WebSphere MQ

Using .NET

Version 5 Release 3

GC34-6328-02

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix B, "Notices," on page 77.

Third edition (January 2004)

This is the third edition of this book. It applies to IBM WebSphere MQ classes for .NET Version 5.3, and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this book

This book describes:

- WebSphere MQ classes for .NET, which can be used to access WebSphere MQ systems

Note: Consult the README file for information that expands and corrects information in this book. The README file is installed with the WebSphere MQ classes for .NET code and can be found in the `doc` subdirectory.

Who this book is for

This information is written for programmers who are familiar with the procedural WebSphere MQ application programming interface as described in the *WebSphere MQ Application Programming Guide*. It shows how to transfer this knowledge to become productive with the WebSphere MQ classes for .NET programming interfaces.

What you need to know to understand this book

You need:

- Knowledge of the .NET programming environment
- Understanding of the purpose of the message queue interface (MQI) as described in the *WebSphere MQ Application Programming Guide* and the chapter about Call Descriptions in the *WebSphere MQ Application Programming Reference*
- Experience of WebSphere MQ programs in general, or familiarity with the content of the other WebSphere MQ publications

How to use this book

Part 1 of this book tells you how to use WebSphere MQ classes for .NET; Part 2 helps programmers wanting to use WebSphere MQ classes for .NET.

First, read the topics in Part 1 that introduce you to WebSphere MQ classes for .NET. Then use the programming guidance in Part 2 to understand how to use the classes to send and receive WebSphere MQ messages in the .NET environment.

Remember to check the README file installed with the WebSphere MQ classes for .NET code for later or more specific information.

Terms used in this book

The terms **WebSphere MQ** and **WebSphere MQ for Windows® systems** mean WebSphere MQ running on the Windows platforms:

- Windows NT®
- Windows 2000
- Windows XP

We also use the term **Windows systems** or just **Windows** as general terms for these Windows platforms.

Summary of changes

This section describes changes in this edition of *WebSphere MQ Using .NET*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (GC34–6328–02)

The changes to this edition of *Using .NET* include:

- The correction of the prerequisite level of the .NET Framework.

Changes for previous edition (GC34–6328–01)

The changes to this edition of *Using .NET* include:

- The correction of NMQ_MQ_LIB values in "Defining which connection to use" on page 15.

Changes

Part 1. Guidance for users

Chapter 1. Getting started

This topic gives an overview of WebSphere MQ classes for .NET and their uses.

What are WebSphere MQ classes for .NET?

WebSphere MQ classes for .NET allow a program written in the .NET programming framework to:

- Connect to WebSphere MQ as a WebSphere MQ client
- Connect directly to a WebSphere MQ server

WebSphere MQ classes for .NET enable .NET applications to issue calls and queries to WebSphere MQ. This gives access to mainframe and legacy applications, typically over the Internet, without necessarily having any other WebSphere MQ code on the client machine. With WebSphere MQ classes for .NET, Internet users can become true participants in transactions, rather than just givers and receivers of information.

Who should use WebSphere MQ classes for .NET?

If your enterprise fits any of the following scenarios, you can gain significant advantage by using WebSphere MQ classes for .NET:

- A medium or large enterprise that is introducing intranet-based client/server solutions. Here, Internet technology provides low cost easy access to global communications; WebSphere MQ connectivity provides high integrity with assured delivery and time independence.
- A medium or large enterprise with a need for reliable business-to-business communications with partner enterprises. Here again, the Internet provides low-cost easy access to global communications; WebSphere MQ connectivity provides high integrity with assured delivery and time independence.
- A medium or large enterprise that wants to provide access from the public Internet to some of its enterprise applications. Here, the Internet provides global reach at a low cost; WebSphere MQ connectivity provides high integrity through the queuing paradigm. In addition to low cost, the business can achieve improved customer satisfaction through 24 hour a day availability, fast response, and improved accuracy.
- An Internet Service provider, or other Value Added Network provider. These companies can exploit the low cost and easy communications provided by the Internet. They can also add value with the high integrity provided by WebSphere MQ connectivity. An Internet Service provider that exploits WebSphere MQ can immediately acknowledge receipt of input data from a Web browser, guarantee delivery, and provide an easy way for the user of the Web browser to monitor the status of the message.

