MQSeries Integrator V2
PUT Utility

Version 1.1

1st May 2001

Jill Lennon

jdlennon@us.ibm.com

Ed Fletcher

efletch@uk.ibm.com

Take Note!

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

Third Edition, May 2001

This edition applies to Version 1.1 of *MQSeries Integrator V2 - Put Utility* 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 each item has been reviewed by IBM 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

The following terms are trademarks of other companies:

- Windows NT Microsoft Corporation
- NEON New Era of Networks, Inc.

Acknowledgments

Many thanks to the MQSeries Integrator V2 Test team at Hursley for kindly allowing us to base this SupportPac on one of their test tools.

Summary of Amendments

Date	Changes
09 October 2000	Initial release
12 October 2000	Renamed sample input files to .txt instead of .msg. Correction to sample RFH2 file to correct RFH2 header (Msd, Set, Type and Fmt - upper-cased first character of these field names)
01 May 2001	Added new executable, mqsiputc.exe. This is the same as the mqsiput.exe file, but compiled with the MQSeries Client library and so can be used over a client connection.

Preface

This SupportPac consists of a sample utility program in executable form that allows you to place messages on named queues to enable easier verification of MQSeries Integrator V2 implementations. It provides additional function over that supplied in the MQSeries Explorer Put Message function or the MQSeries sample program - *amqsput*.

Source code is not provided for this SupportPac..

Bibliography

- *IBM MQSeries Application Programming Reference,* IBM Corporation. SC33-1673
- *IBM MQSeries Integrator V2 Programming Guide,* IBM Corporation. SC34-5603

Chapter 1. The MQSIPUT Utility

Overview

IBM's MQSeries Integrator (MQSI) V2 provides a powerful solution to the challenge of integrating both data and applications in an enterprise. In its simplest form, MQSI takes a description of a message (layout) and, when presented with messages in this format, can break apart that message into its constituent fields. Once the original message has been parsed, the message can be output in a different format using the fields/elements contained in the original message.

MQSIPUT is a simple utility that allows messages to be put into various formats allowed in the MQSeries Integrator V2 product and to include various headers. The values of fields in those headers can be set using MQSIPUT. This will allow for ease of testing both valid and invalid messages without having to write programs to set the headers. In addition, it is useful to test the installation of the product.

It runs in a Windows NT environment but normal distributed MQSeries messaging allows the resulting output message from the utility to be routed to a broker on any supported platform.

Installation

The SupportPac is supplied in the zip file, ih02.zip. This should be uncompressed to a directory of choice and will produce the following files:

- mqsiput.exe — The utility for running as an MQSeries application in a Windows NT environment
- mqsiputc.exe — The utility for running as an MQSeries Client application in a Windows NT environment
- rfh2.txt — A sample parameter file providing an example of an MQRFH2 header
- xml.txt — A sample parameter file providing an example of XML message data
- ih02.pdf — This User Guide in Adobe Acrobat format

MQSIPUT Details

MQSIPUT is somewhat like amqsput in the base MQSeries product. It must be invoked from a DOS command prompt. However, it requires the naming of both the queue and the queue manager as well as the use of an input file for the message data.

Invocation of MQSIPUT looks like the following:

C:\MQSIPUT Q_IN QMGRNAME < InputFile.txt

Where:

Q_IN is your queue name where the messages are to go

QMGRNAME is the name of the queue manager owning the output queue

InputFile.txt is the file which contains the actual data to be placed on the queue.

In the following chapter, you will see the various headers that are supported and what parameters may be used. You will also be referred to the *IBM MQSeries Application*

Programming Reference (SC33-1673) for additional assistance. This user guide will not repeat all the information contained in that manual.

Setting up the parameter file

A sample file

```
# A SAMPLE MQSIPUT INPUT FILE
OPTIONS

DEBUGLEVEL          1
OPENOPTIONS         2064
TESTSTART

MQMD

STRUCID             MD
VERSION             2
REPORT              0
MSGTYPE             1
EXPIRY              -1
FEEDBACK            0
FORMAT              MQSTR
PRIORITY            0
PERSISTENCE         1
BACKOUTCOUNT        0
REPLYTOQ            MQREPLY

STARTDATA

<Message>
 <Version>2</Version>
 <CustomerName>
   <First>Ed</First>
   <Last>Fletcher</Last>
 </CustomerName>
 <Address>
   <Line1>123 Shady Lane</Line1>
   <Line2>Suite 1009</Line2>
   <Town>Anytown</Town>
   <County>MyCounty</County>
   <Zip>22345</Zip>
   <Country>US</Country>
 </Address>
 <Complaint>
   <Type>Order</Type>
   <Reference>XYZ123ABC</Reference>
   <Text>I am sending a complaint</Text>
 </Complaint>
</Message>
ENDDATA

TESTEND
```

Setting up an input file

The sample shown is a simple xml message. It has only one header, the MQMD. You do not need to set all the values in any of the headers; those not set will default. Remember, this is just an MQSeries program. Now we will look at the various settings used to tailor the output messages

Comments

You can add a comment line into the input file by starting the line with the # symbol.

