MQSeries® Integrator

Application Development Guide

Version 1.1

SC34-5508-02

Note: Before using this information, and the product it supports, be sure to read the general information under *Notices* on page 39**.**

Third edition (December 1999)
This edition applies to IBM® MQSeries Integrator, Version 1.1 and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1
Introduction

MQSeries Integrator Overview

MQSeries Integrator, from IBM and New Era of Networks, Inc. (NEON), provides the flexibility and scalability that allows true application integration. MQSeries Integrator consists of four components:

- IBM MQSeries
- NEONFormatter
- NEONRules
- MQIntegrator Rules daemon

MQSeries Integrator is a cross-platform, guaranteed delivery, messaging middleware product designed to facilitate the synchronization, management, and distribution of information (messages) across large-scale, heterogeneous networks.

MQSeries Integrator is configurable and uses a content-based rules message evaluation, formatting, and routing paradigm. MQSeries Integrator also provides a powerful data content-based, source-target mechanism with dynamic format parsing and conversion capability.

The application program interfaces (APIs) and graphical user interfaces (GUIs) allow you to use these systems. Refer to the ***Programming Reference*** documents for instructions on using the APIs and the ***User's Guide*** for instructions on using the GUIs.

Formatter

NEONFormatter translates messages from one format to another. NEONFormatter handles multiple message format types from multiple data value sources with the ability to convert and parse messages. When a message is provided as input to Formatter, the message is parsed and data values are returned.

Message formats in the NEONFormatter database are defined through the graphical user interface (GUI). The GUI leads you through the definitions of format components, for example, tags, delimiters, and patterns, to the building of complete message definitions.

Rules

NEONRules lets you develop rules for managing message destination IDs, receiver locations, expected message formats, and any processes initiated upon message delivery. Creation and dispatch of multiple messages to multiple destinations from a single input message is supported.

Note:
For more in-depth descriptions of NEONFormatter and NEONRules, refer to the overviews in Chapters 3 and 4 of the ***MQSeries Integrator User's Guide***.

MQSeries Integrator Rules Daemon

The MQSeries Integrator Rules daemon combines MQSeries, NEONFormatter, and NEONRules in a generic server process. The MQSeries Integrator Rules daemon processes messages from an MQSeries input queue, uses NEONFormatter to parse messages, uses NEONRules to determine what transformations to perform and where to route the messages, and then puts the output messages on MQSeries queues for delivery to applications.

MQSeries

MQSeries is a message-oriented middleware that is ideal for high-value message handling and high-volume applications because it guarantees each message is delivered only once, and it supports transactional messaging. Messages are grouped into units of work and either all or none of the messages in a unit or work are processed. MQSeries coordinates message work with other transaction work, like database updates, so data integrity is always maintained.

Product Documentation Set

The MQSeries Integrator for OS/390 documentation set includes:

- ***MQSeries Integrator for OS/390 Installation and Configuration Guide*** details the installation and initial implementation of MQSeries Integrator and the MQSeries Integrator applications.
- ***User's Guide*** helps MQSeries Integrator users understand and apply the program through its graphical user interfaces (GUIs).
- ***System Management*** is for SPs and DBAs who work with MQSeries Integrator on a day-to-day basis.
- ***Programming References*** are intended for users build and maintain the links between MQSeries Integrator and other applications. This document includes the following volumes:
  - ***Application Development Guide*** assists programmers in writing applications that use MQSeries Integrator APIs.
  - ***Programming Reference for NEONFormatter*** is a reference to NEONFormatter APIs for those who write applications to translate messages from one format to another.
  - ***Programming Reference for NEONRules*** is a reference to NEONRules APIs for those who write applications to perform actions based on message contents.

Before You Contact Technical Support

If you have difficulty executing one of the MQSeries Integrator programs, analyze your environment using the following steps. Be prepared to send the listed information and files to technical support.

1. Has this program ever worked in your environment?

   If so, identify what has changed.

2. Check the values specified in the SQLSVSES (DD-name SQLSVSES) file that the failing job is using to make sure it refers to an existing DB2 subsystem and an existing DB2 database within that subsystem.

3. Check the values specified in the CLIINI (DD-name DSNAOINI) file that the failing job is using to make sure it refers to an existing DB2 subsystem and an existing DB2 database within that subsystem.

4. Check whether the System Affinity is causing your job to execute on a system that does not contain the DB2 subsystem, MQSeries queue manager, or IBM datasets that MQSeries Integrator is trying to access.

