

MQSeries® Integrator

# **Application Development Guide**

Version 1.0

**Note:** Before using this information and the product it supports, be sure to read the general information under Appendix B entitled “Notices”.

**First edition (January 1999)**

This edition applies to IBM® MQSeries Integrator, Version 1.0 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.

© Copyright New Era of Networks, Inc., 1998, 1999. All rights reserved.

© Copyright International Business Machines Corporation, 1999. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

---

<b>Chapter 1: Introduction .....</b>	<b>1</b>
MQSeries Integrator Overview .....	1
MQSeries .....	1
NEONFormatter.....	1
NEONRules .....	2
MQSeries Integrator Rules Daemon .....	2
Product Documentation Set .....	3
Supported Platforms and Compilers .....	4
Disk Space and Memory Requirements .....	5
<b>Chapter 2: Application Programming .....</b>	<b>7</b>
Rules Processing Daemon.....	7
Messages.....	7
NEON Header .....	8
MQRFH Structure.....	8
NEON Option Buffer.....	9
<b>Chapter 3: Database Abstraction Layer</b>	
<b>APIs.....</b>	<b>11</b>
APIs and Header Files.....	11
<b>Appendix A: buildMessage .....</b>	<b>23</b>
Source Code for buildMessage .....	24
Sample Application Using buildMessage .....	26
<b>Appendix B: Notices.....</b>	<b>33</b>
Trademarks and Service Marks .....	34
<b>Index.....</b>	<b>37</b>



## MQSeries Integrator Overview

MQSeries Integrator provides the flexibility and scalability that allows true application integration. MQSeries Integrator consists of four components:

- MQSeries
- NEONFormatter
- NEONRules
- MQSeries Integrator Rules daemon

### MQSeries

MQSeries is 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 of work are processed.

### NEONFormatter

NEONFormatter translates messages from one format to another. Formatter handles multiple message format types from multiple data value sources with the ability to convert and parse messages. Messages can be converted from any described format to any other described format (if fields in input data formats are missing, you can set up defaults for those fields on output). When a message is provided as input to Formatter, the message is parsed and data values are returned. Formatter can handle virtually any message format, including fixed (for example, COBOL records), delimited (for example, C null delimited strings), and variable, tagged, delimited, repetitive and recursive formats (for example, S.W.I.F.T. messages).

Defining message formats in Formatter's database is done 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.

## NEONRules

NEONRules lets you develop rules for managing message destination IDs, receiver locations, expected message formats, and any processes initiated upon message delivery. The creation and dispatch of multiple messages to multiple destinations from a single input message is supported, and different formats and transport methods for each is allowed. The dynamic nature of the Rules Engine means that rules can be effective immediately, staged over time, or delayed, depending on how the reload messages are timed, allowing flexibility in rapidly changing environments.

NEONRules can examine the value of any field or group of fields in a message to make its determinations. It can aggregate conditions with the Boolean AND and OR operators without architectural limits as to the number or complexity of the expressions.

## MQSeries Integrator Rules Daemon

The MQSeries Integrator Rules daemon combines MQSeries, Formatter, and Rules in a generic server process. The MQSeries Integrator Rules daemon processes messages from an MQSeries input queue, uses Formatter to parse messages, uses Rules 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.

# Product Documentation Set

The MQSeries Integrator documentation set includes:

- ***MQSeries Integrator Installation and Configuration Guide*** helps end users and engineers install and configure MQSeries Integrator.
- ***MQSeries Integrator User's Guide*** helps MQSeries Integrator users understand and apply the program through its graphical user interfaces (GUIs).
- ***MQSeries Integrator System Management Guide*** is for system administrators and database administrators who work with MQSeries Integrator on a day-to-day basis.
- ***MQSeries Integrator Application Development Guide*** assists programmers in writing applications that use MQSeries Integrator APIs.
- ***Programming References*** are intended for users who build and maintain the links between MQSeries Integrator and other applications. The documents include:
  - ***MQSeries Integrator Programming Reference for NEONFormatter*** is a reference to Formatter APIs for those who write applications to translate messages from one format to another.
  - ***MQSeries Integrator Programming Reference for NEONRules*** is a reference to Rules APIs for those who write applications to perform actions based on message contents.

## Note

For information on message queuing, refer to the ***IBM MQSeries*** documentation.

---

# Supported Platforms and Compilers

<b>Operating System</b>	<b>DBMS</b>	<b>Compiler</b>
Windows NT 4.0	DB2 5.0 Oracle 7.3 Oracle 8 SQL Server 6.5 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	Microsoft Visual C++ version 4.2
Solaris 2.5.1, 2.6	DB2 5.0 Oracle 7.3 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	Sparcworks C++ compiler version 4.0
HP-UX 10.20	DB2 5.0 Oracle 7.3 Oracle 8 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	HP C++ version 10.34
AIX 4.2	DB2 5.0 Oracle 7.3 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	IBM C Set ++ version 3.1.4



# Disk Space and Memory Requirements

Required disk space is a function of the number of queues, formats, and rules. Recommended memory for satisfactory performance depends on message rates, message sizes, and application-specific factors. For Windows NT/SQLServer, the recommended memory is 128 MB; for other platforms, the recommended memory is 256 MB.