Options

Denotes that what follows must be a keyword describing a global property.

OPTIONS has scope until TESTSTART is declared. OPTIONS must be the first keyword if it is invoked.

You can specify quite a bit or let things default. In the sample, we have set DEBUG level and the MQOO_Options (Open options for the MQSeries API). 2064 means that the queue will be opened for output an all context can be set.

DEBUGLEVEL n

The level (**n**) at which messages are displayed to the user where **n** can be one of the following:

0 Standard Level - presenting the major program events:

Example:

Time: 10:31:12 Date: 15/02/00

program started

c:\parsers\data_files\mqsiput_input\misc\badcfsl.dat

End of message definition... putting to queue...

Qmgrname: parse

MQCONN reason code: 0

MQOPEN: sgrq

MQOPEN reason code: 0

Message data: ?

MQPUT reason code: 0

MQCLOSE reason code: 0

MQDISC reason code: 0

MQSIPUT SUCCESS

1 Debug Level - presenting detailed debugging information:

Example:

```
Time:  11:21:36       Date:  15/02/00
program started
c:\parsers\data_files\mqsiput_input\misc\badcfsl.dat
Debuglvl: 1
Record read:   OPENOPTIONS            2064
Openoptions: 2064
Record read:   MQPMOOPTIONS           2048
Put message options: 2048
Record read:
Record read:   TESTSTART
Record read:
Record read:   MQMD
Record read:   FORMAT                 MQPCF
MQMD Format: MQPCF
Record read:
Record read:   MQCFH
Record read:   TYPE                   1
MQCFH Type options: 1
Record read:   STRUCLENGTH            36
MQCFH StrucLength: 36
Record read:   VERSION                1
MQCFH Version options: 1
Record read:   COMMAND                123
MQCFH Command: 123
Record read:   MSGSEQNUMBER           1
MQCFH MsgSeqNumber: 1
Record read:   CONTROL                1
MQCFH Control: 1
Record read:   COMPCODE               0
MQCFH CompCode: 0
Record read:   REASON                 0
MQCFH Reason: 0
Record read:   PARAMETERCOUNT         1
MQCFH ParameterCount: 1
Record read:
Record read:   MQCFSL
```

```
Record read:    TYPE                6
MQCFSL Type options: 6
Record read:    PARAMETER           456
MQCFSL Parameter options: 456
Record read:    CODEDCHARSETID      0
MQCFSL CodedCharSetId: 0
Record read:    STRINGS
Record read:    STARTDATA           3 10
STARTDATA: 3
Record read:    data
Message data:
Record read:    ENDDATA
Record read:
Record read:    TESTEND
End of message definition... putting to queue...
Qmgrname: parse
MQCONN reason code: 0
MQOPEN: sgrq
MQOPEN reason code: 0
Message data: ?
MQPUT  reason code: 0
MQCLOSE reason code: 0
MQDISC reason code: 0
MQSIPUT SUCCESS
```

2 Standard Level - presenting the major program events:

Example:

```
Time:   11:42:13      Date:   15/02/00
program started
c:\parsers\data_files\mqsiput_input\miscadcfsl.dat
MQSIPUT SUCCESS
```

DUPLICATE c

The number of times that the test is executed with one invocation of mqsiput. It also denotes how many times an MQPUT is executed.

Default value 1.

TEXTDELIMITER c

Used to enable white space to be captured.

Default value x'02'.

Example:

```
TEXTDELIMITER    '
FORMAT           'MQHWIH  '
```

DELIMITER c

Used to enable hexadecimal data to be written using character format. One byte of binary is represented by two characters in the range 0-9 and A-F.

Default value x'01'.

Example:

```
DELIMITER    ++
FORMAT       ++4D51485749482020++
```

FILEDELIMITER c

Used to enable data to be captured in from a file.

Default value x'03'.

Example:

```
FILEDELIMITER    ==
STARTDATA
==c:\mqsi\data\include.dat==
ENDDATA
```

NONALIGN

Instruction to RFH, RFH2 and PCF not to align its data on a 4-byte boundary which is the default by padding with spaces.

MESSAGELENGTH nnn

Code MESSAGELENGTH if your output buffer needs to be more than 32k

COMMITMQ

Code COMMITMQ if using MQPMO_SYNCPOINT. The default for NT is ROLLBACK

ROLLBACKMQ

Code ROLLBACKMQ if using MQPMO_SYNCPOINT. The default for MVS is COMMITMQ

QMGRNAME ccccccccccccccccccccccccccccccccccccccccccccccccc

Code QMGRNAME if using neither the default or not specifying the queue manager as an input line parameter

QNAME ccccccccccccccccccccccccccccccccccccccccccccccccc

Code QNAME if not specifying the queue name as an input line parameter

OPENOPTIONS nnnnn

Used to specify the OPTIONS value on an MQOPEN

Default value: 2064

MQPMOOPTIONS nnnnn

Used to specify the OPTIONS value in the MQPMO

Default value: 0

Headers

Following the Options settings, one or more headers can be specified. The fields that can be specified for each header and their associated values are simply the ones allowed in the MQSeries MQI. Those can be found in the *IBM MQSeries Application Programming Reference*. Additional information on programming with the MQRFH2 is available in the *MQSeries Integrator Version 2 - Programming Guide*.

