscribe environment. The base MQSeries publish and subscribe offering does not support RFH2 headers. All of the publish and subscribe environments do not support all of the options and fields.

The register publisher (reg pub) and unregister publisher (unreg pub) request types are only supported by base MQSeries publish and subscribe and can only be used with a version 1 header.

Request Type

The request type specifies the type of request. Once the request type is selected, then the appropriate input fields for that type of request will be enabled.

Topic(s)

There are four topic string fields, which allow a maximum of four topic strings to be entered, depending on the request type. Each topic is a separate complete topic.

Filter

Enter a filter string, if desired. The filter string is an ESQL expression that is used for content based filtering.

Sub Point/Stream

Enter the subscription point within the topic tree that this subscription should be entered at. Subscription points are sometimes used for versioning among other uses. If this request is being sent to a base MQSeries publish and subscribe environment, this field contains the stream name.

Sub Name, Identity and Data

WebSphere MQSeries Event Broker currently supports these three fields. These fields allow a subscription to be shared by multiple users. They are used by WebSphere Application Server.

Queue manager to connect to

Enter the name of the queue manager to connect to. If this field is left blank, then the default queue manager on the system will be used.

Queue Name

For publication requests, enter the name of the queue to send the publication to. This queue must have an active message flow that will process the message and create a publication, by the use of a publication node. For all other request types, the queue name is fixed at SYSTEM.BROKER.CONTROL.QUEUE.

Broker Queue Manager Name

Enter the name of the broker queue manager, if different from the queue manager that RFHUtil is connected to. A channel between the broker queue manager and the local queue manager that the utility is connecting to must have been defined, including an appropriate transmission queue.

Publish Queue Manager

Enter the name of the queue manager to receive published messages.

Publish Queue

Enter the name of the queue that will receive published messages.

Pub Time

This field is the publication time.

Seq No

Enter a sequence number, if desired.

Other Fields

This field contains any fields that are not recognized by the RFHUtil program. This field is only used with a version 2 RFH. Any unrecognized fields found in a version 1 RFH will be put

be put in the Other Info field on the pscr tab. Unlike the other fields on this tab, this field will contain raw XML tags as well as the data value. This field must contain pairs of well-formed XML tags with the corresponding data values. The data values must use the XML escape sequences for certain special characters (such as & for the “&” character).

Options

A number of options can be selected. The actual options that are available will depend on the type of request. The options are available as check boxes, which can be selected and deselected. Options that are selected but do not apply to a particular type of request will be ignored.

The Local option applies to collectives. It indicates that the action is local to the specific broker and should not be propagated to other brokers in a collective.

If the New Only option is selected, then only new publications will be sent. Existing retained publications will not be sent, even if they match the selection criteria.

The Other Only applies to publishers that are also subscribers, and indicates that the publisher does not wish to receive its own publications.

The On Demand option is valid for registering of subscriptions, and is only effective for retained publications. It indicates that the application will issue a request publication when it is interested in receiving a publication, and that publications are not to be automatically sent to the subscribing application as they are published.

The Retain Pub option is used with publication requests and indicates that a particular message is a retained publication. The broker will maintain a copy of a retained publication until it is either replaced with a newer version or it is removed with a delete pub request.

The correlasid option uses the correlation id of the subscription message to identify a publication message. This option is primarily used when a publication queue is shared by multiple applications.

The Inf if Retain option will set an option in the RFH2 header to indicate that a published message is a retained publication. This option only applies to messages that are sent in response to subscribe or request publication requests.

The Dereg All option applies only to deregister requests, and will deregister all subscriptions for a particular publication queue.

The isRetained option is set by the broker to indicate that a particular message is a retained publication. This indication is set by the broker only in response to a Inf if Retain option being set on a subscription.

WebSphere MQSeries Integrator does not support the following options.

A publish or subscribe request with the Full Resp option (full response) selected indicates that a full response is desired in reply to the request. All characteristics of the subscription will be returned with any reply message. The MQMD should specify either a message type of request or the PAN report option if this option is specified.

The Join Shared and Join Excl (exclusive) options allow a single subscription to be shared by multiple users. A subscription identity must be specified, which will be added to the identity set for this subscription. These two options are mutually exclusive. Setting one option will automatically turn off selection of the other option. WebSphere Application Server is the primary user of these options. The broker will return the Locked option if a previous subscription request has requested exclusive use of the subscription.

The Add Name, No Alter and Var User Id options are used with shared subscriptions, and are supported by WMQ Event Broker. If Add Name is specified, then a subscription name is required. The variable user id option allows a subscription to be modified or removed from a different user id than was used to create the subscription. The No Alter option is used when a new subscription is registered. If a subscription already exists, the attributes of the subscription will not be changed.

The Leave Only option applies to deregister (unsub) requests on shared subscriptions. It indicates that the subscriber identity should be deleted from the identity set for the subscription but that the subscription should be maintained even if there are no other users.

The anonymous option applies to base MQSeries publish and subscribe. If selected, information about the publisher will not be sent to subscribers.

The Direct Request and Dups OK options apply to base MQSeries publish and subscriber support. Direct Request indicates that a publication can be requested directly from the publisher. The Dups OK option is a performance optimization. If a subscriber has multiple subscriptions registered with a broker, then it is possible for a publication to match more than one subscription. The broker will normally consolidate the multiple matches and send only a single copy of the published message, regardless of the number of matching subscriptions. If the Dups Ok option is selected on a subscription request, this consolidation is not performed and a message is sent for each matching subscription.

The Incl Stream (include stream name) option indicates that the stream name is to be included in the header of any messages that are sent in response to a particular subscription. The base MQSeries publish and subscribe offering supports this option.

The base MQSeries publish and subscribe support allows publishers to register explicitly. This registration is normally optional, since a publisher will be registered implicitly when a publication is sent on a particular topic and the publisher has not explicitly registered. The No Reg option indicates that this implicit registration is not to be performed, and that the publication request will fail if the publisher has not previously registered as a publisher.

Persistence

The persistence option indicates whether published messages are sent as persistent or non-persistent messages. The default is persistence as the publication. Persistence can also be set to yes, no or as the subscriber's queue.

Clear button

The clear button will clear all data and selections on the pubsub tab.

Save to File button

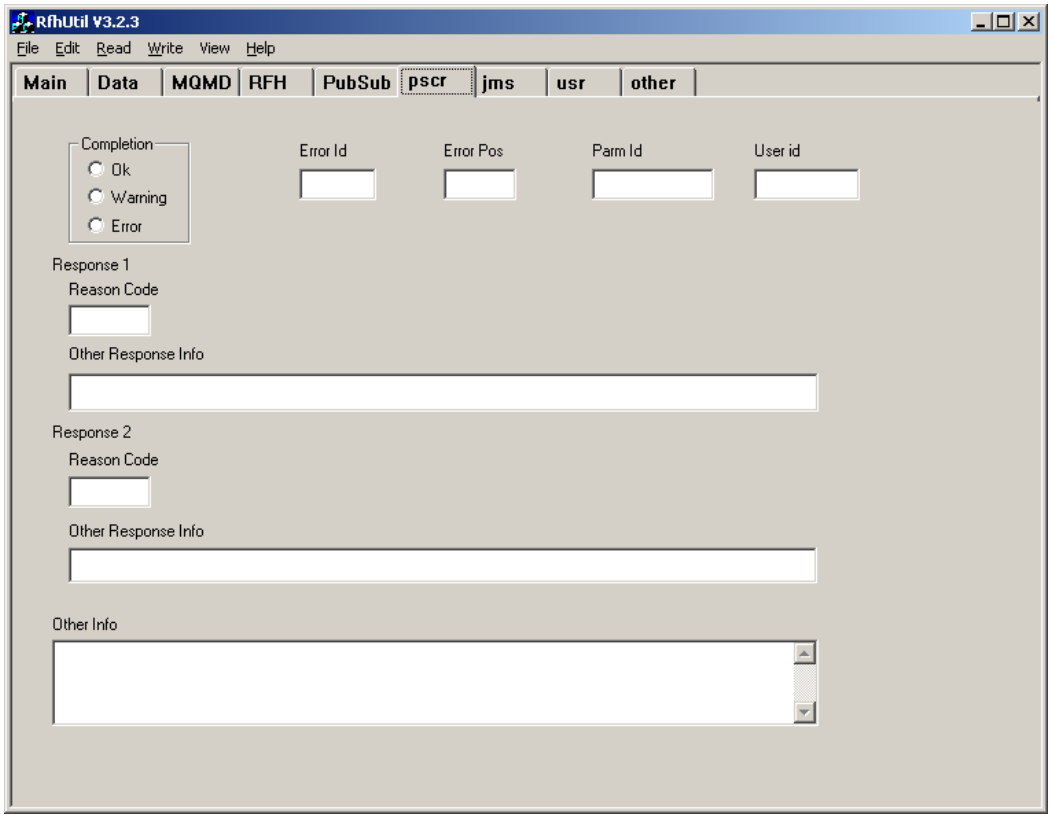
The save to file button will save the current RFH header and any accompanying data to a file. This button is the same as the Write File button on the main tab. A message indicating the results of the file operation will be displayed in the message area at the bottom of the tab.

Process Request button

The process request button will send the current request to the broker, by performing an MQSeries put to the broker queue. A message indicating the results of the MQSeries put operation will be displayed in the message area at the bottom of the tab. This button is the same as the Write Q button on the main tab.

pscr tab

The pscr tab is used to display fields within a publish/subscribe response folder.



This tab will be populated with the contents of a pscr (publish/subscribe response) folder in an RFH 2 header that is found in a message or file that is read by the RFHUtil program. It is also used to display fields found in a version 1 header that are contained in broker responses. A maximum of two responses will be displayed.

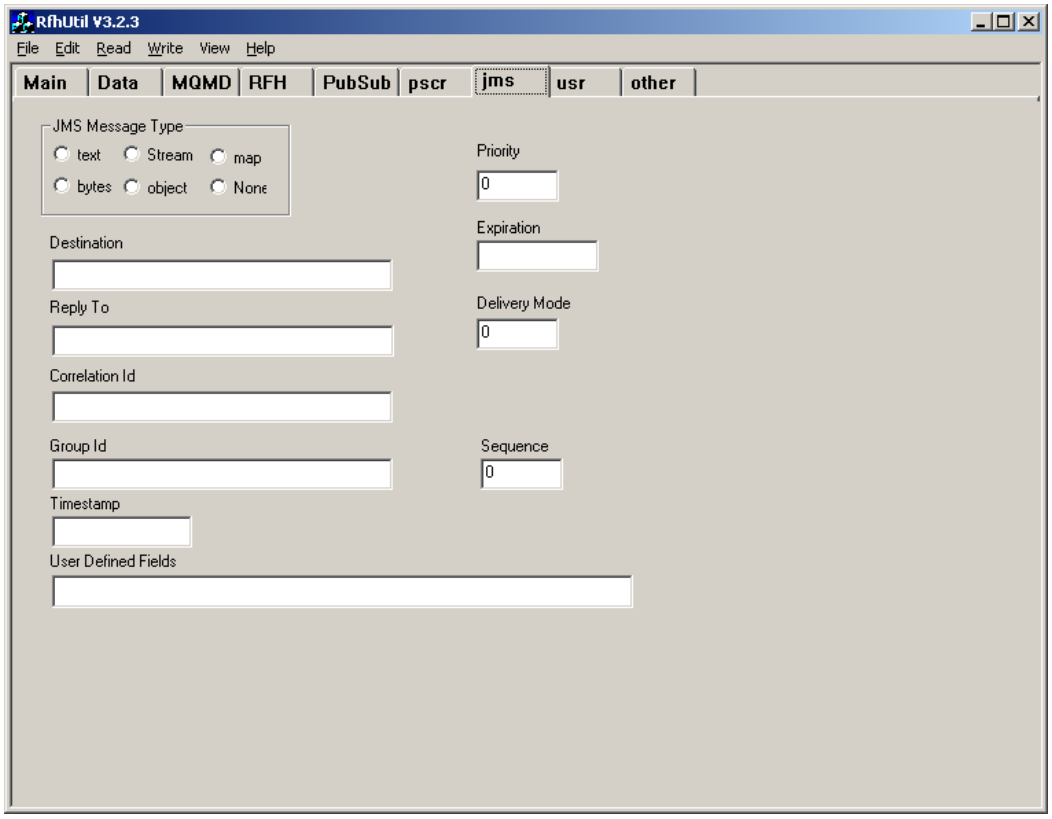
The *Error Id*, *Error Pos*, *Parm Id* and *User Id* fields are only used with a version 1 RFH. The *Other Info* field may contain fields that are not recognized in a version 1 RFH.

If any entries are found in the variable section of an RFH version 1 header that are not recognized by the program, the information will be added to the *Other Info* field and the Resp check box will be selected on the RFH tab. On output operations, the contents of this field will be appended to any version 1 header that is generated if the Resp check box is selected on the RFH tab.

Responses are generally sent by the broker and not created by an application.

jms tab

The jms tab is used to display and set fields within a jms folder in an RFH version 2 header.



The jms tab is used to display or enter the fields found in a jms folder in an RFH 2 header.

There appears to be considerable confusion about the JMS standard, which actually defines only an application program interface and does not say anything about wire formats or protocols. The use of a jms folder and an RFH 2 header is unique to the IBM Websphere MQSeries implementation of JMS, as delivered in SupportPac MA88 for MQSeries V5.2 and earlier, or integrated in WebSphere MQSeries V5.3.

JMS Message Type

There are five types of messages supported by JMS. They are:

- text
- byte
- stream
- map
- object

The message type refers to the format and layout of the data. For example, the user data in a text message is in XML format. Byte data is an arbitrary stream of bytes and can contain any other type of data. Object data contains java objects, which can only be interpreted by other JMS java programs.

The none option can also be selected, in which case the Msd (domain) parameter in the mcd folder will be set to a value of “jms_none”.

Destination

The destination contains the name of the target JNDI object. This object is mapped to an MQSeries queue using the JMSAdmin command, and is equivalent to the destination queue and queue manager names.

Reply To

The replyto variable contains the name of a JNDI object that the receiving program can use to send a reply message. This object is mapped to an MQSeries queue using the JMSAdmin command, and is equivalent to the MQSeries reply to queue and queue manager names.

Correlation Id

The correlation id is an arbitrary string that can be used by an application to match a reply with a corresponding request.

Group Id and Sequence

Group id and sequence numbers allow several messages to be sent and processed as a group. This is similar to the MQSeries group capability.