5. In the CLIINI file (DD-name DSNAOINI), edit the following line:

   ```
   CLITRACE=0
   ```

   Change it to:

   ```
   CLITRACE=1
   ```

   Rerun your job. The CLITRACE produced (DD-name CLITRACE) is invaluable in diagnosing problems between the DB2 database and the MQSeries Integrator application. Your JCL should have a DD-statement that defines CLITRACE to either a disk file or SYSOUT class. This file is required by technical support to diagnose problems.

Note:
It is assumed that the DB2 CLI is installed, the DSNACLI Plan has been bound, and you are granted execute authority on it.

---

6. Examine all files produced by MQSeries Integrator for error or informational messages. Some error messages are written to SYSOUT, some to SYSPRINT, and some to STATLOG.
7. Look for Operating System messages that may indicate why the job has failed, such as missing files, no room to log messages (E-37, B-37 type failures), full queue conditions, and so on.
8. If failing to put or get from an MQSeries queue, make sure the queue is enabled for sharing:

   ```
   Permit shared access . . . . Y  Y=Yes,N=No
   Default share option . . . . S E=Exclusive,S=Shared
   ```

9. If the problem is related to poor Rules daemon performance, check the values of the timers specified in the input stream (DD-name SYSIN) file of the RULENG job. Setting these timers too high can result in poor performance of the Rules Engine.

When contacting technical support be prepared to send the following information via email or ftp:

- The complete listing of your jobs execution, including SYSOUTs, SYSPRINTs, STATLOG, JESMSGS, and so forth.
- The contents of the CLITRACE file
- Any dump files produced (CEEDUMP or SYSUDUMP)
- Your site’s SQLSVSES file
- Your site’s CLIINI file

Year 2000 Readiness Disclosure

MQSeries Integrator, when used in accordance with its associated documentation, is capable of correctly processing, providing, and/or receiving date information within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with this IBM program properly exchange accurate date information with it.

Customers should contact third party owners or vendors regarding the readiness status of their products.

IBM reserves the right to update the information shown here. For the latest information regarding levels of supported software, refer to: http://www.software.ibm.com/ts/mqseries/platforms/supported.html

For the latest IBM statement regarding Year 2000 readiness, refer to: http://www.ibm.com/ibm/year2000/

Chapter 2

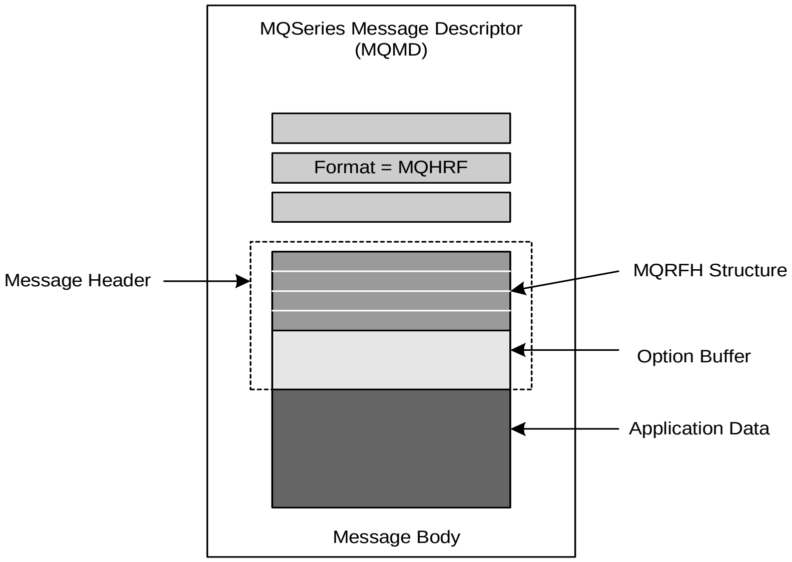
Application Programming

Rules Processing Daemon

The MQSeries Integrator Rules daemon combines MQSeries, NEONFormatter, and NEONRules in a generic server process. The MQSeries Integrator Rules daemon processes messages from one or more MQSeries input queues, uses NEONFormatter to parse messages, uses NEONRules to determine what transformations to perform and where to route the messages, then puts the output messages on MQSeries queues for delivery to applications.

Messages

MQSeries messages sent to the rules processing daemon have the following format:



MQSeries Integrator Message Format

Message Header

The first part of the message body is the message header. This header contains the application group and message type information that the NEONRules processing daemon requires to parse the message. If the NEONRules processing daemon receives a message that does not contain a message header, it assigns default values for both the application group and message type.

Applications that put messages to the NEONRules processing daemon input queue indicate that a message header precedes the application data by setting the format field of the MQMD structure to MQFMT_RF_HEADER where MQFMT_RF_HEADER is defined as the eight-character string: MQHRFbbb (b = space).