WebSphere MQ classes for .NET provide an excellent infrastructure to access enterprise applications and develop complex Web applications. A service request from a Web browser can be queued then processed when possible, allowing a timely response to be sent to the end user, regardless of system loading. By placing this queue *close* to the user in network terms, the load on the network does not impact the timeliness of the response. Also, the transactional nature of WebSphere

MQ messaging means that a simple request from the browser can be expanded safely into a sequence of individual back end processes in a transactional manner.

Connection options

The following sections describe these options in more detail.

Client bindings connection

To useWebSphere MQ classes for .NET as a WebSphere MQ client, you can install it either on the WebSphere MQ server machine, or on a separate machine.

Server bindings connection

When used in server bindings mode, WebSphere MQ classes for .NET uses the queue manager API, rather than communicating through a network. This provides better performance for WebSphere MQ applications than using network connections.

To use the bindings connection, you must install WebSphere MQ classes for .NET on the WebSphere MQ server.

Prerequisites

To run WebSphere MQ classes for .NET, you need the following software:

- WebSphere MQ for Windows systems.
- Microsoft® .NET Framework (v1.1.4322 or later).

Check the README file for the latest information about operating system levels this product has been tested against.

Chapter 2. Installation

This topic tells you how to install the WebSphere MQ classes for .NET code.

How to install

WebSphere MQ classes for .NET are supplied as part of CSD 5. There is no separate installation procedure beyond that for the CSD.

Note: The .NET Framework is a prerequisite for WebSphere MQ classes for .NET but is not a prerequisite for any other part of CSD 5.

What is installed

The latest version of WebSphere MQ classes for .NET is installed. The class libraries are implemented in the single file `amqmdnet.dll`.

We also supply sample applications, including source; see “Sample applications” on page 7

Note: Sample applications are only installed as part of the CSD installation if samples were previously installed.

What is installed

Chapter 3. Using WebSphere MQ classes for .NET

This topic tells you how to:

- Configure your system to run the sample programs to verify your WebSphere MQ classes for .NET installation.
- Modify the procedures to run your own programs.

Remember to check the README file installed with the WebSphere MQ classes for .NET code for later or more specific information for your environment.

The procedures depend on the connection option you want to use. Follow the instructions in the section that is appropriate for your requirements.

Configuring your queue manager to accept client connections

Use the following procedures to configure your queue manager to accept incoming connection requests from the clients.

TCP/IP client

1. Define a server connection channel using the following procedures:
   a. Start your queue manager by using the `strmqm` command.
   b. Type the following command to start the runmqsc program:
      ```
      runmqsc [QMNAME]
      ```
   c. Define a sample channel called NET.CHANNEL by issuing the following command:
      ```
      DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for WebSphere MQ classes for .NET')
      ```
2. Start a listener program with the following commands:
   ```
   runmqlsr -t tcp [-m QMNAME] -p 1414
   ```

   Note: If you use the default queue manager, you can omit the -m option.

Sample applications

Three sample applications are supplied:

- a simple put message application
- a simple get message application
- a ’hello world’ application

″Put message″ program SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)

This program shows how to put a message to a named queue. The program has 3 parameters:

- The name of a queue (required) e.g. SYSTEM.DEFAULT.LOCAL.QUEUE
- The name of a queue manager (optional)
- The definition of a channel (optional) e.g. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

If no queue manager name is given, the queue manager defaults to the default local queue manager. If a channel is defined, it should have the same format as the MQSERVER environment variable.

″Get message″ program SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
This program shows how to get a message from a named queue. The program has 3 parameters:

- The name of a queue (required) e.g. SYSTEM.DEFAULT.LOCAL.QUEUE
- The name of a queue manager (optional)
- The definition of a channel (optional) e.g. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

If no queue manager name is given, the queue manager defaults to the default local queue manager. If a channel is defined, it should have the same format as the MQSERVER environment variable.