Timestamp

This field contains the JMS timestamp field. This field contains the time when the message was created.

Priority

This variable indicates the JMS message priority. It is similar to the MQSeries message priority.

Expiration

The expiration variable indicates if a message should be discarded after a certain time interval, and if so, what that time interval is.

Delivery mode

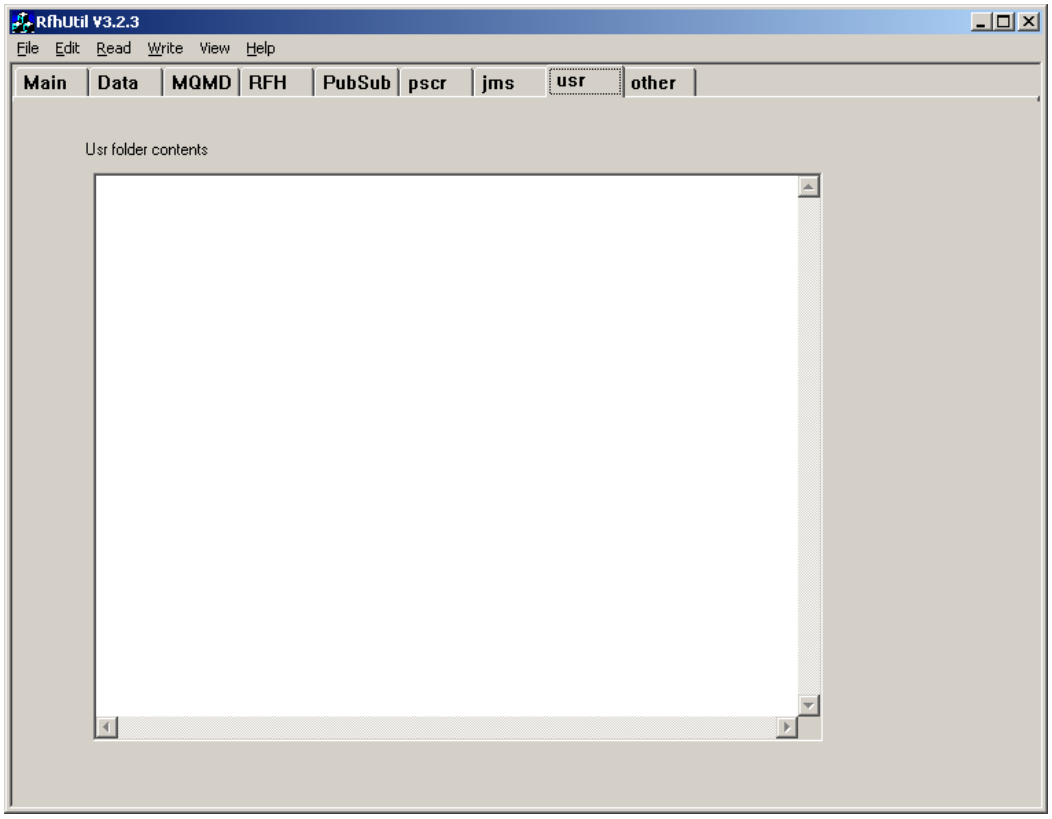
The delivery mode in JMS indicates if a message is to be persistent or not.

User Defined Fields

User defined fields are allowed in the jms folder. The *User Defined Fields* control should contain valid XML triplets (e.g. begin tag, data and matching end tag) that will be included in the jms folder. The contents of this field will be appended to the end of the XML stream that is generated from the defined fields and included in the jms folder. No checking of this data is performed on output.

usr tab

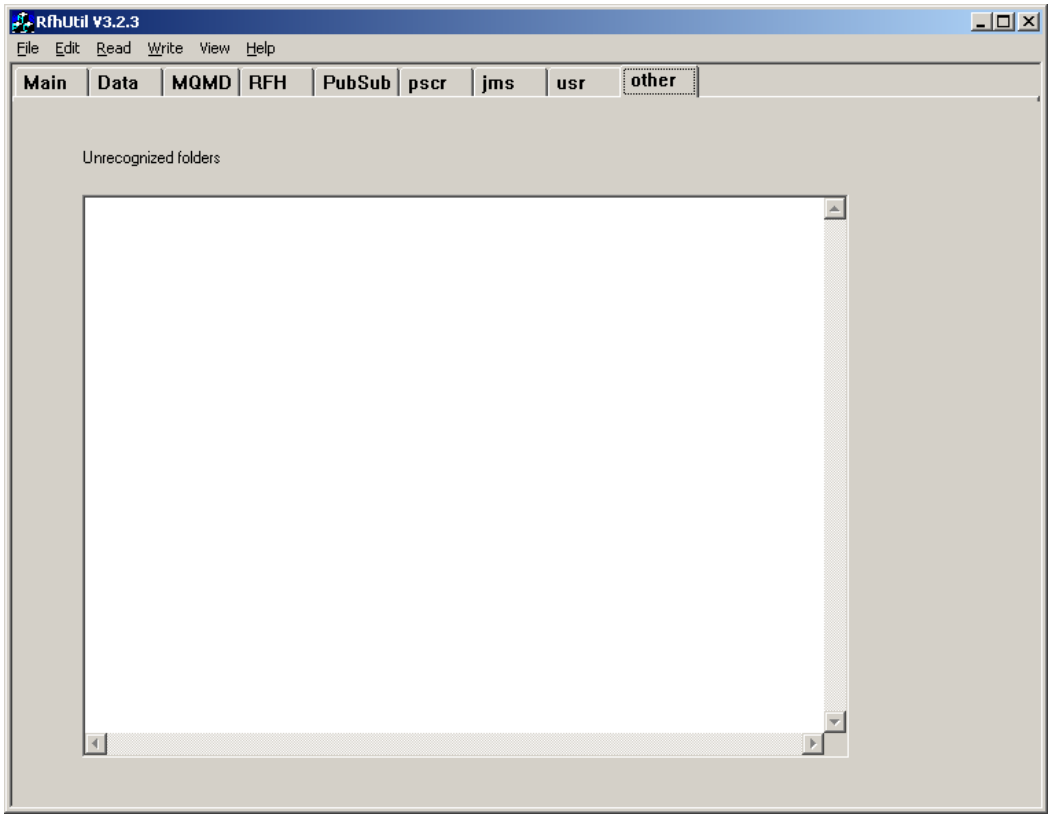
The usr tab is used to display and set the data in a usr folder in an RFH version 2 header.



The <usr> start tag and </usr> end tag are removed before the contents of the usr folder is displayed. The rest of the contents between these start and end tags is displayed in the accompanying edit box. The contents can be edited.

other tab

The other tab is used to display and set data in other folders that are not recognized by the RFHUtil program. The other tab can only display or set data for a single unrecognized folder. The data contents must be in XML format and must contain a single high-level root element.



The other tab is used for any folders that are not recognized as belonging to one of the other tabs. The RFHUtil program supports only one unrecognized folder. If more than one unrecognized folder is found in an input file or message, the last one encountered will be displayed.

The contents of this tab are displayed in an edit box. The data can be edited and changed as desired. No formatting is provided.

Chapter 4. Performance measurement utilities

Overview

Two command line performance measurement utilities are provided with this SupportPac. The first utility is a driver program that will read test data from one or more files and write messages to a specified queue. This will provide a measurement workload for one or more WMQI message flows. The second program analyses the results of the message flow and calculates the number of messages processed per second. Two additional versions of the driver program are provided. They are minor variations on the MQPUT2.exe program, and are necessary when remote queues or client connections must be used. The additional driver programs are called MQPUTS.exe and MQPUTSC.exe respectively. All of the driver programs use the same parameters file and message data files for their input.

In addition to the performance measurement utilities, two capture utilities are provided that help to capture messages in queues and store them as files. The first utility will read all the messages in a queue and write them to a single file, using a delimiter sequence between individual messages. The delimiter sequence is specified in a separate parameters file. The second utility will read a single message from a queue and will write the data to a file.

Please be sure to read the entire section entitled Requirements for performance measurement that immediately follows this section. This section details a requirement of the message flow itself that is absolutely necessary for the utilities to produce meaningful performance data.

Requirements for performance measurement

There are a few requirements that must be met to measure the performance of one or more IBM WebSphere MQ Integrator V2.1 message flows using the utilities provided with this SupportPac.

First, the user data from one or more test messages must be captured as one or more files. The test messages must be able to be replayed repeatedly and generate the desired results each time. The message test and display utility that is included with this SupportPac can be used to read such messages from a queue and store them as local files, provided the messages can be moved to a queue that is accessible from a 32-bit Windows workstation. These sorts of messages are usually required for testing and debugging of the message flows prior to any attempt to measure performance, so they are usually readily available.

The other key requirement is that each execution of the message flow produces at least one output message that is written to a queue. The throughput will be measured as the number of output messages per second, and will be determined by the number of messages in this particular queue. In some cases, a message flow may have to be modified to ensure that this requirement is met. Fortunately, most message flows meet this requirement.

One change is normally required to the message flow itself. This change is necessary to ensure that each message written to a queue that will be used to determine the throughput contains a new timestamp that indicates when the message was written. The message context field in the advanced properties of the MQOutput node for the queue must be changed to a value of “default” rather than the default value of “Pass All”. **This change is absolutely necessary. If the value is left as “Pass All” or some other setting, then the time the message is written will be passed from the input message, and the performance data will reflect the performance of the driver program rather than the performance of the message flow itself.** This change should be removed after the performance testing has been completed.

The change of the MQOutput parameter should not have a measurable impact to the performance of the message flow itself.

The author would like to mention one other important assumption that is usually made with regards to performance measurements. Performance measurements are usually measured in an optimised and controlled environment. In all cases, adequate real storage is assumed. For a full IBM WebSphere MQ Integrator V2.1 environment, this usually means 512 MB or more of main storage. Paging must be kept to a minimum. This is true for both the message flow environment itself as well as the performance driver utilities. If significant paging occurs during the measurement period, or if other unpredictable workloads are run on the system during the measurement period, the numbers are likely to be non-reproducible. Such results are usually regarded as useless. In most cases, the performance of IBM WebSphere MQ Integrator V2.1 message flows is CPU bound, while IBM WebSphere MQSeries itself may be either CPU bound (especially if non-persistent messages are used), or disk bound (if persistent messages are used). Like all assumptions, these assumptions should be validated.

It is recommended that that the task manager be invoked with the performance tab selected. This will display the current CPU and memory usage.

Using the Performance Utilities

To use the performance driver utility, a parameters file must be created. A sample parameters file is included with this SupportPac (Parmtst1.txt). This file contains many comments as well as examples of the various parameters. The parameters file can be edited with notepad or some other text editor. The parameters file contains the name of the queue that the input messages are to be written to. This should be the name of a queue that is referenced in an MQInput node within the message flow that is to be tested. It also must contain the name of one or more message data files, and the total number of messages that are to be written to the queue by the driver program. It is usually a good idea to try a small number of messages first, to make sure that everything is working as expected. The tune parameter should also be used when the parameters file is first tried, to help understand how to set the optimal queue depth and sleep time parameters, as described below.

Once the parameters and data files are available, and the message flow is deployed and ready to execute, the driver utility program (MQPUT2.exe) should be invoked from a command line prompt. The command syntax is as follows:

```
MQPUT2 –f parameter_file_name
```

The program accepts a single required parameter. The parameter is the name of the input parameters file. This file name must be fully qualified if the parameters file is not in the current directory. The MQPUT2 program will produce some command line output, which should be reviewed for error messages and to verify that the parameters used are in fact the ones that were intended. The queue and queue manager names in the parameters file can be overridden on the command line by using the –q and –m command line arguments.

After all the messages have been processed by the message flow, the results should be analyzed with the performance analysis utility program (MQTIMES.EXE). This utility should be executed from a command prompt as follows:

```
MQTIMES –q queue_name <-m queue_manager_name> <-b nnn>  >testout.txt
```

The performance analysis utility accepts one to three parameters. The first parameter is the name of the output queue that contains the output messages of the test message flow. The second parameter is the name of the queue manager that this queue is defined in. If the queue is defined on the default queue manager on the local system, then the second parameter can be omitted. The final parameter is the number of messages read within a single unit of work (batchsize). It helps the MQTimes program to run more quickly. The default of 25 is normally sufficient. The output will be written to the command line (stdout), and should be redirected to a file, so that the results can be saved. The results file can then be viewed with a text editor, such as notepad.

The MQTIMES utility will read all messages that are on the specified queue and will ignore all data in the messages except the timestamp in the MQSeries message descriptor. This program can also be used to drain queues of unwanted messages.

The MQTIMES utility will report the number of messages that were processed within each second on a separate output line. It will also report totals as well as the average number of messages per second for all intervals except the first and last. The first and last intervals are omitted since it is likely that neither one represents a full second.

The MQTimes2 program is the preferred performance analysis tool for measurement of MQSeries performance. The major difference between MQTimes and MQTimes2 is that MQTimes2 is designed to run while the driver program is operating. It reads messages from the output queue in real time. By consuming the messages as soon as they are available, overflows of the queue buffer to disk are minimized, and the size of periodic queue checkpoints, where the queue buffer is flushed to disk, are reduced significantly. MQTimes2 accepts some additional parameters. The format of the MQTimes2 command is as follows:

> MQTimes2 -c messageCount <-t timeout> -q queue <-m queue manager> <-d> <-b nnn> <-p PAN reply data file> <-n NAN reply data file>

The message count (-c) is the number of messages to read before automatically ending. The time out (-t) parameter is the maximum number of seconds to wait for a message to arrive before ending the program. The program will end when either it has read the expected number of messages or has waited more than the timeout interval for a message to arrive. The timeout interval must be at least 30 seconds.

The queue and queue manager names indicate which queue messages are to be read from. The queue name is required. The queue manager name is not required if the queue is defined on the default queue manager. The drain (-d) parameter indicates that any messages found in the queue when the MQTimes2 program is started are to be read and discarded. Any drained messages will not be included in the performance measurements.

The NAN and PAN reply data files are used when the consuming application must send report messages in response to a request for either positive or negative acknowledgments. Certain applications, such as JMS publishing applications, require responses from the broker to continue running. If a positive or negative acknowledgment is required, then a file containing the data portion of the response must be captured or created. This file must then be referenced with the appropriate command line argument (-p or –n). If a message requests both positive and negative acknowledgments, a positive acknowledgment will be sent.

Measuring performance of JMS publishing applications

Recent versions of the IBM JMS publishing client contains a form of pacing support. The idea is to prevent a publishing application from publishing messages faster than the broker can deal with them, and subsequently flooding a broker with messages. The implementation of this function involves setting of positive and negative acknowledgment report options periodically.