The message header consists of two parts: the MQRFH structure and the option buffer.

MQRFH Structure

The MQRFH structure contains the following fields:

Field	Description
StrucId (MQCHAR4)	The identifier for the MQRFH structure. The value must be: MQRFH_STRUC_ID = “RFHb” (b = space).
Version (MQLONG)	The identifier for Version-1 MQRFH structure. The value must be: MQRFH_VERSION_1.
StrucLength (MQLONG)	The length of the MQRFH structure and the subsequent option buffer. There is no default value for this field because the value depends on the length of the option buffer, which may be different for each message.
Encoding (MQLONG)	Encoding of the data following the MQRFH structure. The queue manager does not check the value of this field. The initial value of this field is MQENC_NATIVE.
CodedCharSetId (MQLONG)	Character set identifier of the data following the MQRFH structure. The queue manager does not check the value of this field. The initial value of this field is zero (0).
Format (MQCHAR8)	Format name of the data following the MQRFH structure. The queue manager does not check the value of this field. See the description of the Format field in the MQMD structure for more information about Format names. The initial value of this field is MQFMT_NONE.
Flags (MQLONG)	General flags.

Option Buffer

The option buffer immediately follows the MQRFH structure in the message header. The option buffer consists of a collection of space-delimited tag/value pairs. The size of the option buffer is calculated as follows:

```
OptionBufferLength = MQRFH.StrucLength - sizeof(MQRFH)
```

The data in the NEON option buffer is in the following form:

<tagname1>b<value1>b<tagname2>b<value2>b (etc.)

Tag names and values cannot contain nulls or spaces (b = space).

Recognized names in option buffer:

- OPT_APP_GRP
  Application Group
- OPT_MSG_TYPE
  Message Type
- OPT_RELOAD_RULE_SET
  Reload Rule Set
- OPT_SHUTDOWN
  Shutdown

Chapter 3

Database Abstraction Layer APIs

The Database Abstraction Layer APIs section describes functions used in NEONFormatter and NEONRules APIs for database abstraction. Database Abstraction Layer APIs provide a means of managing transactions and maintaining data integrity.

Make sure the session state is accessible by using the Ok() function. OpenDbmsSession() provides MQSeries Integrator functions a session name to associate with an MQSeries Integrator database. CloseDbmsSession() cleans up an MQSeries Integrator session and releases any residual storage that may have been allocated by MQSeries Integrator or the DBMS during a program’s execution.

APIs and Header Files

Header files contain declarations for class functions and declarations for data types and constants.

Header Files

Object Class	Description	Header File
DbmsSession	For Class DbmsSession Declarations	ses.h
	Procedural APIs for DbmsSession	sqlapi.h

DbmsSession Factory Functions

Return Type	Function	Arguments
int	Ok	()
DbmsSession*	OpenDbmsSession	(char *SessionName, int DbmsType)
DbmsSession*	OpenDbmsSession	(void* SessionHandle, int DbmsType)
DbmsSession*	OpenDbmsSession	(const char* const sessionName, const char* const configFileName, int DbmsType)
DbmsSession*	OpenDbmsSession	(const char * const serverName, const char * const userID, const char * const passwd, const char * const dbInstance, int DbmsType)
void	CloseDbmsSession	(DbmsSession* Session)

Ok

The state of the DbmsSession class.

Syntax

```
int Ok();
```

Parameters

None

Remarks

None.

Return Value

Returns 1 or TRUE for a class state that is usable, and zero (0) or FALSE if the class is not usable. A (0) value should return an error message or failure message.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession(NN_DB_TYPE_MVSDB2);
if ( !mySession || !mySession->Ok() ) {
... // Database session not created or not connected
}
```

OpenDbmsSession

OpenDbmsSession() searches the SQLSVSES configuration file for an entry named SessionName and instantiates a DbmsSession object of type DbmsType.

Syntax

```
DbmsSession* OpenDbmsSession(char *SessionName,
                             int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
SessionName	char *	Input	Identifies the session tag name in the configuration file to be used. The tag name is the first field of a line in the configuration file.
DbmsType	int	Input	Identifies the type of database to use. Supported types are defined in dbtypes.h.

Remarks

A call to OpenDbmsSession() is required prior to using any of the high-level, Formatter or Rules APIs.

Return Value

Returns a session pointer for use in other MQSeries Integrator API calls; NULL if the session object could not be allocated.