<b>Operating System</b>	<b>DBMS</b>	<b>Libraries &amp; Executables</b>
AIX 4.2	DB2 5.0	144 MB
	Oracle 7.3	117 MB
	Sybase 11.03	130 MB
	Sybase 11.5	130 MB
HP-UX 10.20	DB2 5.0	169 MB
	Oracle 7.3	117 MB
	Oracle 8	117 MB
	Sybase 11.03	120 MB
	Sybase 11.5	120 MB
Solaris 2.5.1, 2.6	DB2 5.0	166 MB
	Oracle 7.3	117 MB
	Sybase 11.03	120 MB
	Sybase 11.5	120 MB
Windows NT 4.0	DB2 5.0	125 MB
	Oracle 7.3	117 MB
	Oracle 8	117 MB
	SQLServer 6.5	120 MB
	Sybase 11.03	120 MB
	Sybase 11.5	120 MB

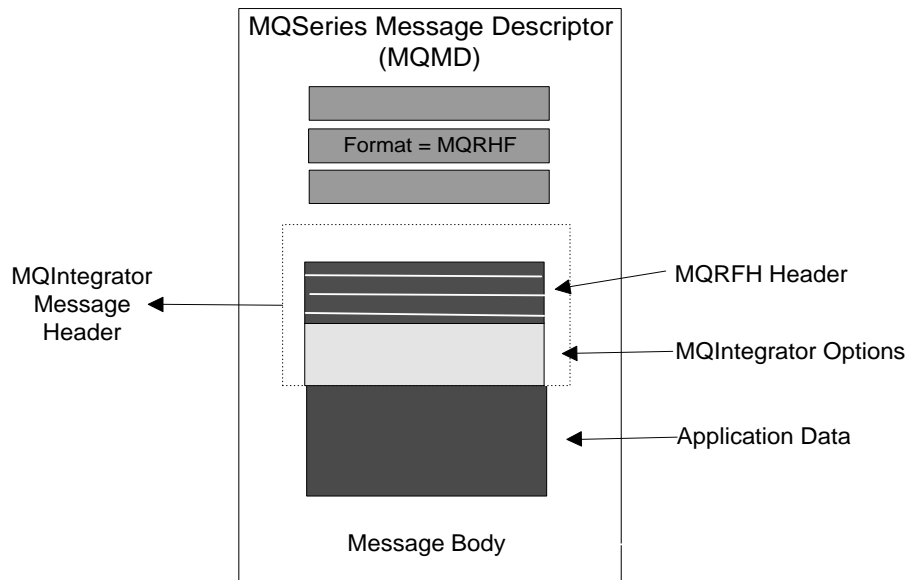


## Rules Processing Daemon

The MQSeries Integrator Rules daemon combines MQSeries, Formatter, and Rules in a generic server process. The MQSeries Integrator Rules daemon processes messages from an MQSeries input queue, uses Formatter to parse messages, uses Rules 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*

## NEON Header

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

Applications that put messages to the Rules processing daemon input queue indicate that a NEON 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: MQRFHbbb (b = space).

The NEON header consists of two parts: the MQRFH structure and the NEON 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.
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.

## NEON Option Buffer

The NEON option buffer immediately follows the MQRFH structure in the NEON header. The option buffer consists of a collection of space-delimited tag/value pairs. The size of the NEON option buffer can be 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 b (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 Formatter and Rules 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 a 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
DbmsSession*	OpenDbmsSession	(char *SessionName, int DbmsType)
DbmsSession*	OpenDbmsSession	(void* SessionHandle, int DbmsType)
DbmsSession*	OpenDbmsSession	(const char* const sessionName, const char* const configFileName, int DbmsType)

<b>Return Type</b>	<b>Function</b>	<b>Arguments</b>
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

### Overview

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.

### Example

```
...  
DbmsSession *mySession;  
... // logon to the Sybase database  
mySession = OpenDbmsSession("neonSYB", SYBASE49);  
if ( !mySession || !mySession->Ok() ) {  
    ... // Database session not created or not connected  
}
```

## OpenDbmsSession

### Overview

OpenDbmsSession searches the sqlsvses.cfg configuration file for an entry named SessionName and instantiates a DbmsSession object of type DbmsType. The sqlsvses.cfg file must be in the same directory as the program executable file.

### 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 could not be created.

It is the application programmer's responsibility to make sure the session state is accessible using the Ok() function. Ok() should return a zero (0) if the session state is operational.

## Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession("mytag",ORACLE7);
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 ()

## OpenDbmsSession

### Overview

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. For example, this is the DBPROC handle for Sybase and SQL Server, and LDA for Oracle.
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 unable to associate SessionHandle with a MQSeries Integrator database handle.