The actual implementation of this function is as follows. On the first message published, the PAN and NAN report options are set. Additional messages are then sent until the number of published messages reaches 20. On the first MQ commit after the 20th message has been published, the JMS client will first commit the unit of work, and will then perform an MQGet with the wait interval set to 30 seconds against the SYSTEM.JMS.REPORT.QUEUE. After the report message has been received, publication will resume and the cycle above is repeated. If the report message is not received within the wait interval, then the MQGet will complete with an error (reason 2033 – no message available) and an exception will be thrown indicating no response has been received from the broker.

The PAN responses normally sent by the broker must be sent by the measuring program (MQTimes2) to properly measure the performance of a JMS publishing application. In the case a jms publishing, a pscr reply message must be returned to the reply to queue name in the published message. This will be done if the positive acknowledgment file option (-p) is specified. An appropriate message can be captured and written to a file with the RFHUtil program (basically, publish a message with the PAN report option selected and the reply to queue name in the MQMD set to a user queue from which the reply can then be read and subsequently written to a file).

Conflicting options in the parameters file

It is possible to select conflicting options in the parameters file for the MQPUT2 program. For example, if the message type is set to 1 (request message) and the reply to queue or queue manager name is not specified, then the put requests will fail and the MQSeries error message will be displayed. Please correct the settings in the parameters file and rerun the performance driver program.

Capturing message data in files

Two utilities are provided with this SupportPac that will read messages in a queue and will write the data to a file. The utilities are mqcapture.exe and mqcapone.exe respectively. Both utilities use a parameter file to control the necessary input parameters. The mqcapture.exe utility will read all the messages in a queue and write them to a single file. The mqcapone.exe program will always read a single message from a queue and write it to a file.

The command syntax is as follows:

```
MQCAPTURE   parameter-file-name     data-file-name
MQCAPONE    parameter-file-name     data-file-name
```

The parameters file includes the name of the queue and queue manager that messages are to be read from. A parameter (MsgCount) can be specified in the parameters file to limit the number of messages that will be read by the MQCapture program. The delimiter sequence is also specified in the parameters file, either in ASCII (delimiter parameter) or in hexadecimal characters (delmiterx). The MsgCount and delimiter parameters are ignored by the mqcapone.exe utility.

A sample parameters file (parmcapt.txt) is provided with this SupportPac. This file should be modified to match the local configuration.

Driver memory usage

The MQPUT2 driver program reads all the messages that will be used in a performance run into memory prior to writing any messages to the designated queue. Therefore, sufficient memory must be available for not only the application(s) being driven but for the message driver program as well. If the driver program is forced to page its memory to and from disk, then the results obtained may not reflect the actual performance of the system.

Driving a system remotely

If messages are written to a remote queue, then the real driver of the application becomes the MQSeries message channel agent rather than the driver program itself. The driver program is merely writing messages to a transmission queue, and the message channel agent on the remote system is the program that is actually writing the test messages to the input queue.

Measuring performance of WebSphere MQ Integrator V2.1

The author's experience has been that the number of messages processed per second normally is very even and level, with only an occasional dip or other deviation. The author is usually able to calculate a message rate by just looking at the output. Select one or two ten or twenty-second intervals that look typical and then add up the number of messages processed and divide by the total number of seconds. The author has not seen the need to make lengthy runs, since the data generally seems very steady and unvarying.

Rather than measuring a variety of workloads concurrently, the author generally drives each message flow individually. The author also does separate measurements of different message sizes. This makes it easier to tell what the actual performance characteristics of each message size actually are and provides insights into how the throughput varies with different types of input. The author recommends that a variety of test cases be run, so that the throughput characteristics of each message flow are fully understood. The simplicity of the measurement process makes this an attractive option.

Before discussing the details of the performance utilities, some background will be provided about IBM WebSphere MQSeries and WebSphere MQ Integrator V2.1 performance.

MQSeries Performance considerations

Like any software system, MQSeries performance requires some detailed knowledge of certain aspects of the MQSeries queue manager, especially with regards to log usage and other disk operations. The following sections attempt to give a bare overview of the most common performance considerations that affect MQSeries performance. Like any performance discussion, there is no certainty that the performance of any individual application will behave in the manner described.

Persistent and non-persistent messages

MQSeries messaging supports two classes of messages, namely persistent and non-persistent. The performance characteristics of these message types are quite different. Persistent messages promise assured once and only once delivery and therefore must be logged to disk. Since they are logged to disk in all cases, the performance of MQSeries messages is often bounded by disk performance. Non-persistent messages do not require any disk usage, and the ultimate performance limitation is often some other resource limitation, such as CPU.

MQSeries log and queue disk operations

MQSeries performs disk operations to two primary types of disk, namely a log file and a queue data file. The log file is only used for persistent messages, so the log file has no effect on the performance of non-persistent messages. Persistent messages, however, must be written to the log before a unit of work can be completed for the messages.

When an application gets and puts persistent messages, no disk log activity is performed immediately if the message data will fit in the current log buffer. In MQSeries V5.1, the log buffer is 68K bytes if the default size of seventeen 4K pages is used. The buffer size can be increased to a maximum of 128K bytes (thirty-two 4K log pages). Each MQGet and checkpoint operation requires about 500 bytes of log space, and each MQPut operation requires about 500 bytes plus the length of the message and MQMD data. Thus, if a one-thousand byte message is read, then written and the application then issues a checkpoint, three log operations are involved and a total of approximately 3500 bytes of data are placed in the log buffer. Once an application issues a commit, however, all data that is currently stored in the log buffer must be written to disk before the application can continue. The log buffer may contain data from other applications as well, so the size of the buffer write may be considerably larger or the buffer may need to be written sooner, if it becomes full.

Once the log buffer becomes full or an application issues a commit, the log buffer must be written to disk. In MQSeries V5.1, a single log buffer cannot contain more than one commit. If other applications issue commits before the log buffer write is complete, then each one must wait in turn for the previous commit to finish. If the applications are writing small messages, the limiting factor in performance is usually the rate that the disk can perform individual writes to the disk. Each write requires the disk to spin around at least once. Since disks tend to spin at about the same speed (5400 and 7200 RPM are common), an individual dedicated disk drive generally cannot perform more than 75 writes per second. If the log disk is shared, then the number of log writes per second can be significantly reduced. If each application is performing a single MQSeries put per unit of work, and the message sizes are fairly small (typically 5000 bytes or less), then the limiting factor for persistent messages is usually the MQSeries log file.

As a result, the log file usually limits MQSeries message rates for a single queue manager to less than 100 messages per second. Using a solid-state disk can dramatically increase the performance (in some cases by a factor of greater than ten), but these disks tend to be extremely expensive. Certain disks with high-speed SSA connections and battery-backed fast write caches offer a more economically solution, with messages rates sometimes increasing by a factor of four. It must be emphasized that the above numbers have been observed in certain specific environments, and are not meant to be representative of what can

be achieved in any particular situation. However, they are useful for understanding what performance could be achieved with a single MQSeries queue manager and thus represent an upper bound.

Since non-persistent messages are not logged, there is no obvious constraint. Therefore, generalized performance statements of what is possible are more difficult. In many cases, the CPU is the limiting factor for non-persistent messages. Non-persistent messages can also benefit from larger memories and large buffers. In some cases the peak rate of an application using non-persistent messages has been seen to be ten times or more that of the same application using persistent messages.

In all cases, there is wide variation in the other resources and facilities that any individual application will be using, and therefore the degree to which MQSeries is a constraint or even a significant consideration in the performance of the application. The applications cited in the above examples were applications that primarily used MQSeries facilities. If an application is constrained by some other software or hardware considerations, then changes to MQSeries may have little or no effect on the overall performance of the application.

In addition to logging persistent messages, MQSeries queue managers also write a copy of each message that an application sends to a queue file. This operation uses what is called a “lazy write”. This means that the data is kept in a cache in memory. The cache is written to disk on an infrequent basis. In the case of MQSeries V5.1, the cache is flushed to disk (synchronized) once for every one thousand log operations, or in the event that the cache fills up. Every MQSeries get, put and commit is considered a log operation. For example, if an application issues one MQGet, one MQPut and one commit, then this activity is counted as three operations to the log. When the number of operations to the log from all applications reaches one thousand, then the entire buffer is written to the queue file. If a message has been written and read before the buffer is forced to the queue file, no record is ever written to the queue file for that message, even if the message is persistent.

If the buffer for the queue file overflows before messages are read, then many messages are forced out to disk, and then must be read back from the disk into memory when they are processed. This extra disk activity can have a significant impact on performance, especially if non-persistent messages are used. The driver program provided with this SupportPac is designed to minimize this effect. To avoid this situation, the driver program allows a minimum and a maximum queue depth to be specified. The program writes the maximum numbers of messages to the queue, and then sleeps for a specified period of time. When the program resumes execution after the sleep time expires, it checks the depth of the queue. If the queue depth is less than the minimum queue depth, it writes enough messages to the queue to bring the level back to the desired maximum queue depth. In this manner, messages are written at about the same rate that they are processed and extra queue buffer overflows avoided.

In many cases, the applications that create the input messages and process the output messages are on different systems. When calculating performance, the effects of MQSeries operations performed by message channel agents must also be taken into account. A message channel agent for a sender channel will issue MQGet commands from a transmission queue and send then down a channel. The channel agent will issue a commit when it has read a number of messages that matches the batch size (default of 50) or when the transmission queue becomes empty. Unless a channel is backed up for some reason, commits are usually issued more often than the batch size might indicate. In a similar manner, a receiver channel agent will read from a communications port and issue an MQPut to write the received messages to the appropriate queue (which could be a local queue or a transmission queue).

Changes in MQSeries Version 5.2

MQSeries version 5.2 contains some very significant performance improvements for both persistent and non-persistent messages. The performance of non-persistent messages has been improved by up to a factor of two, mostly by improving and optimising internal processing paths. The processing paths for persistent messages have also been improved, but the more significant improvements have resulted from significant changes in the log algorithms.

With all versions of MQSeries prior to V5.2 (with the exception of MQSeries on OS/390), a single log write was limited to a single commit. While data from other units of work might also be written to the disk, each commit required a separate log write. This behavior was changed in MQSeries V5.2. Assuming all applications are using units of work, then a log write will be started when the first application issues a commit. While that application is waiting for the corresponding disk write operation to complete, other applications can continue to use the log buffer for all operations, including commit operations. When the current disk write operation completes, the contents of the log buffer can then be written using a single disk write operation, even if many different applications have issued commits. Thus, a single log buffer write can be used to complete many units of work. The log buffer can also be larger (512K), so more work can be completed without the log buffer overflowing. MQSeries V5.2 has effectively done in software what solid state and fast disk caches did for the performance of earlier releases of MQSeries. There is still a performance benefit to be gained from solid state and cached disk, but the relative performance improvements are much less.

Finally, the queue data is forced to disk every 10,000 log operations rather than every 1,000 log operations as in V5.1 and earlier. This allows more messages to remain in memory until they are processed, rather than being written to disk and then read back in when they are processed.

The very significant performance improvements in MQSeries V5.2, including other improvements not mentioned, make it the preferred choice for any IBM WebSphere MQ Integrator V2.1 performance measurements.

Connections, disconnections and queue open/close

Although queue manager connections and the opening and closing of queues can be very significant in many applications, they do not tend to make much difference in the case of IBM WebSphere MQ Integrator V2.1, since the connections and open queue handles are maintained between message flows and reused by later flows. Queues are opened on first use and reused, and therefore do not generally play a significant role in. It should be noted that the performance of MQSeries connections and queue opens has been substantially improved in MQSeries V5.2. For further details, consult the performance reports available as SupportPacs.

Suggestions for performance testing of IBM WebSphere MQ Integrator

Now that some background on MQSeries performance has been given, some simplified recommendations will be given.

1) If the message rate is less than about 20 messages a second, then MQSeries performance is probably not critical. If the message rates are expected to be above 50 messages a second for persistent messages, or 200 messages a second for non-persistent messages, then careful tuning of MQSeries is probably necessary.
2) If persistent messages are being used, use a dedicated disk for MQSeries logging. If possible, use a disk with a SSA connection and a fast write cache. Increase the number of log pages to 32. Message rates above 75 will require special hard for the MQSeries log disk (fast write cache or solid state). Multiple queue managers and therefore multiple brokers may also be necessary.
3) If the primary interest is in the performance of IBM WebSphere MQ Integrator V2.1 message flows, use non-persistent messages even if the application will use persistent messages in a production environment. This tends to limit the impact that MQSeries itself will have on the measurements.
4) As a rule of thumb, IBM WebSphere MQ Integrator V2.1 message flows tend to be CPU bound. A faster CPU or more CPUs will generally increase proportionally to the relative CPU power, assuming that no base MQSeries or other software constraints limit the performance.
5) Use MQSeries V5.2, for the reasons discussed below.

6) Ensure that enough memory is available on the test system for both the application to be tested and the MQPUT2 driver program.

Format of the parameters file for the MQPUT2 utility

The MQPUT2 workload performance utility for IBM WebSphere MQ Integrator V2.1 uses a text file to specify the values of a number of parameters. These parameters include the name of the queue and queue manager that messages are to be written to, the name of the file(s) that contains the message data, timing parameters and values for fields in the MQSeries message descriptor and Rules and Formatting Header (RFH) fields. The parameter names can be any combination of upper and lower case letters. Trailing blanks in the parameter values are truncated. Parameter values can be enclosed within double quotation marks (“”), in which case the value will be assumed to be between the leading and trailing quotation marks.

Blank lines, as well as any line that begins with an asterisk (“*”), semi-colon (“;”) or a number sign (“#”), are considered comments. A fully commented sample parameters file is provided.

The parameters file is divided into sections of two types. The section headers are enclosed in square brackets (“[]”). The two section types (including the square brackets) are:

- [HEADER]
- [FILELIST]

The initial section header for the HEADER section can be omitted. The section header for the file list is required.

The following fields are used to set the number of messages to be written to the queue, the queue to write to and to control the rate that the messages are written to the queue.