It is the application programmer’s responsibility to make sure the session state is accessible using the Ok() function. Ok() should return 1 if the session state is operational.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession("mytag",NN_DB_TYPE_MVSDB2);
if ( !mySession || !mySession->Ok() )
{
... /* Database session not created or not connected */
}
... /* Use for Rules or Formatter */
```

See Also

OpenDbmsSession(SessionHandle, DbmsType)

OpenDbmsSession(SessionName, configFileName, DbmsType)

OpenDbmsSession(serverName, userID, passwd, dbInstance, DbmsType)

CloseDbmsSession (DbmsSession * Session)

OpenDbmsSession

OpenDbmsSession() enables the user to connect to a MQSeries Integrator database using a pre-existing, database-specific, user-created (such as a Sybase DBPROC or Microsoft SQL Server handle, or Oracle LDA) session handle. DbmsType indicates the database vendor and version.

Syntax

```
DbmsSession* OpenDbmsSession(void* SessionHandle,
                             int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
SessionHandle	void *	Input	Identifier for interacting with MQSeries Integrator databases.
DbmsType	int	Input	Supported types are defined in dbtypes.h.

Remarks

A call to OpenDbmsSession() is required prior to using any of the high-level, Formatter APIs, or Rules APIs.

Return Value

Returns a session pointer for use in other MQSeries Integrator API calls; NULL if the session object could not be allocated.

It is the application programmer’s responsibility to make sure the session state is accessible using the Ok() function. Ok() should return 1 if the session state is operational.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
Lda_Def * myLda;
... // Manually log on to Oracle database
mySession = OpenDbmsSession((void *)myLda,NN_DB_TYPE_MVSDB2);
if ( !mySession || !mySession->Ok() ) {
... // Database session not created or not connected
}
... // Use for Rules or Formatter
```

See Also

OpenDbmsSession(SessionName, DbmsType)

OpenDbmsSession(SessionName, configFileName, DbmsType)

OpenDbmsSession(serverName, userID, passwd, dbInstance, DbmsType)

CloseDbmsSession (DbmsSession * Session)

OpenDbmsSession

Use this call to open a DbmsSession with a specific file other than SQLSVSES.

Syntax

```
OpenDbmsSession (const char* const sessionName,
                 const char*const configFileName,
                 int DbmsType)
```

Parameters

Name	Type	Input/ Output	Description
SessionName	const char* const	Input	Identifies the session tag name in the configuration file used. The tag name is the first field of a line in the configuration file.
configFileName	const char* const	Input	The configuration filename that has the same format as the SQLSVSES DD-name.
DbmsType	int	Input	Identifies the type of database to use. Supported data types are defined in dbtypes.h.

Remarks

The alternative configuration file must be in the same format as the SQLSVSES file. A call to OpenDbmsSession() is required prior to using any of the high-level Formatter or Rules APIs.

Return Value

If the OpenDbmsSession() call is successful, returns a currently open DbmsSession; NULL if the session object could not be allocated.

It is the application programmer's responsibility to make sure the session state is accessible using the Ok() function. Ok() should return 1 if the session state is operational.

Example

```
include dbtypes.h
DbmsSession *session = OpenDbmsSession ("oraHub",
                                       "DD:CONFFILE",
                                        NN_DB_TYPE_MVSDB2);
    if (!session)
      // handle error
```

See Also

OpenDbmsSession(SessionName, DbmsType)

OpenDbmsSession(SessionHandle, DbmsType)

OpenDbmsSession(serverName, userID, passwd, dbInstance, DbmsType)

CloseDbmsSession (DbmsSession * Session)

OpenDbmsSession

This overloaded version of OpenDbmsSession() enables the user to connect to a MQSeries Integrator database using a pre-existing database server name, user ID, password, database instance, and database type.

Syntax

```
DbmsSession* OpenDbmsSession(const char* const serverName,
                             const char* const userID,
                             const char* const passwd,
                             const char* const dbInstance,
                             int DbmsType);
```

Parameters

Name	Type	Input/ Output	Description
serverName	const char* const	Input	Server where the MQSeries Integrator database is resident.
userId	const char* const	Input	This field is not currently used, but must be specified. Specify xxxx as a placeholder for this field.
passwd	const char* const	Input	This field is not currently used, but must be specified. Specify xxxx as a placeholder for this field.
SessionHandle	void *	Input	Database session name used by MQSeries Integrator applications. This can be any string as long as it is unique.
DbmsType	int	Input	Identifies the type of database to use. Supported types are defined in dbtypes.h.

Remarks

A call to OpenDbmsSession() is required prior to using any of the high-level, Formatter APIs or Rules APIs.

Return Value

Returns a session pointer for use in other MQSeries Integrator API calls; NULL if the session object could not be allocated.

It is the application programmer's responsibility to make sure the session state is accessible using the Ok() function. Ok() should return 1 if the session state is operational.

Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession("Portland","xxxx","xxxx",
    "myHandle",NN_DB_TYPE_MVSDB2);
if ( !mySession || !mySession->Ok() )
{
.../* Database session not created or not connected */
}
.../* Use for Rules or Formatter */
```