Samples of the MQMD and MQRFH2 are in **Appendix A**.

Note: **PRIOR** to the first header (the MQMD), you must notify the MQSIPUT program that you are done with options by specifying the keyword **TESTSTART**.

Some of the headers that have variable data may also need to have keywords to denote their start and end. For example, the MQRFH2 has variable data depending on what you want to specify. You need to specify the start and end of the data as the following example shows:

MQRFH2
NAMEVALUECCSID 1208
NAMEVALUEDATA
STARTDATA
<mcd><Msd>NEON</Msd><Set>AppGrp1</Set><Type>MsgInFormat</Type></mcd>
ENDDATA

Now let's look at a list of all the supported headers.

MQMD	Header to signify an MQMD structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQMDE	Header to signify an MQMDE structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQDLH	Header to signify an MQDLH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQIIH	Header to signify an MQIIH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQCIH	Header to signify an MQCIH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQRMH	Header to signify an MQRMH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQWIH	Header to signify an MQWIH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQRFH	Header to signify an MQRFH structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQRFH2	Header to signify an MQRFH2 structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQCFH	Header to signify an MQCFH structure. Its field names are as per the MQSeries Application Programming Reference manual.

MQCFIN	Header to signify an MQCFIN structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQCFST	Header to signify an MQCFST structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQCFIL	Header to signify an MQCFIL structure. Its field names are as per the MQSeries Application Programming Reference manual.
MQCFSL	Header to signify an MQCFSL structure. Its field names are as per the MQSeries Application Programming Reference manual.

Finally, we will now look at how you specify your actual message data.

Message Data

Your message data can be a mixture of datatypes as long as you have properly specified delimiters for the various datatypes. First, an example of strictly text data, an XML message:

```
STARTDATA
<Message>
<Name>IBM</Name>
<Version>2</Version>
<Complaint>
<Type>Delivery</Type>
</Complaint>
</Message>
ENDDATA
```

As you can see, you must specify the beginning and end of your message with the **STARTDATA** and **ENDDATA** tags.

Let's look at an example of how to specify some integer data along with some text data in the following (assuming that DELIMITER %% has been specified in the Options settings):

```
STARTDATA
%%00000001%%
XYZ Company
ENDDATA
```

Note that no white space will be carried with the text data. If you need a field to be longer than the actual data, you need to specify TEXTDELIMITER in the Options. For example, in the above, let's assume the text field needs to be 15 characters long (a fixed length field) and assume that TEXTDELIMITER $$ has been specified:

```
STARTDATA
%%00000001%%
$$XYZ Company    $$
ENDDATA
```

This allows white space to be kept in a field.

Summary

You can see that this utility requires accurate input data but it can save some time when you are trying to learn the MQSI V2 product or when you want to test some variable conditions to see how your message flows handle them.

Appendix A

Following are just a few sample input files that could be used in your testing:

Straight XML input (self-defining):

```
OPTIONS

DEBUGLEVEL              1
OPENOPTIONS             2064

TESTSTART

MQMD

STRUCID                 MD
VERSION                 2
REPORT                  0
MSGTYPE                 8
EXPIRY                  -1
FEEDBACK                0
ENCODING                273
CODEDCHARSETID          437
FORMAT                  xml
PRIORITY                0
PERSISTENCE             1
BACKOUTCOUNT            0
REPLYTOQ                MQREPLY

STARTDATA
<Message>
<Name>IBM</Name>
<Version>2</Version>
<Complaint>
<Type>Delivery</Type>
</Complaint>
</Message>
ENDDATA
```

AN MRM message (an integer and fixed length string):

```
OPTIONS
DELIMITER %%
OPENOPTIONS             16
DEBUGLEVEL              1
TESTSTART

MQMD
MSGTYPE                 8
FORMAT                  MQHRF2

MQRFH2
NAMEVALUEDATA
STARTDATA
<mcd><Msd>MRM</Msd><Set>DHA1HAO06S001</Set><Type>EX1_MESS_ID</Type><Fm
t>cwf</Fmt></mcd>
ENDDATA
STARTDATA
%%00000001%%
This is your shipment
ENDDATA
TESTEND
```

A NEON message (delimited fields):

```
OPTIONS
DEBUGLEVEL      1
TEXTDELIMITER $$
TESTSTART
MQMD
FORMAT                  MQHRF2
MQRFH2
NAMEVALUECCSID      1208
NAMEVALUEDATA
STARTDATA
<mcd><Msd>NEON</Msd><Set>MYAPPGRPNAME</Set><Type>MYINFMT</Type></mcd>
ENDDATA
STARTDATA
$$Jill;Lennon;$$
ENDDATA
TESTEND
```

One thing to note, as with the MQRFH, the FORMAT field in the MQMD specifies MQHRF2, not MQRFH2.

End of Document