- Qmgr Name of the local queue manager to connect to
- Qname Name of the queue to write messages to
- RemoteQM If the queue is remote, name of remote QM
- Msgcount Total number of messages to write
- Sleeptime Time in milliseconds to wait before checking queue depth
- Qdepth Queue depth at which messages will be written
- Qmax Maximum queue depth
- Batchsize Maximum number of messages to write in a single unit of work
- Tune Indicator whether to adjust sleeptime automatically and report

The following fields in the message descriptor can be set:

- ReplyToQ Reply to queue name
- ReplyToQM Reply to queue manager
- Format Format of the user data
- Priority Message priority
- Persist Message persistence
- Msgtype Message type (e.g. reply, request, datagram, report, etc)
- Codepage Code page of the RFH if present or user data if no RFH
- Encoding Encoding type of the RFH if present or the user data if no RFH
- Correlid Correlation id (specified in ASCII characters)
- Correlidx Correlation id (specified in hex characters)
- Groupid Group id (specified in ASCII characters)
- Groupidx Group id (specified in hex characters)

The following fields can be used to set the Rules and Formatting Header (RFH):

- RFH Indicator if Rules and Formatting Headers are to be used
- RFH_CCSID Code page of the user data
- RFH_Encoding Encoding of the user data
- RFH_DOMAIN Messasge domain (V2 only)
- RFH_MSG_SET Message set (V2 only)
- RFH_MSG_TYPE Message type (V2 only)

- RFH_MSG_FMT Message fmt (V2 only)
- RFH_APP_GROUP Application group (V1 only)
- RFH_FORMAT Message format (V1 only)

The following fields can be used to set fields in the publish and subscribe folder (psc) in an RFH2 message header.

- RFH_PSC_REQTYPE Request type
- RFH_PSC_TOPIC1 Topic (up to 4 topics can be specified)
- RFH_PSC_TOPIC2 Topic (up to 4 topics can be specified)
- RFH_PSC_TOPIC3 Topic (up to 4 topics can be specified)
- RFH_PSC_TOPIC4 Topic (up to 4 topics can be specified)
- RFH_PSC_SUBPOINT Subscription point
- RFH_PSC_FILTER Filter
- RFH_PSC_REPLYQM Publish/subscribe reply to queue manager
- RFH_PSC_REPLYQ Publish/subscribe reply to queue
- RFH_PSC_PUBTIME Publication time
- RFH_PSC_SEQNO Sequence number
- RFH_PSC_LOCAL Local only
- RFH_PSC_NEWONLY New only
- RFH_PSC_OTHERONLY Other only
- RFH_PSC_ONDEMAND On demand (will use request publication)
- RFH_PSC_RETAINPUB Retained publication (set by publisher when publishing message)
- RFH_PSC_ISRETAINPUB This is a retained publication (set by broker)
- RFH_PSC_CORRELID Include correlation id in published messages
- RFH_PSC_DEREGALL De-register all (unsubscribe only)
- RFH_PSC_INFRETAIN Inform if retained publicaton

The request type must be one of the five valid publish and subscribe command types. The valid command types are:

- Subscribe “Subscribe”, “Sub” or “1”
- Unsubscribe “Unsubscribe”, “Unsub” or “2”
- Publish “Publish”, “Pub”, or “3”
- Request publication “ReqPub” or “4”
- Delete publication “DelPub” or “5”

The publish and subscribe options (RFH_PSC_LOCAL through RFH_PSC_INFRETAIN) are Boolean values and should be specified as either “Y” or “1” for yes and “N” or “0” for no. The default is no.

The following fields can be used to set fields in the jms folder in an RFH2 message header.

- RFH_JMS_REQTYPE JMS data type
- RFH_JMS_DEST Destination object name
- RFH_JMS_REPLY Reply to object name
- RFH_JMS_CORRELID Correlation id (JMS)
- RFH_JMS_GROUPID Group id (JMS)
- RFH_JMS_PRIORITY Message priority (JMS)
- RFH_JMS_EXPIRE Expiration time (JMS)
- RFH_JMS_DELMODE Delivery mode
- RFH_JMS_SEQ Sequence number (JMS)

The JMS data type must be one of the following values”:

- text “text” or “1”
- bytes “bytes” or “2”

- stream “stream” or “3”
- object “object” or “4”
- map “map” or “5”
- none “none” or “6”

Using more than one message data file

Support is provided for more than one message data file in a single parameters file. The driver utility program will process as many data file names as are found in the parameters file. The entire contents of each data file are read into memory during program initialization and are held in memory for the duration of the program execution. There are several important considerations.

The driver utility program will write one message of each type (file), in the sequence that the files are specified, and will then return to the first message specified in the parameters file. This process will be repeated until the number of messages written is equal to the number specified in the message count parameter. If the message count is not a multiple of the number of data files, then more of some message types written than other message types. The same file name can be specified more than once, but each instance of the file will be considered to be a different file and the file data will be read into memory more than once.

The MQSeries message descriptor and Rules and Formatting header fields for the file are determined by those specified in the parameters file at the time the message data file is read into memory.

In all cases, the messages will be written to a single queue. The queue and queue manager names for the output queue are those in effect when the entire parameters file has been read and processed. If the queue manager or queue names are included more than once, then the later settings will overwrite the earlier settings.

Using different parameters for message data files

When using more than one message data file in a parameter file, it is possible to use different values for the message descriptor and Rules and Formatting header fields. This is accomplished by using more than one header and more than one filelist section in the parameters file. The values for the first file are specified in the first header section. These settings are then saved with the file data. A second header section and a second filelist section then follow the first filelist section. The parameters in the second header section then modify the values in the MQSeries message descriptor and/or Rules and Formatting header. The new or changed values are then used for the files in the second filelist. This process can be repeated as many times as necessary, using a different header for each data file if necessary.

Multiple messages in a single data file

Each input data file used by the performance driver program can contain one or more messages. If a file contains more than one message, then a delimiter sequence must be inserted between each pair of messages. No delimiter is needed at the beginning or end of the file. The delimiter sequence must be specified in the parameters file using the delimiter or delimiterx file.

Handling of RFH and RFH2 headers

There are three basic options for the use of Rules and Formatting headers (RFH) by the performance driver utility. The first option is “No”, which indicates that no Rules and Formatting headers are to be used. The second option is “Auto” (automatic). This option indicates that the program should look for a Rules and Formatting header at the front of the message data file. If it finds what appears to be a valid Rules and Formatting header at the beginning of the data file, then the settings for the Rules and Formatting header are taken from the message data file. The RFH_CCSID and RFH_ENCODING parameters should still be set in the parameters file. The final option is to add a header based on parameters set in

the parameters file. In this case, the appropriate fields in the parameters file should be set to the desired values, as described below.

Overall handling of Rules and Formatting headers (RFH) is controlled by the RFH parameter in the parameters files. RFH headers can be stored with the message data in a message data file or can be added based on values provided in the parameters file.

The RFH parameter can be set to one of five values. If the RFH parameter is set to “N” or “No”, then no RFH headers will be used. It the RFH parameter begins with an “A” (such as “A”, “Auto” or “Automatic”), then the program will look for an RFH at the beginning of the message data file. If the first four characters are “RFH “ (in either ASCII or EBCDIC) and the next four characters are either a binary 1 or a binary 2 (in either big-endian or little-endian format), then it is assumed that the file contains an RFH. If the parameter begins with an “X” (such as “XML” or “X”), then a special version of the RFH will be built at the beginning of the messages, to indicate that the messages are XML messages. If the parameter is set to “V2” (or “2”) or “V1” (or “1”), then a version 2 or version 1 RFH will be inserted at the beginning of the message. The individual parameters will be taken from values set in the parameters file. This parameter can be abbreviated to a single character, which can consist of “N” for No, “X” for XML, “A” for Automatic, “2” for version 2 and “1” for version 1.

RFH parameters

If the RFH parameter is set to “V1” or “1”, then the following fields in the parameters file are used to build the RFH:

- RFH_APP_GROUP
- RFH_MESSAGE_SET
- RFH_CCSID
- RFH_ENCODING

If the RFH parameter is set to “V1” or “2”, then the following fields in the parameters file are used to build the RFH:

- RFH_DOMAIN
- RFH_MESSAGE_SET
- RFH_MESSAGE_TYPE
- RFH_MESSAGE_FMT
- RFH_NAME_CCSID
- RFH_ENCODING
- RFH_NAME_CCSID

If the RFH parameter is set to XML or Auto, the code page and encoding parameters for the RFH are used. If the RFH parameter is set to No, the other RFH parameters are ignored.

General parameters

Certain input parameters are global in nature and should only be specified once in the parameters file. These parameters are as follows:

- Output queue and queue manager name
- Total number of messages to be written
- Time to sleep before checking the depth of the output queue
- Whether dynamic tuning of sleep time is enabled
- Desired minimum and maximum queue depths
- Maximum number of messages to write in a single unit of work

The queue manager (QMGR) name is the name of the queue manager to connect to. This should normally be the queue manager that the queue is defined on. This is necessary if the

normal version of the driver utility is used, since the program does inquiries to determine the current depth of the queue and these inquiries are not supported for remote queue definitions.

If the queue exists on a remote queue manager, then several factors must be taken into account. First, the queue can be accessed either directly by explicitly giving the name of the queue manager that the queue is defined on, or by creating a remote queue definition on the local queue manager. Although a version of the utility is provided that does not use the inquiry API to determine the queue depth, the author does not recommend this approach. The problem is that the driver program for the workload has in effect become the message channel agent on the remote system. While this may be what is desired, a detailed understanding of the inner workings of the message channel agents may be required to understand exactly what is being measured.

The minimum and maximum queue depth parameters (qdepth and qmax) are used by the normal version of the driver utility to attempt to maintain a relatively steady number of messages in the queue. The driver utility will sleep for a specified period and will then wake up and check the depth of the queue. If the depth is below the minimum (qdepth) value, then the program will write calculate the difference between the current queue depth and the maximum desired queue depth (qmax) and write this number of messages to the queue. It will then sleep again for the specified period. If the number of messages written is less than the batch size parameter, all the messages written will be in a single unit of work. Otherwise, more than one unit of work is used.

The sleep time parameter is the number of milliseconds that the program should sleep before it checks the depth of the queue. The sleep time parameter should be set to an amount that ensures the queue depth will never reach zero, but large enough so that the program does not wake up and check the queue depth more often than is necessary. The minimum value for this parameter is 10 (.01 seconds) and the maximum is 10000 (10 seconds). This parameter can be adjusted dynamically if the tune parameter is set to a one (“1”). The author recommends using the tune parameter to modify the sleep time parameter to find an optimal setting. The optimal setting allows the qdepth to drop to the desired minimum level in one or two sleep time intervals. For example, if the desired minimum queue depth is 20 and the desired maximum queue is 40, and the workload is executing at 40 messages a second, then the optimal sleep time is approximately 500 milliseconds.

If dynamic tuning is requested, the following tuning algorithm is used. The sleep time interval is increased and/or decreased after each sleep time interval until an optimal value is found. The algorithm is fairly simple. If the depth of the queue has not changed since the previous time interval, the sleep time interval will be increased by a factor of two plus one millisecond. If the queue depth is less than the amount at the beginning of the previous interval but still more than the minimum queue depth, then the sleep interval is increased by one eighth of the current value plus one millisecond. If the queue depth is zero, the sleep time interval is divided by two. In all cases, the new sleep interval cannot be less than ten milliseconds or greater than 10000 milliseconds (10 seconds). The purpose of this algorithm is to quickly find a value that is a reasonable compromise between waiting too long and having the queue become empty and checking the queue depth more often than necessary and incurring unnecessary overhead. A large number of console messages are produced if the tuning option is selected, to help to understand the optimal setting.

The author recommends that the minimum and maximum queue depth parameters and the sleep time parameter be modified and tried with the tuning parameter turned on. After the approximate optimal values for these parameters are understood, then the tuning parameter should be turned off and the desired values set in the parameters file.

In all cases, an error message will be generated if the queue depth is found to be zero after any time interval. Any runs in which the queue depth falls to zero should be discarded, since they are not measuring maximum throughput.

Tuning parameters and functions for the Performance Driver Utility

If the TUNE parameter is set to 1 in the parameters file, the program will dynamically adjust the sleep time to try to match the rate at which messages are being processed. The following algorithms are used.

If the queue depth reaches zero, the sleep time parameter is divided by two and an error message is generated. If the qdepth has not changed since the last time the sleep time interval expired, the sleep time interval will be increased by one-half of the current value plus one millisecond. If the qdepth has decreased but the value has not reached the desired value, then the sleep time interval will be increased by one-eighth of the current value plus one millisecond. If the qdepth is less than the desired value the interval will be decreased by one-eighth of the current value plus one. In no case, will the new value be less than ten milliseconds nor more than ten seconds. Every time the value is changed, a message will be written to stdout. Console messages are also written to stdout each time MQSeries messages are added to the queue. Tuning messages will not be written to the console if the TUNE parameter is turned off (set to zero).

The intent of the tune parameter is to find an optimal value for the sleep interval. Once that value has been determined, then the value should be set in the parameters file and automatic tuning is then turned off for the actual performance measurement runs.

MQMD fields

Certain fields in the MQSeries message descriptor can be set in the parameters file. All of these parameters are optional. If these fields are set in the parameters file, the values will be used for all messages that are written to the output queue. The fields that can be set are:

- Format
- Persistence
- Priority
- Reply to Queue and Queue Manager
- Correlation Id (either character or hex)
- Group Id (either character or hex)
- Message Type
- Expiry

If the above fields are set in the parameters file, the values in the parameters file will be used for all messages that are generated by the driver utility.

The format field can be up to eight characters long, and will be padded with blanks on the right. The persistence field can be 0 for non-persistent, 1 for persistent or 2 for persistence as defined by the queue. Message priority can be any value greater than or equal to zero. If the priority is set to zero, then default priority is assumed. The Reply To Queue and Queue Manager fields can be up to 48 characters in length. The message expiry can be specified as a positive integer to create messages with a definite expiration time. If the expiry parameter is set to zero, then the messages will have no expiration time.

The message type paramater (MSGTYPE) is specified as a number. The following values are allowed for this parameter:

- Request 1
- Reply 2
- Report 4
- Datagram 8
- MQ/E fields from MQ/E 112
- MQ/E fields 113
- User 65536 to 999,999,999

The message type should be specified as the number that corresponds to the desired message type.

The correlation and group id fields can be specified as either ASCII text or as a hexadecimal string. Each field has two different keywords that indicate if the corresponding field is being specified as ASCII characters or as hexadecimal characters.

Format of the parameters file for the data capture utilities

The data captures utilities (MQCapture and MQCapone) also use a simple parameters file to specify certain attributes, such as the name of the queue and queue manager that messages are to be read from. A sample parameters file (parmcapt.txt) is provided with this SupportPac. Both utilities can use the same parameters file (the MQCapone utility will ignore the message count and delimiter parameters).