See Also

OpenDbmsSession(SessionName, DbmsType)

OpenDbmsSession(SessionHandle, DbmsType)

OpenDbmsSession(SessionName, configFileName, DbmsType)

CloseDbmsSession (DbmsSession * Session)

CloseDbmsSession

CloseDbmsSession() cleans up a MQSeries Integrator session and releases any residual storage that may have been allocated by MQSeries Integrator during execution. Once a session is closed, use OpenDbmsSession() to establish another DBMS session.

Syntax

```
void CloseDbmsSession(DbmsSession* Session);
```

Parameters

Name	Type	Input/ Output	Description
Session	DbmsSession*	Input	Pointer to a currently open DbmsSession. Session must have been allocated using one of the OpenDbmsSession() methods.

Remarks

CloseDbmsSession() should be the last call after all MQSeries Integrator processing is complete.

Return Value

None. There are no error-handling functions for CloseDbmsSession().

Example

```
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession(...)
... // All work on open session mySession is complete

CloseDbmsSession(mySession);
```

See Also

OpenDbmsSession(SessionName, DbmsType)

OpenDbmsSession(SessionHandle, DbmsType)

OpenDbmsSession(SessionName, configFileName, DbmsType)

OpenDbmsSession(serverName, userID, passwd, dbInstance, DbmsType)

DbmsType

Identifies the database type of DbmsSession object.

Syntax

```
int DbmsType();
```

Parameters

None

Return Value

The DBMS type of DbmsSession is returned. Based on the DBMS you are using, the NN_DB_TYPE macro is set accordingly and provides the appropriate value. Complete definitions of the DBMS types is located in dbtypes.h. The dbtypes.h file must be included in the header file.

If you are using platform-specific or database-specific sections of code, you must add compiler flags at compile time.

Database Session Types

Return Value	Value	Description	Compiler Flag
NN_DB_TYPE_SYB_CT	1	Sybase ctlib	-Dsybase
NN_DB_TYPE_SYB_DB	2	Sybase dblib	-Dsybase
NN_DB_TYPE_MSSQL	4	Microsoft SQLServer	-Dmssql
NN_DB_TYPE_DB2	5	IBM DB2 ODBC CLI	-Dodbc
NN_DB_TYPE_ODBC	6	ODBC	-Dodbc
NN_DB_TYPE_MVSDB2	6	same value as ODBC in MQSeries Integrator 1.1	-Dmvs
NN_DB_TYPE_MQSERIES	7	IBM MQSeries	-Dmqseries
NN_DB_TYPE_ORA7	8	Oracle 7.3.X	-Doracle

Return Value	Value	Description	Compiler Flag
NN_DB_TYPE_ORA8	9	Oracle 8.0.X	-Doracle

Example

```
...
if (mySession->DbmsType() == NN_DB_TYPE_MVSDB2)
{
  myHandle = (DBPROCESS *)mySession->Handle();
}      ...
```

See Also

DbmsSession::Handle ()

DbmsSession (SessionName, DbmsType)

DbmsSession (sessionName, configFileName, DbmsType)

DbmsSession (serverName, UserID, passwd, dbInstance, DbmsType)

OpenDbmsSession (sessionHandle, DbmsType)

Chapter 4
buildMessage

The buildMessage routine builds messages with an MQSeries Integrator header and initializes the associated message descriptor. After calling buildMessage, the application can call MQPUT with the message descriptor and the message buffer supplied by buildMessage.

Function Declaration for buildMessage

```
int buildMessage(MQMD* md,
                 long dataLength,
                 char* data,
                 char *dataFormat
                 int *bufferLength,
                 char *buffer,
                 char *applicationGroup,
                 char *messageType,
                 int shutdown,
                 int reload)
```

Parameters

Name	Description
md	Pointer to an MQSeries message descriptor allocated by the calling application.
dataLength	Length of the application data.
data	Pointer to the application data.
dataFormat	The format of the data contained in the buffer pointed to by the data parameter.
bufferLength	The size of the buffer.