It is the application programmer's responsibility to make sure the session state is accessible using the Ok() function. Ok() should return a zero (0) 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,ORACLE7);
if ( !mySession || !mySession->Ok() ) {
... // Database session not created or not connected
}
... // Use for Rules or Formatter
```

## See Also

OpenDbmsSessions(sessionName, DbmsType)

OpenDbmsSession(SessionName, configFileName, DbmsType)

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

CloseDbmsSession()

## OpenDbmsSession

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

### 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 to be used. The tag name is the first field of a line in the configuration file.
configFileName	const char* const	Input	The configuration file name that has the same format as the sqlsvses.cfg default file 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.cfg file.

### Return Value

If the OpenDbmsSession call is successful, returns a currently open DbmsSession; if the call fails, returns NULL.

### Example

```
DbmsSession *session = OpenDbmsSession ("oraHub",
    "configFile.txt", NN_DBMS_TYPE_ORA);
    if (!session)
        // handle error
```

### See Also

OpenDbmsSession(SessionName, DbmsType)

OpenDbmsSession(SessionHandle, DbmsType)

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

CloseDbmsSession()

## OpenDbmsSession

### Overview

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	The database user name.
passwd	const char* const	Input	The database password.
SessionHandle	void *	Input	Database session name to be 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 <code>dbtypes.h</code> .

### 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 unable to associate the provided information with a session.

It is the application programmer's responsibility to make sure the session state is accessible using the `Ok()` function. `Ok()` should return a zero (0) if the session state is operational.

## Example

```
...
#include "dbtypes.h"
DbmsSession *mySession;
mySession = OpenDbmsSession("Portland", "Reno", "Denver",
    "Atlanta", ORACLE7);
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(serverName, userID, passwd, dbInstance, DbmsType)

CloseDbmsSession()



## CloseDbmsSession

### Overview

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

```
... // All work on open session mySession is complete
CloseDbmsSession(mySession);
```

### See Also

OpenDbmsSession(char \*sessionName, int DbmsType)

OpenDbmsSession(void\*SessionHandle, int DbmsType)

OpenDbmsSession(const char \* const serverName, const char \* const userID, const char \* const passwd, const char \* const dbInstance, int DbmsType)



---

## Appendix A

# 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,
                char* data,
                long dataLength,
                char *dataFormat,
                int *bufferLength,
                char *buffer,
char *applicationGroup,
                char *messageType,
                int shutdown,
                int reload)
```

### **Parameter Descriptions for buildMessage**

<b>Parameter</b>	<b>Description</b>
md	Pointer to an MQSeries message descriptor allocated by the calling application.
data	Pointer to the application data.
dataLength	Length of the application data.
dataFormat	The format of the data contained in the buffer pointed to by the data parameter.
buffer	The pointer to the memory where buildMessage will put the MQSeries Integrator message.
bufferLength	The size of the buffer.
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 buildMessage routine as follows:

```
buildMessage(&md, dataLength, data, "MQSTR",
            bufferLength, buffer, "TestApp", "testMsgType", 0.0);
```

To build a SHUTDOWN message, call the routine as follows:

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

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

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

# Source Code for buildMessage

```
#include "MQSInfheader.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 amqspu0.c MQSeries sample application and is modified to use the buildMessage routine. The program functions the same as amqspu0, except it prepends an MQSeries Integrator header to each message that it 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 */
/* e.g. 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,
        &Hcon,
        &CompCode,
        &CReason);
/* 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 B

# Notices

---

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other 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 NON-INFRINGEMENT, 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.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this document to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those

Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,  
Mail Point 151,  
Hursley Park,  
Winchester,  
Hampshire,  
England,  
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. 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

The following, which appear in this book or other MQSeries Integrator books, are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

MQSeries  
AIX  
DB2  
IBM

NEONFormatter and NEONRules are trademarks of New Era of Networks, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

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





---

# Index

---

## A

application programming 7

## B

buildMessage 23  
  samples application 26  
  source code 24

## C

CloseDbmsSession 11, 21  
compilers 4

## D

Database Abstraction Layer APIs 11  
Database Abstraction Layer  
  DbmsSession Factory functions 11  
  header files 11  
DbmsSession Factory functions 11  
disk space requirements 5  
documentation set 3

## F

Formatter 1

## H

header files 11  
headers 8  
  MQRFH structure 8  
  NEON option buffer 9

## M

memory requirements 5  
messages 7, 8  
MQRFH structure 8  
MQSeries 1  
MQSeries Integrator components  
  MQSeries 1  
  MQSeries Integrator Rules daemon 2  
  NEONFormatter 1  
  NEONRules 2  
MQSeries Integrator overview 1  
MQSeries Integrator Rules daemon 2, 7  
MQSeries messages 7

## N

NEON header 8  
  MQRFH structure 8  
  NEON option buffer 9  
NEON option buffer 9  
NEONFormatter 1  
NEONRules 2

## O

OK function 13  
OpenDbmsSession 11, 14, 16, 18, 19  
option buffer 9  
overview 1

## P

platforms 4  
programming applications 7

## R

Rules 2  
rules processing daemon 7

## S

sample application for buildMessage 26  
ses.h 11  
source code for buildMessage 24  
supported compilers and platforms 4



## **Sending your comments to IBM**

### **MQSeries Integrator**

### **Application Development Guide**

### **SC34-5508-00**

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 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