The following fields are used to set the maximum number of messages to be read from the queue, the queue to read from, and whether or not RFH headers are to be stripped off or saved in the file with the message data.

- Qmgr Name of the local queue manager to connect to
- Qname Name of the queue to write messages to
- StripRFH Indicates if RFH headers are to be removed
- MsgCount Maximum number of messages read from the queue
- Delimiter Delimiter sting specified in character (ASCII) format
- Delimiterx Delimiter string specified in hexadecimal characters

The Qmgr and Qname parameters specify the name of the queue manager and queue that messages are to be read from. The StripRFH parameter indicates if RFH headers are to be removed from messages or if they are to be left at the front of the data and written to the file. The MsgCount parameter indicates the maximum number of messages that will be read from the queue and written to the file. Only one of the delimiter and delimiterx parameters should be specified in a given parameters file. The MsgCount and delimiter/delimiterx parameters are ignored by the MQCapone utility, since this version of the utility reads only a single message from the input queue.

The delimiter string is used when more than one message is written to a file. The delimiter string is placed between messages so that the MQPUT2 utility or other programs that might use the file can tell where one message ends and the next message starts. The delimiter can be specified as ASCII characters (delimiter parameter) or hexadecimal characters (delimiterx parameter). The delimiter string can be from 1 to 512 bytes in length. It must be a sequence of characters that never occurs in the messages that are saved in the file. The delimiter can be a single character that never occurs in the message data (such as a binary zero if the messages contain only text data) or it can be a longer string that will not occur in normal user message data. If the sequence contains more than one character, then the sequence should contain more than one character value (e.g. it is probably better to use a sequence like “#%#” rather than “###”, so that a message ending with a “#” character will be processed properly). The default delimiter sequence is "#@#@#" (without the double quotes).

If multiple messages are captured in a single file using the MQCapture program, and they are to be replayed using the MQPUT2 utility, the delimiter or delimiterx parameter specified in the parameters file for the MQPUT2 utility should be set to match the delimiter that was used to capture the messages.

Use on systems other than Windows NT/2000, Solaris and AIX

The performance utilities have been written and tested on the Windows 2000, Sun Solaris and AIX platforms. Full source code, including the Microsoft Visual C++ V6 project files, is provided for the performance utilities. The programs are written in standard ANSI C. The utilities have been ported to the AIX and Sun Solaris environment by the author and the necessary changes have been included as conditional compilation statements. The author believes the programs could be easily ported to other Unix environments as well, but has not

done any testing to validate this. The author will be glad to assist any users on a best efforts basis who attempt to do this port, but cannot guarantee results. Any such assistance may also be limited by the lack of a suitable test environment for the author to use, and by the normal job demands of the author.

To compile the programs in a different environment, the C source programs for the capture, driver and performance analysis programs (mqcapture.c, mqcapone.c, mqput2.c and mqtimes.c respectively), located in *source* subdirectory, must be transferred to the target Unix system. A simple make file is provided (Makefile), which can also be uploaded. This can be done using FTP or some other similar transfer scheme, using the text or ASCII option.

The author used the following command line input to compile the programs in an AIX environment. Please note that command syntax and other environmental considerations may be different on another system.

```
/usr/vac/bin/cc mqcapture.c -l mqm -L /usr/mqm/lib -o mqcapture
/usr/vac/bin/cc mqcapone.c -l mqm -L /usr/mqm/lib -o mqcapone
/usr/vac/bin/cc mqtimes.c -l mqm -L /usr/mqm/lib -o mqtimes
/usr/vac/bin/cc mqtimes2.c -l mqm -L /usr/mqm/lib -o mqtimes2
/usr/vac/bin/cc mqput2.c  -l mqm -L /usr/mqm/lib -o mqput2
```

The IBM VisualAge C compiler was used in the example above. These commands can be used as an alternative to the make file.

The author has used the following command line input to compile the programs in a Sun Solaris environment.

```
export LIB=/opt/mqm/lib
/opt/SUNWspro/bin/CC -DSOLARIS –o mqcapture mqcapture.c –lmqm –lmqmcs –lmqmzse –lsocket –lnsl –ldl
/opt/SUNWspro/bin/CC -DSOLARIS –o mqcapone mqcapone.c –lmqm –lmqmcs –lmqmzse –lsocket –lnsl –ldl
/opt/SUNWspro/bin/CC -DSOLARIS –o mqtimes mqtimes.c –lmqm –lmqmcs –lmqmzse –lsocket –lnsl –ldl
/opt/SUNWspro/bin/CC -DSOLARIS –o mqtimes2 mqtimes2.c –lmqm –lmqmcs –lmqmzse –lsocket –lnsl –ldl
/opt/SUNWspro/bin/CC -DSOLARIS –o mqput2 mqput2.c –lmqm –lmqmcs –lmqmzse –lsocket –lnsl –ldl
```

Compiled binaries for the mqcapture, mqcapone, mqput2 and mqtimes2 utilities on AIX and Sun Solaris are provided. These files should be uploaded using FTP or some similar transfer scheme, using the binary option.

Chapter 5. Miscellaneous utilities

Two command line utilities for the Windows environment are provided with the SupportPac.

The first utility can be used to display all queue managers defined on a local system, either directly as queue managers or accessible via MQSeries clients. This utility accepts one command line argument. To use the utility, start a DOS command window and type findmqm and press enter.

```
>findmqm
find MQSeries Queue Managers V1.00

Local Queue Managers defined

 MQSI
 TEST
 WMQI.qm

MQSERVER environment variable
 MQSERVER environment variable is not set

client channel table (amqclchl.tab)
Opening channel table C:\Program Files\MQSeries\amqclchl.tab

Channel Name         Type   Max Msg Connection Name             Queue Manager Name

SYSTEM.DEF.CLNTCONN  TCP   04194304
TESTREM.CLIENT       TCP   04194304 MACNAIR2(1414)             TESTREM
WMQI21.CLIENT        TCP   04194304 MACNAIR2(1415)             WMQI21
```

If a –q argument is included on the command line, then the utility will list all queues found for all local queue managers.

The second utility can be used to display and verify the entries in the classpath, path and lib environment variables. The entries in the classpath environment variable will always be checked. This means that each entry will be confirmed to be either a valid file name or directory name. In addition, the utility will check for duplicate entries. There are three additional optional command line arguments that will perform additional checking, as follows:

- `-p or –P` Verify the path as well as the classpath
- `-l or –L` Verify the lib as well as the classpath
- `-j or –J` Verify that the required JMS files are present in the classpath

The utility will verify that each file or directory in the classpath exists, and will check for duplicate entries. If the jms switch is set, the utility will verify that the files required for JMS support (SupportPac MA88) are included in the classpath. The path (-p) and lib (-l) switches will verify that each file or directory in the path and lib environment settings exists.

Chapter 6. Reporting problems or suggestions for improvement

Although no official support is provided, the author is interested in hearing of any problems or suggestions for improvement for this SupportPac. If a bug is suspected, please send an email with a problem description. If possible, please attach a file with a copy of the message so that the author can reproduce the problem locally. The author's email address is on the front cover of this document.

Appendix A. Sample parameters file for MQPUT2 utility

```
[header]
* Input parameters for MQPut2 program *
*
* name of the queue and queue manager
* to write messages to
*
qname=TEMP.OUT
qmgr=MQSI
*
* total number of messages to be written
* the program will stop after this number of
* messages has been written
*
msgcount=300
*
* desired qdepth for input queue
* the program will write messages until the queue depth
* is equal to the qmax parameter.  When the queue depth
* reaches the desired depth, more messages will be written
* to bring the depth back to twice the desired value.
*
qdepth=15
qmax=30
*
* number of milliseconds to delay before checking
* the queue depth
*
sleeptime=50
*
* whether to dynamically adjust the sleeptime or not and whether
* to write a line of output each time messages are written to the queue
* tune=0          do not dynamically adjust sleeptime
* tune=1          adjust sleeptime dynamically and report changes
*
tune=0
*
* maximum number of messages to write in a single unit of work
*
batchsize=20
*
* MQMD format field
*
*format= "MQSTR   "
*
* Correlation id specified in ASCII
*
*correlid="Performance Utility Test  "
*
* Correlation id specified in Hex
*
*correlidx=20313233343536000037381a1b1c00
*
* Message priority
*
priority=2
```

```
*
* group id specified in ASCII
*
*groupid=group 1
*
* group id specified in hex
*
*groupidx=c1c2f1f2
*
* reply to Queue manager
*
replyqm=TEST
*
* reply to queue name
*
replyq=TEST.REMOTE
*
* message persistence
* persist no = 0
* persist yes = 1
* persist as queue def = 2
*
persist=0
*
* message type
*
* Allowed values for message type
*   1 - request
*   2 - reply
*   4 - report
*   8 - datagram
*   112 - MQE fields from MQE
*   113 - MQE fields
*   65536 to 999,999,999 - user
msgtype=1
*
* encoding for user data
* encoding=546 for PC integers, etc
* encoding=785 for host integers, etc
*
encoding=546
*
* code page for user data
* codepage=437 for PC characters
* codepage=500 for host characters
*
codepage=437
*
* the character sequence that separates messages in a file
*   message delimiter parameters in ascii or hex
* only one or the other should be specified
*
* delimiter="#$%^&"
* delimiterx="0D0A"
*
* rfh usage
* rfh = N for No rfh
* rfh = A for Automatic (look for RFH at beginning of data file)
* rfh = 1 or V1 for Version 1 rfh from parameters in parm file
* rfh = 2 or V2 for Version 2 rfh from parameters in parm file
```

```
* rfh = X for special V2 rfh with fixed portion only and format=xml
*
* only first character checked, except for V when second character is also checked
*
rfh=1
*
* rfh data parameters - only used if rfh=1 or rfh=2
*  ignored if rfh=N, rfh=A or rfh=X
*
* both V1 and V2
*  encoding should be 546 for PC and 785 for host 390
RFH_CCSID=500
RFH_ENCODING=785
* V2 only
RFH_NAME_CCSID=1208
RFH_DOMAIN=MRM
RFH_MSG_SET=DH4M6DO072001
RFH_MSG_TYPE=Customer_Root
RFH_MSG_FMT=CWF
* V1 only
RFH_APP_GROUP=Customer_Msgs
RFH_FORMAT=Customer_Root
*
*
* END OF PARAMETERS SECTION
*
* beginning of list of message data files
* each line must contain a fully qualified file name
* each file will provide the data for a separate message
* the messages will be written in the order given below
* only file names may follow the [filelist] entry
*
[filelist]
msg1.dat
msg2.dat

[header]
*
* look for rfh with file data
rfh=auto

[filelist]
c:\v2test\rfhtest\msg3.dat
```

Appendix B. Sample parameters file for MQCapture utility

```
*
* name of the queue and queue manager
* to read messages from
*
qname=TEMP.OUT
qmgr=MQSI
*
* total number of messages to be written
* the program will stop after this number of
* messages has been written
*
MsgCount=4
*
* delimiter and delimiterx are used to
* define the message seperator sequence.
*
delimiter="#@%@#"
*delimiterx="0D0A"
*
* striprfh parameter determines if RFH
* headers are to be written to the
* data file along with the message
* data or removed from the message
*
striprfh=Y
```

Appendix C: Exploring Publish and Subscribe

This appendix provides a scripted tutorial to demonstrate the facilities and features of publish and subscribe features found in WebSphere MQ Integrator V2.1 (WMQI). It uses publish and subscribe features of the RFHUtil utility program provided with this SupportPac. The intent of this tutorial is to help people to become familiar with publish and subscribe features provided with WMQI. This tutorial does not cover all features and the reader is encouraged to further explore other options and features not covered by this tutorial.

The exercises are designed for WebSphere MQ Integrator V2.1 (WMQI) and WebSphere MQ Integrator Broker V2.1. Most of the exercises can be used with WebSphere MQ Event Broker V2.1 (the exercise demonstrating the use of subscription points will not work but the other exercises should work).

Setup

Prior to running any of the examples described below, the necessary components must be started. A simple message flow must also be created and deployed to an execution group on the chosen broker.

1. Start the WMQI configuration manager and broker. The following commands can be used (<broker name> should be replaced with the name of the WMQI broker). If the event broker product is being used, please use the wmqpsstart command instead.
   - mqsistart configmgr
   - mqsistart <broker name>
2. Start the WMQI control center (select Start->Program Files->IBM WebSphere MQSeries Integrator V2.1->Control Center). If using event broker, please start its control center instead.
3. Start two RFHUtil sessions. If the local system is using an MQ Client connection instead of a local queue manager, then start two RFHUtilc sessions instead. One session will be used to send messages and the other to receive messages
4. Start an MQExplorer session. The MQExplorer session will be used to examine the contents of the queues.

Create publication message flow and required queues

The following steps can be used to create the message flow that is necessary to publish messages. The flow itself is very simple and basic and does no processing or transformation of the messages it publishes, although this need not be the case. The message flow can perform whatever transformation and logic is necessary prior to publishing the message.

1. Using MQExplorer, create five local queues on the broker queue manager. Name them PUBLISH.IN, PUBLISH.OUT, PUBLISH.REPLY, PUBLISH.RETAIN and FAILURE respectively.
2. Using the Message Flows tab in the control center, select Create->Message Flow from the popup menu that appears when the message flows item is selected and the right mouse button is clicked.
3. Call the message flow PublishMsg and press finish to create the message flow.
4. Expand the menu in the left hand pane by clicking the plus sign next to IBM Primitives.
5. Drag and drop an MQInput node, a Publication node and an MQOutput node onto the right hand pane.
6. Select the MQInput node, click the right mouse button and select properties. Select the basic tab and set the queue name to PUBLISH.IN. Select the Default tab and set the Message Domain to XML. Click Ok. It is good practice to rename the node to match the queue name (PUBLISH.IN).

7. Select the MQOutput node, click the right mouse button and select properties. Select the basic tab and set the queue name to FAILURE. Click Ok. It is a good practice to rename the node to match the queue name (FAILURE).
8. Wire the out terminal of the MQInput node to the input terminal of the publication node. Wire the failure terminal of the MQInput node to the input terminal of the MQOutput node.
9. Check in the new message flow (right mouse click in the right pane and select check in from the drop down menu).
10. Select the Assignments tab.
11. Click on an execution group in the right hand pane and press the right mouse button. Select check out from the drop down menu.
12. Drag the PublishMsg message flow from the middle pane and drop it on the execution group. The message flow will now be added to the execution group.
13. Click on the execution group in the right hand pane and press the right mouse button. Select check in from the drop down menu.
14. Click on the execution group in the right hand pane and press the right mouse button. Select Deploy->Delta Assignments Configuration from the drop down menu.
15. Confirm that the deploy operation was successful by selecting the Log tab and pressing the refresh button. Remember that the broker must be running for a deploy operation to complete.
16. Using MQSeries Explorer, confirm that the open input count of the PUBLISH.IN queue is now one.