Name	Description
buffer	The pointer to the memory where buildMessage puts the MQSeries Integrator message.
applicationGroup	The application group to associate with the message. This parameter should be set to NULL when building SHUTDOWN messages.
messageType	The message type to associate with the message. This parameter should be set to NULL when building SHUTDOWN messages.
shutdown	Set to 1 for SHUTDOWN messages; 0 otherwise.
reload	Set to 1 for RELOAD messages; 0 otherwise.

Example Calls to buildMessage

To build a message with application group “TestApp” and message type “TestMsgType”, call the following buildMessage routine:

```
buildMessage(&md, dataLength, data, "MQSTR", &bufferLength,
             buffer, "TestApp", "TestMsgType", 0 , 0);
```

To build a SHUTDOWN message, call the following routine:

```
buildMessage(&md, 0, NULL, NULL, 0, NULL, NULL, NULL, 1, 0);
```

To build a RELOAD message to reload the TestApp / TestMsgType rule set, call the following routine:

```
buildMessage(&md, 0, NULL, NULL, 0, NULL, "TestApp",
             "TestMsgType", 0, 1);
```

Source Code for buildMessage

```
#include "MQSIrfheader.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int buildMessage(MQMD* md, long dataLength, char* data,
     char *dataFormat, int *bufferLength, char *buffer,
     char *applicationGroup, char *messageType,
     int shutdown, int reload)
{
     char optionBuffer[1024];
     int outputCursor = 0;
     int optionBufferLength = 0;
     MQMD tmpMd = {MQMD_DEFAULT};
     MQRFH header = {MQRFH_DEFAULT};

     memcpy(md, &tmpMd, sizeof(MQMD));
     memset (optionBuffer, 0, sizeof(optionBuffer));

     /*Construct the Option Buffer*/
     if (applicationGroup != NULL)
     {
          strcat(optionBuffer, "OPT_APP_GRP");
          strcat(optionBuffer, " ");
          strcat(optionBuffer, applicationGroup);
          strcat(optionBuffer, " ");
     }

     if (messageType != NULL)
     {
          strcat(optionBuffer, "OPT_MSG_TYPE");
          strcat(optionBuffer, " ");
          strcat(optionBuffer, messageType);
          strcat(optionBuffer, " ");
     }

     if (shutdown > 0)
```

```
    {
         strcat(optionBuffer, "OPT_SHUTDOWN");
         strcat(optionBuffer, " ");
         strcat(optionBuffer, "SHUTDOWN");
         strcat(optionBuffer, " ");
    }

    if (reload > 0)
    {
         strcat(optionBuffer, "OPT_RELOAD_RULE_SET");
         strcat(optionBuffer, " ");
         strcat(optionBuffer, "TRUE");
         strcat(optionBuffer, " ");
    }

    if (strlen(optionBuffer) > 0)
    {
         /*Remove Trailing Blank*/
         optionBufferLength = strlen(optionBuffer) - 1;
    }
    else
    {
         optionBufferLength = strlen(optionBuffer);
    }


    /*Construct the MQRFH structure*/
    header.StrucLength = sizeof(MQRFH) +
         optionBufferLength;

    if (dataFormat != NULL)
    {
         strncpy(header.Format, dataFormat,
              sizeof(header.Format));
    }


/*Make sure there is enough room in the buffer to hold*/
/*the header, options and data*/
if (*bufferLength <
    (sizeof(MQRFH) + optionBufferLength + dataLength))
{
    return (0);
}

/*Transfer header, options, and data to the message */
/* buffer */
memcpy(buffer + outputCursor, &header, sizeof(MQRFH));
outputCursor += sizeof(MQRFH);
memcpy(buffer + outputCursor, optionBuffer,
       optionBufferLength);
outputCursor += optionBufferLength;
if (data != NULL)
{
     memcpy(buffer + outputCursor, data, dataLength);
     outputCursor += dataLength;
}
else
     {
       return(0);
     }

/*Return the size of the header + options + data*/
*bufferLength = outputCursor;

/*Set the message descriptor format field            */
/*to indicate that an MQIntegrator header is present */
/*in the message buffer.                             */
strncpy(md->Format, "MQHRF   ", sizeof(md->Format));

return(1);
}
```