Simple subscription example

1. Using one of the RFHUtil sessions, select the PubSub tab.
2. Select the Sub (subscribe) option.
3. Enter STOCK/IBM/# as a topic. The “#” (hash) character is a multilevel wild card. This subscription should match any topics that begin with STOCK/IBM/.
4. Select the broker queue manager from the drop down list for Queue Manager to Connect to. If the broker is running on a different system, then select the name of the local queue manager to connect to and insert the name of the broker queue manager in Broker Queue Manager Name. Note that the queue name is automatically set to SYSTEM.BROKER.CONTROL.QUEUE. Set the Publish Queue Manager to the name of the local queue manager and set the publish queue to PUBLISH.OUT. Select the New Only option.
5. Select the MQMD tab and set the reply to queue manager to the local queue manager and the reply to queue name to PUBLISH.REPLY. Select the NAN report option.
6. Select the PubSub tab and press Process Request. A subscribe message should be sent to the broker, as indicated in the messages window.
7. Switch to the WMQI control center and select the Subscriptions tab.
8. Press the refresh button. Locate the new subscription. Widen the Client column to display the queue name that the subscription applies to. If the subscription is not displayed after the refresh button has been pushed, please see the following section on problem diagnosis.

Problem Diagnosis

1. If publish or subscribe messages do not appear to be working, the following steps should be followed to attempt to ascertain the cause of the problem.
2. Make sure the broker and configuration manager are running.
3. When checking for subscription status, make sure the refresh button has been pushed after a reasonable period of time has elapsed (10 seconds is usually enough).
4. Open the event viewer and check the application folder for error messages.
5. Make sure you have specified a valid reply to queue and queue manager for the subscription on the PubSub tab. As an alternative, the reply to queue manager and queue from the MQMD will be used if the reply to queue is not specified in the

PubSub folder. If no reply to queue and queue manager is specified, then the subscription request will be rejected but no error message will be returned.

6. Send a request message rather than a datagram message and specify the reply to queue in the MQMD. Alternatively, turn on the NAN (with or without PAN) in the report options and specify a reply to queue name in the MQMD. This will result in a PubSub reply message being sent to the reply to queue specified in the MQMD.

Simple publication example

This example will publish a message. It assumes the subscription used in the previous example is in effect, so the message will be published and will arrive on the PUBLISH.OUT queue.

1. Create a simple text file using notepad. This file will be used as the data part of the published message. In this example, we will be publishing a sample stock price for IBM, so it probably makes sense to enter data that can be interpreted as a stock price. However, any data will work, as this example has no dependency on the actual data used. The following line of text represents a priced for IBM in an XML format, with the price assumed to be in US Dollars and cents. Save the file as ibmstock (notepad will add an extension of txt).
   <IBM><Price>00010000</Price></IBM>
2. Start an RFHUtil session and select the Main tab and read the file you created.
3. Select the PubSub tab and select the Publish option. Enter STOCK/IBM/PRICE in the topic field. Enter the name of the broker queue manager as in the subscription example and enter PUBLISH.IN as the queue name.
4. Press the Process Request button.
5. Using the MQExplorer, examine the PUBLISH.OUT queue. You should see the published message.
6. Next, create a second simple text file using notepad. Type in the following line as the only data in the file and save the file as ibmstock2 (notepad will add an extension of txt).
   <IBM><TRADE>100</TRADE><PRICE>00009500</PRICE></IBM>
7. Using the RFHUtil session, select the Main tab and read the file you just created.
8. Select the PubSub tab and select the Publish option. Enter STOCK/IBM/TRADE in the topic field. Enter the name of the broker queue manager as in the subscription example and enter PUBLISH.IN as the queue name.
9. Press the Process Request button.
10. Using MQExplorer, examine the PUBLISH.OUT queue. You should now see both messages, since the topic in the subscription contained a wild card and therefore matched the topic strings of both of the published messages.

Reply and error messages

There are two different ways to request that publish/subscribe reply messages are to be generated by the broker.

The first method is to set the PAN and/or NAN report options. The PAN report option requests that replies be sent from the processing application (broker) when a successful action is taken (e.g. no errors). In the case of publish and subscribe, the broker is the processing application. Similarly, the NAN report option indicates that replies should be sent when a negative action is taken (e.g. errors occur).

The second method is to send the message with the message type in the MQMD set to request rather than datagram.

In either case, the reply to queue name and queue manager name must be set in the MQMD. In all cases, the reply message will be in the form of a publish/subscribe reply message. The message body will contain only an RFH2 header with a single publish/subscribe reply folder.

The RFHUtil program is capable of reading such a message from a queue and displaying the results. If such a message is read from a queue, the reply data can be found on the pscr tab.

Reply messages can be requested for any publish-subscribe operation, including publish, subscribe and unsubscribe.

The following exercise will demonstrate the production and interpretation of publish-subscribe reply messages.

1. Repeat the publication exercise above. For the first message, change to the MQMD tab and set the message type to request and the reply to queue name to PUBLISH.REPLY in the MQMD. Set the reply to queue manager name as appropriate to match the reply to queue. For the second message, set the message type to datagram, and set the PAN and NAN report options, set the reply to queue name to PUBLISH.REPLY and set the reply to queue manager name to match this queue on the MQMD tab.
2. Use MQExplorer to examine the reply messages in the PUBLISH.REPLY queue. You can read the messages with one of the RFHUtil sessions. To read a reply message, select the Main tab and select the queue manager name as appropriate. Set the queue name to PUBLISH.REPLY and press the READQ button. Examine the pscr, rfh and MQMD tabs.

Simple unsubscribe request

The following example will remove the subscription used in the previous example.

1. Using an RFHUtil session, select the PubSub tab. Select the Unsubscribe option.
2. Enter STOCK/IBM/# as the topic, and enter PUBLISH.OUT as the Publish Queue Name. Set the Publish Queue Manager name to match this queue.
3. If an acknowledgment message is desired, select the MQMD tab, and then select the PAN and NAN report options. Enter PUBLISH.REPLY as the reply to queue name, and set the reply to queue manager to match the queue name. Return to the PubSub tab.
4. Press the Process Request button.
5. Using the WMQI control center, select the Subscriptions tab. Press the refresh button. The subscription should have been removed.
6. If an acknowledgment message was requested, use MQExplorer to examine the PUBLISH.REPLY queue. If desired, read the acknowledgment message using an RFHUtil session, and examine the pscr tab.

Retained publications and state

Normally, to receive a published message, an application must subscribe before the message is published. When a message is published, a copy of the message is sent to each subscription that matches the topic string. Once this processing is complete, the original message is discarded. This type of publish/subscribe is often referred to as events or as a push operation (messages are “pushed” from the broker to the client).

A second form of publication is also available. This type of publish/subscribe is often referred to as state. It can operate as in normal publish/subscribe (events), in that applications can subscribe and will then be sent additional messages as they are published. Retained publications offers additional possibilities however. Retained publications make it possible to support “pull” as well as “push” operations, where the client requests the publication when it needs the information, and receives a copy of the last message published on a given topic.

When a message is published and the retained publication option is selected the broker will save the last message for a given topic. Each new message published for a given topic will replace the previous message for the same topic.

Retained publications can be set to expire at a given time. To use this facility, set the message expiration in the MQMD.

Retained publications are stored in a table (BRETAINEDPUBS) in the broker database. It may be of interest to use the DB2 control center and examine this table (using sample contents) at various times during this exercise.

There are some additional options on subscriptions that can affect which retained publications are sent in response to a subscription, both when the subscription request itself is processed and when published messages are processed.

The New Only subscription option indicates that the current retained publications that match the topic string of a new subscription should not be sent. This option indicates that only messages that are published after the subscription are to be sent. The On Demand option indicates that messages for this subscription are only to be sent when a request publication message is sent from the client.

The Other Only option indicates that this publication is not to be sent to the publisher, even if the publisher has a subscription that matches the publication. This option is used for conference type applications where a publisher also subscribes to the same topic.

The inform if retained option will set an additional option if a retained publication is returned in response to either a register subscription or a request publication message. If the register subscription message has this option set, then the retained publication option will be set for any retained publications that are sent out as a result of processing the subscribe or request publication message. This flag will not be set for any messages that are sent from normal publishing of messages.

Publishing a retained publication and using Request Publication

1. Using the WMQI control center, check the current subscriptions and make sure there is no current subscription for STOCK/IBM/PRICE (select the subscriptions tab and press the refresh button).
2. Using notepad, edit the IBM stock price file (ibmstock.txt) so the price is 0010500 ($105.00).
3. Using an RFHUtil session, select the main tab and press the *Read File* button. Navigate to the stock price file in the previous step and select the file. Press the *Ok* button to read the file.
4. Select the MQMD tab. Set the PAN and NAN report options. Enter PUBLISH.REPLY as the reply to queue name. Set the reply to queue manager name to match this queue name.
5. Select the PubSub tab, and then select the publish option.
6. Enter STOCK/IBM/PRICE as the topic. Select the Retain Pub option. Set the Queue Name to PUBLISH.IN and the Queue Manager Name(s) to match.
7. Press the Process Request button. A retained publication should now have been created for topic STOCK/IBM/PRICE.
8. Using MQExplorer, examine the PUBLISH.REPLY queue.
9. Select the main tab and press the Clear Data button to clear the user data buffer.
10. Select the MQMD tab. Deselect the PAN and NAN report options. Select the PubSub tab.
11. A subscription is necessary in order to send request publication messages. Using the RFHUtil session, unselect the Retain Pub option and then select the Sub (subscribe) radio button. Select the New Only, On Demand and Inf if Retained options. The On Demand option will result in published messages begin sent to this subscription in response to a request publication request, and not automatically when messages are published. Messages will only be sent in response to a request publication message from the subscriber. Set the Publish Queue name to PUBLISH.RETAIN and set the Publish Queue Manager name to match this queue.
12. Press the Process Request button. The subscription request should now be processed.

13. Check the PUBLISH.REPLY and the PUBLISH.RETAIN queues.
14. Using the WMQI control center, select the Subscriptions tab and press the refresh button. Verify that the subscription request has been processed and that the subscription for STOCK/IBM/PRICE is now registered with the broker.
15. We will now request the current retained publication. Using the RFHUtil session, select the PubSub tab and then the Req Pub (Request Publication) option. The Queue Name parameter is automatically set to the correct queue (SYSTEM.BROKER.CONTROL.QUEUE).
16. Using MQExplorer, examine the PUBLISH.REPLY and PUBLISH.RETAIN queues.
17. Press the Process Request button twice.
18. Using MQExplorer, examine the PUBLISH.REPLY and PUBLISH.RETAIN queues. There should be an additional message in each queue. Notice that there are two copies of the same message, since two publication requests were sent.
19. We will now change the retained publication and request it again. Using notepad, change the IBM stock price to 0011000 ($110.00) from 0010500 ($105.00). Save the file.
20. Using an RFHUtil session, switch to the main tab, press the Read File button and select the file you edited in the previous step.
21. Select the PubSub tab. Select the Publish option. Set the topic to STOCK/IBM/PRICE. Select the Retain Pub option. Set the Queue Name to PUBLISH.IN and set the Queue Manager Name to match the queue name.
22. Press the Process Request button. The new message should now be published and the retained publication should be updated.
23. Select the PubSub tab. Select the Req Pub option. Set the topic to STOCK/IBM/PRICE. Set the queue manager fields to match the broker. Set the Publish Queue to PUBLISH.RETAIN and the Publish Queue Manager to match this queue.
24. Press the Process Request button. A copy of the updated publication should now be delivered to the PUBLISH.RETAIN queue.
25. Using MQExplorer, examine the PUBLISH.RETAIN queue.
26. The retained publication will be deleted and the subscription will now be removed. Select the PubSub tab. Select the *Delete Pub* option. Set the topic to STOCK/IBM/PRICE. Set the *Queue Name* to PUBLISH.IN and the Publish Queue Manager to match this queue. Press the Process Request button. The retained publication should now have been deleted.
27. Select the PubSub tab. Select the Unsubscribe option. Set the topic to STOCK/IBM/PRICE. The *Queue Name* to should be set automatically to SYSTEM.BROKER.CONTROL.QUEUE. Set the *Queue Manager to Connect to* value to the broker's queue manager.
28. Press the Process Request button. Use the WMQI control center to verify that the subscription has been removed.

Filters and content based publish/subscribe

In addition to the normal topic structure used to match published messages with subscriptions, an additional level of matching can be applied to the content of the message. A filter is an ESQL clause that evaluates to either true or false. It normally refers to fields within the message body, but can refer to fields in the MQMD and RFH2 as well.

If the filter refers to a field in the message then the message must be parsed. Therefore the message either must either contain an mcd folder in the RFH2 header (in addition to the pub sub folder) or the default properties on the MQInput node must be set.

In the case of the exercises in this appendix, the default domain was set to XML. When a subscription that contains a filter is located and the topics match, then the specified parser will be invoked and the message body will be parsed. The fields in the parsed message will then be used to evaluate the filter expression and determine if the message matches the subscription. In this exercise, a subscription will be made that will only pass messages that contain a total price greater than or equal to $100,000.00. The total price is calculated by multiplying the quantity by the price per share.

1. Using notepad, create a file called ibmsale1.txt. Enter the following text: <IBM><Price>00010000</Price><Shares>1000</Shares></IBM>.
2. Using notepad, create a second file called ibmsale2.txt. Enter the following text: <IBM><Price>00009900</Price><Shares>1000</Shares></IBM>.
3. Using notepad, create a third file called ibmsale3.txt. Enter the following text: <IBM><Price>00010500</Price><Shares>999</Shares></IBM>.
4. The first and third files exceed the threshold value. The second case does not.
5. Create a subscription, including a content filter. Start an RFHUtil session, select the PubSub tab. Select the Sub (subscribe) option. Enter STOCK/IBM/SALES as the topic. Enter "(Body.IBM.Price * Body.IBM.Shares) >= 10000000" as the filter expression (N.B. enter only the expression contained between the double quotes). Enter the name of the broker queue manager. The queue name should be set automatically. Use the PUBLISH.OUT queue defined above as the publish queue name, and set the publish queue manager name to match this queue.
6. Press the Process Request button to process the subscription request.
7. Use the WMQI control center to verify the subscription request has been processed (select the Subscriptions tab and press the refresh button).
8. Using an RFHUtil select the main tab. Press the Read File button and select the ibmsale1.txt file.
9. Select the Data tab and then select the XML option to see the data in XML format. Select the Parsed option to see the data as the filter will see it.
10. Select the PubSub tab. Select the Publish option. Enter STOCK/IBM/SALES as the topic. Enter the name of the broker queue manager and enter PUBLISH.IN as the Queue Name.
11. Press the Process Request button.
12. Use MQExplorer to examine the PUBLISH.OUT queue.
13. Repeat steps 8 through 12, using the second and third files (ibmsale2.txt and ibmsale3.txt). The *saveRFH* option on the *Read* menu can be selected to retain the settings on the pubsub tab.
14. After publishing the three messages, you should see two messages on the PUBLISH.OUT queue (the first and third).
15. The final step is to unsubscribe to remove the subscription. Using an RFHUtil session, select the main tab and press the clear data button. Select the PubSub tab and select the Unsubscribe option. Select the DeregAll option. We will use this option rather than specifying a topic and filter. Be aware that the DeregAll option will remove all subscriptions for the PUBLISH.OUT queue. If this is not desired, then specify the topic and filter and do not select this option. Specify the broker queue manager name and then enter PUBLISH.OUT as the Publish Queue name. Specify the Publish Queue Manager name to match the Publish Queue (PUBLISH.OUT).
16. Press the Process Request button.
17. Use the WMQI control center to verify that the subscription has been removed.

Shared subscription queues and correlation ids

In most cases, a subscribing application will use a separate queue to receive messages sent from a registered subscription. The subscription is identified as a combination of the message type (e.g. rfh2), and the name of the queue manager and queue where messages are to be sent. In some cases, it may be desirable to share the same queue for multiple applications. In this case, the correlation id is used to allow each application to find its messages in the shared queue. Each application sharing a queue must use a unique correlation id. Setting the correlation id in the subscription message and setting the correlasid option in the PubSub folder will establish the correlation id that the broker will use when it publishes a message. The following exercise illustrates this point.

There is one other subtle effect of using the correlation id in this manner. Normally, if multiple subscriptions match a published message, and more than one subscription calls for a message to be sent to the same queue, only a single message is sent to that queue. This checking uses the combination of queue manager and queue name to identify a duplicate subscription. If the correlation id is set, then the combination of queue manager name, queue name and correlation id is used to identify duplicate subscriptions.

The following exercise will demonstrate the use of a shared subscription queue using correlation ids.

1. Create two subscriptions using the same queue and a different correlation id. Start an RFHUtil session. Select the PubSub tab. Select the Subscribe option. Enter STOCK/IBM/SALES as the topic. Select the CorrelAsId option. Enter the name of the broker queue manager in the Queue Manager field. Enter PUBLISH.OUT as the Publish Queue and enter the Publish Queue Manager that corresponds to this queue.
2. Switch to the MQMD tab. Enter “JIM” in the correl id field.
3. Select the PubSub tab. Press the Process Request button.
4. Switch to the MQMD tab. Enter “BOB” in the correl id field.
5. Select the PubSub tab. Press the Process Request button.
6. Use the WMQI control center to verify the two subscriptions have been processed. You may need to make the Client column wider to see the full subscription identifier. Also, remember that the correlation id is displayed in hex.
7. Using the RFHUtil session, select the Main tab. Press the read file button and select the ibmsale1.txt file created in the previous exercise. Actually, any file can be used for this exercise. It just should contain some user data that can be recognized. There is no need for it to be large.
8. Select the PubSub tab. Select the Publish option. Enter STOCK/IBM/SALES as the topic. Enter PUBLISH.IN as the Queue Name and enter the broker queue manager.
9. To delete the subscriptions, use the WMQI control center. Select each subscription and press the right mouse button. Select Delete and click the left mouse button. Press Ok to respond to the information message box. Press the refresh button.

Subscription points example (versioning)

To a large extent, a subscription point functions as a high-level topic qualifier. There are some differences, especially when publishing a message. When a message is published, the subscription point is specified on the publication node, and not in the RFH2. On a subscription request, the subscription point is specified in the RFH2.

The following example will demonstrate the use of subscription points to allow clients at different levels of software to smoothly coexist in the same environment. This is one possible use of subscription points. It is not meant to imply that this is the only use of subscription points nor that subscription points are the best solution for versioning. In the example, the subscription point will indicate the level of message that the client is expecting to receive.

In the example, a new field (date sold) will be added to a message to create a new version of the message. Existing clients will continue to subscribe to a subscription point that will continue to send the older version of the messages (without date sold). New applications can subscribe to a new subscription point and receive messages in the new format (with a date sold field). Messages can be published in either format. Compute nodes will be used to remove or add the date sold field to create the two different versions of the messages before they are published.

The example will require a new message flow with two publication nodes, and a compute node to change the format of the messages. The compute node will check for the existence of the new field, and if it finds it, it will be removed for back level compatibility. For the new format publication node, a default value will be inserted if the field is not present. This allows messages to be published in either the old or new format, and for clients at either level to receive messages in a compatible format.

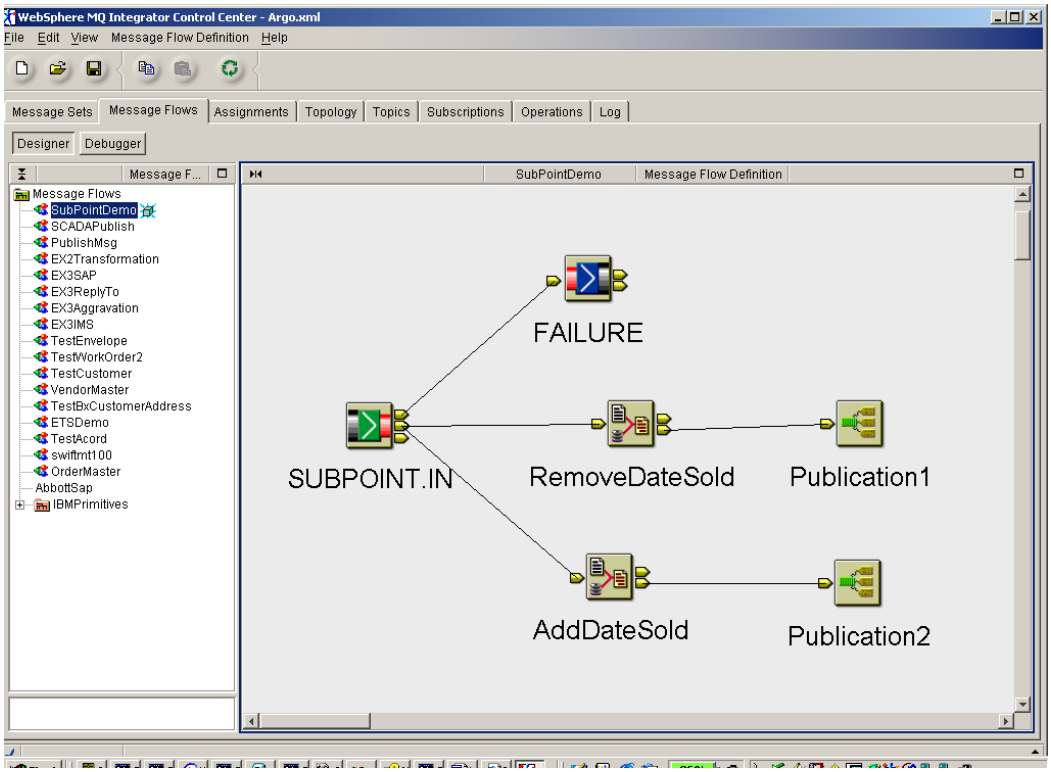
The following exercise requires several distinct steps, and is therefore broken into several parts. The first part is to create a new publication message flow with two publication nodes and support for creating two different versions of a common input message.

1. Make a copy of the publication message flow defined in a previous exercise above. To do this, select the Message Flows tab on the WMQI control center. Select the

PublishMsg flow, click the right mouse button and then select copy. Select the Message Flows item, click the right mouse button and then select paste. A new message flow called PublishMsg_1 will have been created.

2. Select the new message flow. Click the right mouse button and select rename. Change the message flow name to SubPointDemo. Press finish to rename the message flow.
3. In the left hand pane, expand the IBM Primitives by clicking on the plus sign.
4. Drag a publication node and two compute nodes to the right hand pane, and then release them.
5. Select the MQInput node, click the right mouse button and select properties. Select the basic tab and change the name of the input queue to SUBPOINT.IN. Press the Ok button. Select the MQInput node, click the right mouse button and select rename. Change the name to SUBPOINT.IN.
6. Select the first compute node and click the right mouse button. Select Rename. Change the name to RemoveDateSold and press Ok.
7. Select the second compute node and click the right mouse button. Select Rename. Change the name to AddDateSold and press Ok.
8. Select the wire connecting the input node to the publication node, click the right mouse button and select delete.
9. Wire the MQInput node to the RemoveDateSold compute node (select the input node, click the right mouse button and select Connect->Out. Drag the wire to the input terminal of the first compute node and click the left mouse button). Make a second connection from the out terminal of the MQInput node to the AddDateSold compute node.
10. Connect the out terminal of the RemoveDateSold compute node to the first publication node and the out terminal of the AddDateSold compute node to the second publication node.
11. Select the RemoveDateSold compute node. Click the right mouse button and select properties.
12. Select the Copy entire message radio button.
13. Select the ESQL tab. Insert the following statement below the generated comment:
    SET OutputRoot.XML.IBM.DateSold = NULL;
14. Select the RemoveDateSold compute node. Click the right mouse button and select properties.
15. Select the Copy entire message radio button.
16. Select the ESQL tab. Insert the following statement below the generated comment:
    IF CARDINALITY(InputBody.IBM.DateSold[]) = 0 THEN
      SET OutputRoot.XML.IBM.DateSold = CAST(CURRENT_DATE AS CHARACTER);
    END IF;
17. Select the publication node that is connected to the RemoveDateSold compute node. Click the right mouse button and select properties.
18. Enter V1 as the subscription point and press the Ok button.
19. Select the publication node that is connected to the AddDateSold compute node. Click the right mouse button and select properties.
20. Enter V2 as the subscription point and press the Ok button.
21. Select the SubPointDemo message flow. Click the right mouse button and select Check In.

The message flow is now complete. It should look like the sample below.



The next steps are to create the input queue and assign the message flow to an execution group and deploy it.

1. Using MQExplorer, create three queues named SUBPOINT.IN, PUBLISH.V1 and PUBLISH.V2.
2. Using the WMQI control center, select the assignments tab.
3. Select an execution group and press the right mouse button. Select the Check Out option.
4. Drag the SubPointDemo message flow to the selected execution group and drop it.
5. Select an execution group and press the right mouse button. Select the Check In option.
6. Select an execution group and press the right mouse button. Select the Deploy->Delta Assignments Configuration option.
7. Press Ok to acknowledge that the deploy operation has been started.
8. Select the Log tab. After at least 10 seconds, press the refresh button and confirm that the deploy operation was successful. If necessary, repeat this step until a reply is received (remember that the broker must be running for the deploy operation to execute).
9. Finally, select the operation tab and press the refresh button. You should now see your new flow in the assigned execution group.

The next step is to create some sample data in a “version 2 format”. After that, subscriptions will be created to separate queues for version 1 messages and version 2 messages. This is simulating two clients, each at a different level. Finally, messages will be published in both version 1 and version 2 formats (without and with a date sold field), and messages in the appropriate formats will be published to the respective queues.

1. Copy the ibmsale1.txt and ibmsale2.txt files created above in the filters exercise (create them first if this exercise was skipped) to create two new files called ibmsub1.txt and ibmsub2.txt. This can be done with explorer or using a command prompt (e.g. copy ibmsale1.txt ibmsub1.txt).
2. Edit the ibmsub1.txt file using notepad. Insert the following text just before the </IBM> tag:

    a. <DateSold>20020815</DateSold>
3. Save the file and exit the notepad session.
4. Start an RFHUtil session and select the PubSub tab.
5. Click the subscribe radio button. Enter “STOCK/IBM/#” as the topic name (without the double quotes). Enter V1 as the subscription point. Enter PUBLISH.V1 as the name of the Publish Queue. Set the Publish Queue Manager and connection/broker queue manager as appropriate.
6. Press the Process Request button.
7. Change the subscription point to V2 and the Publish Queue to PUBLISH.V2.
8. Press the Process Request button.
9. Using the WMQI control center, select the Subscriptions tab and press the refresh button. Verify that the two subscriptions have been made.
10. Start an RFHUtil session. Using the Main tab, press the Read File button. Select the ibmsub1.txt file and press Ok.
11. Select the PubSub tab. Select the Publish radio button.
12. Enter “STOCK/IBM/SALES” in the Topic1 field (without the double quotes). Enter SUBPOINT.IN as the queue name, and select the queue manager to connect to.
13. Press the Process Request button.
14. Using MQExplorer, examine the PUBLISH.V1 and PUBLISH.V2 queues. Display the messages using MQExplorer or read them using an RFHUtil session. Notice the difference between the two messages.
15. Using the previous RFHUtil session that was used to publish the first message, select the main tab and press the Read File button. Select the ibmsub2.txt file and press Ok.
16. Select the PubSub tab. Select the Publish radio button.
17. Enter “STOCK/IBM/SALES” in the Topic1 field (without the double quotes). Enter SUBPOINT.IN as the queue name, and select the queue manager to connect to.
18. Press the Process Request button.
19. Using MQExplorer, examine the PUBLISH.V1 and PUBLISH.V2 queues. Display the messages using MQExplorer or read them using an RFHUtil session. Notice the difference between the two messages.

Messages have now been published in both formats, and have been sent to subscribers in both formats.

The subscriptions will now be deleted.

1. Using the original RFHUtil session that was used to create the subscriptions, select the PubSub tab and then select the Unsubscribe radio button.
2. Enter PUBLISH.V1 in the Publish Queue Name field and enter the appropriate queue manager names.
3. Select the Dereg All option. Leave the topic and subscription point null.
4. Press Process Request to remove the subscription.
5. Change the Publish Queue Name to PUBLISH.V2.
6. Press Process Request to remove the subscription.
7. Using the WMQI control center, select the Subscriptions tab and press the refresh button.
8. Verify that the subscriptions have been removed.