Sample Application Using buildMessage

The following source code is from the AMQSPUT0 MQSeries sample application and is modified to use the buildMessage routine. The program functions the same as AMQSPUT0, except it prepends an MQSeries Integrator header to each message that is sends. The program assigns the application group TestApp and the message type TestMsg to each message that it puts.

```
/********************************************************************/
/*                                                                  */
/* Program name: AMQSPUT0                                           */
/*                                                                  */
/* Description: Sample C program that puts messages to              */
/*              a message queue (example using MQPUT)               */
/*                                                                  */
/*  Statement:     Licensed Materials - Property of IBM             */
/*                                                                  */
/*                 84H2000, 5765-B73                                */
/*                 84H2001, 6539-B42                                */
/*                 84H2002, 5765-B74                                */
/*                 84H2003, 5765-B75                                */
/*                 84H2004, 6539-B43                                */
/*                 (C) Copyright IBM Corp. 1994, 1997               */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/* Function:                                                        */
/*                                                                  */
/*                                                                  */
/*   AMQSPUT0 is a sample C program to put messages on a message   */
/*   queue, and is an example of the use of MQPUT.                  */
/*                                                                  */
/*      -- messages are sent to the queue named by the parameter    */
/*                                                                  */
/*      -- gets lines from StdIn, and adds each to target           */
/*         queue, taking each line of text as the content           */
/*         of a datagram message; the sample stops when a null     */
/*         line (or EOF) is read.                                  */
/*         New-line characters are removed.                        */
/*         If a line is longer than 99 characters it is broken up  */
/*         into 99-character pieces. Each piece becomes the        */
/*         content of a datagram message.                          */
/*         If the length of a line is a multiple of 99 plus 1, for */
/*         example 199, the last piece will only contain a new-line*/
/*         character so will terminate the input.                  */
/*                                                                 */
/*      -- writes a message for each MQI reason other than         */
/*         MQRC_NONE; stops if there is a MQI completion code      */
/*         of MQCC_FAILED                                          */
/*                                                                 */
/*    Program logic:                                               */
/*         MQOPEN target queue for OUTPUT                          */
/*         while end of input file not reached,                    */
/*         .  read next line of text                               */
/*         .  MQPUT datagram message with text line as data        */
/*         MQCLOSE target queue                                    */
/*                                                                 */
/*                                                                 */
/*******************************************************************/
/*                                                                 */
/*   AMQSPUT0 has 2 parameters                                     */
/*                 - the name of the target queue (required)       */
/*                 - queue manager name (optional)                 */
/*                                                                 */
/*******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   /* includes for MQI                                             */
#include <cmqc.h>
int main(int argc, char **argv)
{
  /*  Declare file and character for sample input                  */
  FILE *fp;
  /*   Declare MQI structures needed                               */
  MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor            */
  MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor           */
  MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options          */
```

```
   /** note, sample uses defaults where it can **/
MQHCONN  Hcon;                  /* connection handle            */
MQHOBJ   Hobj;                  /* object handle                */
MQLONG   O_options;             /* MQOPEN options               */
MQLONG   C_options;             /* MQCLOSE options              */
MQLONG   CompCode;              /* completion code              */
MQLONG   OpenCode;              /* MQOPEN completion code       */
MQLONG   Reason;                /* reason code                  */
MQLONG   CReason;               /* reason code for MQCONN       */
MQLONG   buflen;                /* buffer length                */
char     buffer[100];           /* message buffer               */
char     QMName[50];            /* queue manager name           */

/* buffer to hold MQIntegrator Header and Message data */
char     newBuffer[1024];
/* size of new buffer */
int      newBufferLength = 1024;
printf("Sample AMQSPUT0 start\n");
if (argc < 2)
{
  printf("Required parameter missing - queue name\n");
  exit(99);
}

/******************************************************************/
/*                                                                */
/*   Connect to queue manager                                     */
/*                                                                */
/******************************************************************/
QMName[0] = 0;    /* default */
if (argc > 2)
  strcpy(QMName, argv[2]);
MQCONN(QMName,                 /* queue manager                 */
       &Hcon,                  /* connection handle             */
       &CompCode,              /* completion code               */
       &CReason);              /* reason code                   */
/* report reason and stop if it failed */
if (CompCode == MQCC_FAILED)
{
  printf("MQCONN ended with reason code %ld\n", CReason);
  exit( (int)CReason );
}
```

```
/*****************************************************************/
/*                                                               */
/*   Use parameter as the name of the target queue               */
/*                                                               */
/*****************************************************************/
strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
printf("target queue is %s\n", od.ObjectName);
/*****************************************************************/