Publishing messages without RFH2 headers

Although the RFHUtil is capable of adding the RFH2 headers required for publish and subscribe applications, this function can also be performed in a message flow. This is particularly useful for publishing applications.

The following ESQL sample can be used to insert an RFH2 header into a message tree prior to passing a message to a publication node. In this case, a message is being published. The ESQL should be placed after the comment line in a compute node. Please select the Copy Message headers only prior to entering this ESQL.

```
SET "OutputRoot"."MQRFH2".(MQRFH2.Field)"Format" = 'MQSTR   ';
SET "OutputRoot"."MQRFH2"."psc"."Command" = 'Publish';
SET "OutputRoot"."MQRFH2"."psc"."Topic" = 'STOCK/IBM/SALES';
SET OutputRoot.*[LAST] = InputRoot.*[LAST];
```

The ESQL should be placed in a compute node between the MQInput node and the Publication node. The format field may need to be changed to reflect the format of the actual user data.

Detailed instructions to create and use this message flow are given below.

1. Using MQExplorer, create a queue named PUBNORFH.IN.
2. Using the WMQI control center, select the Message Flows tab and then select the Message Flows item. Click the right mouse button and select Create->Message Flow.
3. Call the message flow PublishNoRFH and press Finish.
4. Press the plus sign next to the IBMPrimitives entry to expand the list of standard nodes.
5. Drag an MQInput node, a compute node, an MQOutput node and a publication node to the right hand pane.
6. Select the MQInput node, click the right mouse button and select properties. Select the basic tab and enter PUBNORFH.IN as the queue name. Select the Defaults tab and select BLOB from the drop down list for domains. Press the Ok button.
7. Select the MQInput node, click the right mouse button and select Rename. Change the name to PUBNORFH.IN and press Ok.
8. Select the compute node, click the right mouse button and select Properties. Select the Copy message headers only radio button. Select the ESQL tab. Enter the following ESQL at the end of the generated statements:
   ```
   SET "OutputRoot"."MQRFH2".(MQRFH2.Field)"Format" = 'MQSTR   ';
   SET "OutputRoot"."MQRFH2"."psc"."Command" = 'Publish';
   SET "OutputRoot"."MQRFH2"."psc"."Topic" = 'STOCK/IBM/SALES';
   SET OutputRoot.*[LAST] = InputRoot.*[LAST];
   ```
9. Press the Ok button.
10. Select the compute node, click the right mouse button and select Rename. Change the name to AddRFH. Press the Finish button.
11. Select the first MQOutput node, click the right mouse button and select Rename. Change the name to FAILURE. Press the Finish button.
12. Select the first MQOutput node, click the right mouse button and select Properties. Select the Basic tab and set the Queue name to FAILURE. Press the Ok button.
13. Select the MQOutput node, click the right mouse button and select Rename. Change the name to FAILURE. Press the Finish button.
14. Select the MQOutput node again, click the right mouse button and select Properties. Select the Basic tab and set the Queue name to FAILURE. Press the Ok button.
15. Using the mouse, connect the out terminal of the MQInput node (PUBNORFH.IN) to the in terminal of the compute node (AddRFH). Connect the fail terminal of the input node to the in terminal of the MQOutput node (FAILURE). Connect the out terminal of the compute node to the in terminal of the publication node.
16. Select the PublishNoRFH message flow in the left hand pane, click the right mouse button and select Check In.
17. Select the Assignments tab. Select an execution group, click the right mouse button and select Check Out. Drag the PublishNoRFH message flow from the center pane and drop it on the selected execution group. Select the execution group, click the right mouse button and select Check In. Click the right mouse button again and select Deploy->Delta Assigments Configuration.
18. Switch to the Log tab. Wait for a short time and then press the refresh button until the results of the deployment operation are displayed. Verify that the deployment was successful.

A message flow has now been added to the broker that will accept a message with no RFH2 header and will publish the message on the topic specified in the message flow. To

demonstrate this function, a subscription must be created. A message will then be sent to the message flow to be published. Once successful publication of the message has been verified, the subscription will be removed.

1. Start an RFHUtil sessions.
2. Select the PubSub tab and then select the Subscribe radio button.
3. Enter STOCK/IBM/# as a topic.
4. Select the broker queue manager from the drop down list for Queue Manager to Connect to. If the broker is running on a different system, then select the name of the local queue manager to connect to and insert the name of the broker queue manager in Broker Queue Manager Name. Set the Publish Queue Manager to the name of the local queue manager and set the Publish Queue to PUBLISH.OUT.
5. Press Process Request. A subscribe message should be sent to the broker, as indicated in the messages window.
6. Switch to the WMQI control center and select the Subscriptions tab.
7. Press the refresh button. Verify that the new subscription has been made.
8. Start a second RFHUtil session. On the main tab, press the read file button. Select a sample file from one of the earlier exercises (or create a simple text file using notepad and then select it). Select the local queue manager as the queue manager to connect to and then enter PUBNORFH.IN as the queue name.
9. Press the Write Q button to send a message to the message flow. The message should now be published and sent to the queues specified in any matching subscriptions.
10. Verify that the publication was successful with the MQSeries Explorer. Select MQSeries Explorer and then select the PUBLISH.OUT queue. View the entries in the PUBLISH.OUT queue and verify that the published message has been received.
11. Select the RFHUtil session that was used to subscribe and select the PubSub tab. Select the Unsubscribe radio button.
12. Press the Process Request button to send the unsubscribe request.
13. Using the WMQI control center, select the subscription tab. Press the refresh button and verify that the subscription has been removed.

Appendix D: Using message groups and segmentation

The following exercises demonstrate the use of message groups and segmentation. They are intended to provide the reader with a basic understanding of these features and to demonstrate how to use the RFHUtil utility with these features.

Message groups are supported on MQSeries version 5.0 and later for all platforms except OS/390, and are supported by version 5.3 on OS/390. Segmentation is not supported on OS/390. Otherwise, segmentation is supported on the same platforms that support message groups.

Message Groups

Message groups allow a number of distinct messages to be associated with a unique group id. The messages in the group are independent messages with their own data formats. There is no requirement for the messages to be of the same type, and even the code page or encoding values can be different. The messages in a group must all be either persistent or non-persistent. The group identifier can be generated by the queue manager or can be created by the sending application. However, it must be unique and must be the same for all messages in a group.

While many of the features of a group can be easily contained in a user data area, there is one aspect that is difficult to provide in the application. The receiving application can request that all messages be available on the final target queue manager before any of the messages can be retrieved. Once the first message in the group has been read, then the subsequent messages can be read in order, regardless of their physical order on the queue or the presence of other messages physically interspersed with the messages within the group. Messages can also be read from the queue in physical order, but this is usually not advisible.

To specify a group, the *Logical Order* check box on the main tab must be selected. The *Last in Group* check box must be selected before the last message in the group is written. The All Avail check box should be checked before the first message is read. If this box is selected, then none of the messages in the group will be visible until the entire group has arrived on the destination queue.

Message group exercise

To reinforce the discussion of message groups in the preceding section, a simple exercise is provided.

.

1. Create three files using notepad, calling the first msg1.txt, the second msg2.txt and the last one msg3.txt. Enter a line of distinct text for each file (e.g. “first message”, “second message” and “third and last message”, without the double quotes). Create a fourth file called notgroup.txt and enter a distinct line of text (“Not in Group”). A group containing the three messages will now be created.
2. Create a queue called GROUP.OUT (or use an existing empty queue).
3. Using an RFHUtil session, press the *Read File* button and select the first file (msg1.txt).
4. Select the *Logical Order* check box.
5. Select the queue manager and enter GROUP.OUT for the queue name.
6. Press the *Write Queue* button to write the first message.
7. On the MQMD tab, note the sequence number field, which is filled in after the message has been written. Return to the main tab.
8. Press the *Read File* button and select the second file (msg2.txt).
9. Press the *Write Queue* button to write the second message. Switch to the MQMD tab and note the sequence number field. Return to the main tab.
10. Start a second RFHUtil session and press the *Read File* button. Select the notgroup.txt file. Select the queue manager name and enter GROUP.OUT as the queue name. Press the *Write Queue* button. This will create an interspersed

message in the queue that is not part of the message group. Return to the first RFHUtil session.

11. Press the *Read File* button and select the third file (msg3.txt).
12. Select the *Last in Group* check box.
13. Press the Write Queue button to write the third and final message.
14. Using MQExplorer, examine the messages on the queue. In particular, note the group identifier and the information on the Segmentation tab for each message in the group, and the logical order of all the messages in the queue

The messages will now be read back using a browse operation. They will be read first with the *Logical Order* check box selected and the queue will then be browsed a second time without the *Logical Order* box selected.

1. Using an RFHUtil session, enter GROUP.OUT as the name of the queue and select the corresponding queue manager. Make sure that the *Logical Order* check box is selected.
2. Press the *Start Browse* button to read the first message in the queue. Since a browse operation is being used, the messages will not be removed from the queue after they are read. Switch to the Data tab and examine the data.
3. Press the *Browse Next* button to view the rest of the messages on the queue. The messages in the group should be seen in logical order, before any message that are not part of the group, regardless of the physical order of the remaining messages on the queue.
4. When all the messages on the queue have been viewed, press the *Browse Next* button one more time to end the browse operation.
5. Return to the Main tab and remove the check from the *Logical Order* check box. Return to the Data tab to examine the data in the first message on the queue.
6. Press the *Browse Next* button to view the rest of the messages on the queue, in physical order rather than logical order. The extra message that was written to the queue in the middle of the group should now be read before the last message in the group.
7. Once the messages in the queue have been examined, they can be removed with the *PurgeQ* button on the main tab.

Message Segmentation

Message segmentation is similar in many ways to message groups, in that a group of distinct physical messages are associated with each other to form a logical grouping. However, in the case of segmentation, the messages are considered to form a single logical data area (single message) rather than a group of distinct logical messages. In the case of segmentation, an offset is maintained for each successive physical message rather than a logical sequence number. The offset of the second message matches the end of the data portion of the first message, and each successive message increases the offset by the size of the user data in the preceding message.

It is worth noting that segmentation can be combined with message groups. Each individual logical message in a message group can consist of one or more message segments.

Automatic (Queue manager) segmentation

There are two distinct ways to invoke message segmentation. Messages larger than the maximum size allowed are automatically split into as many distinct physical messages as required so that each segmented message is no larger than the maximum allowed. A maximum definition can be specified as a property of an individual queue or for the queue manager itself. Message segmentation can also be controlled by the application, as demonstrated in the next section.

The following exercise will demonstrate automatic segmentation of messages by the queue manager.

1. Using MQExplorer, select the queues line under the desired local queue manager and click the right mouse button. Select New->Local Queue. Enter SEGMENT.OUT as the name of the queue and select the extended tab. Change the maximum message length parameter to 1024. Press Ok to create the queue.
2. Locate a file that is approximately 2500-3000 bytes long. The actual length of the file and the data in the file is not important. It is better if the file contains text data that can be recognized but this is not essential. If such a file cannot be easily located, then create a file using notepad (enter some text and then use the copy and paste functions of notepad to rapidly create a file of sufficient size).
3. Using an RFHUtil session, press the *Read File* button and select the file from the previous step.
4. Enter SEGMENT.OUT as the name of the queue and select the appropriate queue manager.
5. Press the *Write Q* button. An error message should be generated indicating that the message is too long to write to the specified queue.
6. Select the *Auto Segmentation* option on the main tab and press the *Write Q* button again. This time, the message display on the main tab should indicate that a message was sent, and the queue depth should increase by the number of physical messages necessary to contain the segmented message.
7. Using MQExplorer, examine the SEGMENT.OUT queue. Display each message in the queue, and look at the Identifiers, Segmentation and Data tabs. There should be more than one physical message, with all except the last message containing 1024 bytes and the last message containing the remaining data. Note the offsets on the Segmentation tab.

The messages we created will now be read back, both as individual physical messages and then as a single complete message.

1. Using the RFHUtil session, uncheck the *Auto Segmentation* option. Leave the queue manager and queue names set as before. Press the *Browse Q* button. The first physical message (segment) in the queue will now be read.
2. Note the data length, and examine the data and MQMD tabs. The offset will be zero, since the first physical message was read. To see all the messages, start a browse operation using the *Start Browse* and *Browse Next* buttons.
3. On the main tab, select the Auto Segmentation option. Press the Browse Q button. This time, the entire logical message (all segments) will be read as if they were a single message. The only indication of segmentation will be the segment flags on the MQMD tab.
4. When done examining the messages on the queue, press the *Purge Q* button to delete all messages on the queue.

When messages are automatically segmented, the length of all messages other than the last one will be a multiple of 16 that is less than or equal to the maximum message size allowed.

Application controlled segmentation

In addition to automatic segmentation by a queue manager, applications can also split a single logical message into more than one physical message. In the case of application segmentation, multiple MQ put operations must be used, and the segmentation flags must be set in the MQMD.

The following exercise will demonstrate application-controlled segmentation of a message.

1. Three files will be created to demonstrate application-controlled segmentation. The files can be created with notepad and should be named seg1.txt, seg2.txt and seg3.txt. Take a complete sentence or phrase and break it into three parts. Enter the first part as the contents of the first file, the second part as the contents of the second file and the last part as the contents of the third file.

2. Start an RFHUtil session and press the Read File button. Select the first file (seg1.txt). Enter SEGMENT.OUT as the name of the queue and select the corresponding queue manager.
3. Select the MQMD tab. Select the *Segment Yes* check box. Set the offset field to zero. Make sure the *Segment Last* check box is NOT selected.
4. Return to the main tab and press the *Write Q* button.
5. Press the read file button and select the second file (seg2.txt). Press the *Write Q* button.
6. Press the read file button and select the third file (seg3.txt). Select the MQMD tab. Select the Segment Last check box (it does not matter if the *Segment Yes* check box is also selected).
7. Press the *Write Q* button to write the last segment of the message.
8. Examine the contents of the SEGMENT.OUT queue using MQExplorer. In particular, display each message in the queue and observe the Identifiers, Segmentation and Data tabs.
9. The segments can now be read back as a single message and then individually using the same RFHUtil session. On the main tab, select the *Auto Segmentation* check box. Press the *Read Q* button. The entire logical message should have been read in as if it were a single message.
10. On the main tab, deselect the *Auto Segmentation* check box. Press the *Start Browse* button. The physical message with the first segment should have been read. Examine the MQMD and data tabs.
11. On the main or data tab, press the *Browse Next* button. Examine the MQMD and data tabs.
12. Repeat the previous step to view the last message segment.

End of Document