/*                                                               */
/*   Open the target message queue for output                    */
/*                                                               */
/*****************************************************************/
O_options = MQOO_OUTPUT           /* open queue for output       */
        + MQOO_FAIL_IF_QUIESCING; /* but not if MQM stopping     */
MQOPEN(Hcon,                      /* connection handle           */
       &od,                       /* object descriptor for queue */
       O_options,                 /* open options                */
       &Hobj,                     /* object handle               */
       &OpenCode,                 /* MQOPEN completion code      */
       &Reason);                  /* reason code                 */
/* report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %ld\n", Reason);
}
if (OpenCode == MQCC_FAILED)
{
  printf("unable to open queue for output\n");
}
/*****************************************************************/
/*                                                               */
/*   Read lines from the file and put them to the message queue  */
/*   Loop until null line or end of file, or there is a failure  */
/*                                                               */
/*****************************************************************/
CompCode = OpenCode;        /* use MQOPEN result for initial test*/
fp = stdin;
memcpy(md.Format,          /* character string format            */
       MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);
while (CompCode != MQCC_FAILED)
{
  if (fgets(buffer, sizeof(buffer), fp) != NULL)
```

```
    {
      buflen = strlen(buffer);       /* length without null        */
      if (buffer[buflen-1] == '\n')  /* last char is a new-line    */
      {
        buffer[buflen-1]  = '\0';    /* replace new-line with null */
        --buflen;                    /* reduce buffer length       */
      }
    }
    else buflen = 0;        /* treat EOF same as null line         */
    /****************************************************************/
    /*                                                              */
    /*   Put each buffer to the message queue                       */
    /*                                                              */
    /****************************************************************/
    if (buflen > 0)
    {
      memcpy(md.MsgId,           /* reset MsgId to get a new one   */
             MQMI_NONE, sizeof(md.MsgId) );
      memcpy(md.CorrelId,        /* reset CorrelId to get a new one*/
             MQCI_NONE, sizeof(md.CorrelId) );

     buildMessage(&md, buflen, buffer, "MQSTR",  &newBufferLength,
                  newBuffer, "TestApp", "TestMsg",0, 0);
      MQPUT(Hcon,                /* connection handle              */
            Hobj,                /* object handle                  */
            &md,                 /* message descriptor             */
            &pmo,                /* default options (datagram)     */
            newBufferLength,     /* buffer length with MQIntegrator*/
                                 /* header                         */
            newBuffer,           /* message buffer with            */
                                 /* MQIntegrator header            */
            &CompCode,           /* completion code                */
            &Reason);            /* reason code                    */
      /* report reason, if any */
      if (Reason != MQRC_NONE)
      {
        printf("MQPUT ended with reason code %ld\n", Reason);
      }
    }
    else   /* satisfy end condition when empty line is read        */
      CompCode = MQCC_FAILED;
  }
```

```
/******************************************************************/
/*                                                                */
/*   Close the target queue (if it was opened)                    */
/*                                                                */
/******************************************************************/
if (OpenCode != MQCC_FAILED)
{
  C_options = 0;                 /* no close options           */
  MQCLOSE(Hcon,                  /* connection handle          */
          &Hobj,                 /* object handle              */
          C_options,
          &CompCode,             /* completion code            */
          &Reason);              /* reason code                */
  /* report reason, if any */
  if (Reason != MQRC_NONE)
  {
    printf("MQCLOSE ended with reason code %ld\n", Reason);
  }
}
/******************************************************************/
/*                                                                */
/*   Disconnect from MQM if not already connected                 */
/*                                                                */
/******************************************************************/
if (CReason != MQRC_ALREADY_CONNECTED)
{
  MQDISC(&Hcon,                  /* connection handle          */
         &CompCode,              /* completion code            */
         &Reason);               /* reason code                */
  /* report reason, if any */
  if (Reason != MQRC_NONE)
  {
    printf("MQDISC ended with reason code %ld\n", Reason);
  }
}
/******************************************************************/
/*                                                                */
/* END OF AMQSPUT0                                                */
/*                                                                */
/******************************************************************/
printf("Sample AMQSPUT0 end\n");
return(0); }
```

Appendix A

Notices

You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and Service Marks

Index

Sending your comments to IBM
MQSeries Integrator
Application Development Guide
SC34-5508-02

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book only and the way in which the information is presented.

To request additional publications or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
  - From outside the U.K., use your international access code followed by 44 1962 870229
  - From within the U.K., use 01962 870229

Electronically, use the appropriate network ID:

- IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
- IBMLink: HURSLEY(IDRCF)
- Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic number to which your comment applies
- Your name / address / telephone number / fax number / network ID

Readers' Comments
MQSeries Integrator
Application Development Guide
SC34-5508-02

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States of America
Belgium	France	Israel	New Zealand	Spain	
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP 095)
Hursley Park
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name ____________________
Company or Organization ____________________
Address ____________________
____
EMAIL ____________________
Telephone ____________________

1 Cut along this line

4 Fasten here with adhesive tape

Printed in U.S.A

SC34-5508-02