″Hello World″ program (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
This program shows how to put and get a message. The program has 3 parameters:

- The name of a queue (optional) e.g. SYSTEM.DEFAULT.LOCAL.QUEUE or SYSTEM.DEFAULT.MODEL.QUEUE
- the name of a queue manager (optional)
- A channel definition (optional) e.g. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

If no queue name is given, the name defaults to SYSTEM.DEFAULT.LOCAL.QUEUE. If no queue manager name is given, the queue manager defaults to the default local queue manager.

You can verify your installation by compiling and running these applications.

The sample applications will be installed to the following locations, according to the language in which they are written, where *mqmtop* represents the high-level directory in which the product has been installed:

C#

mqmtop\Tools\dotnet\samples\cs\nmqswrld.cs

mqmtop\Tools\dotnet\samples\cs\nmqsput.cs

mqmtop\Tools\dotnet\samples\cs\nmqsget.cs

Managed C++

mqmtop\Tools\dotnet\samples\mcp\mmqswrld.cs

mqmtop\Tools\dotnet\samples\mcp\mmqsput.cs

mqmtop\Tools\dotnet\samples\mcp\mmqsget.cs

Visual Basic

mqmtop\Tools\dotnet\samples\mcp\vmqswrld.cs

mqmtop\Tools\dotnet\samples\mcp\vmqsput.cs

mqmtop\Tools\dotnet\samples\mcp\vmqsget.cs

mqmtop\Tools\dotnet\samples\mcp\xmqswrld.cs

mqmtop\Tools\dotnet\samples\mcp\xmqsput.cs

mqmtop\Tools\dotnet\samples\mcp\xmqsget.cs

To build the sample applications a batch file has been supplied for each language.

C#

mqmtop\Tools\dotnet\samples\cs\bldcssamp.bat

The bldcssamp contains a line for each sample, which is all that is necessary to build this sample program:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib:mqmtop\bin
/out:nmqwrld.exe nmqwrld.cs
```

Managed C++

mqmtop\Tools\dotnet\samples\mcp\bldmcpsamp.bat

The bldmcpamp contains a line for each sample, which is all that is necessary to build this sample program:

```
cl /clr mqmtop\bin mmqwrld.cpp
```

Visual Basic

mqmtop\Tools\dotnet\samples\vb\bldvbsamp.bat

The bldcssamp contains a line for each sample, which is all that is necessary to build this sample program:

```
vbc /r:System.dll /r:mqmtop\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Running your own WebSphere MQ .NET programs

To run your own .NET applications, use the procedures described for the verification programs, substituting your application name in place of the sample applications

For information on writing WebSphere MQ classes for .NET applications, see Part 2, “Programming with WebSphere MQ classes for .NET,” on page 11.

Solving WebSphere MQ .NET problems

If a program does not complete successfully, run one of the sample applications, and follow the advice given in the diagnostic messages. These sample applications are described in Chapter 3, “Using WebSphere MQ classes for .NET,” on page 7.

If the problems continue and you need to contact the IBM® service team, you might be asked to turn on the trace facility.

Tracing the sample application

For instructions on using the trace facility, refer to “Tracing WebSphere MQ .NET programs” on page 26.

Error messages

You might see the following common error message:

An unhandled exception of type ’System.IO.FileNotFoundException’ occurred in unknown module

If this error occurs, either ensure the amqmdnet.dll is registered in the ’Global Assembly Cache’ or create a configuration file that points to the amqmdnet.dll assembly.

Error messages

Part 2. Programming with WebSphere MQ classes for .NET

Chapter 4. Introduction for programmers

This topic contains general information for programmers. For more detailed information about writing programs, see Chapter 5, “Writing WebSphere MQ .NET programs,” on page 15.

Why should I use the .NET interface?

If you have applications which use Microsoft’s .NET Framework and wish to take advantage of the facilities of WebSphere MQ, you must use WebSphere MQ classes for .NET.

The WebSphere MQ .NET interface

The procedural WebSphere MQ application programming interface is built around the following verbs:

```
MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
```

These verbs all take, as a parameter, a handle to the WebSphere MQ object on which they are to operate. Because .NET is object-oriented, the .NET programming interface turns this round. Your program consists of a set of WebSphere MQ objects, which you act upon by calling methods on those objects.

When you use the procedural interface, you disconnect from a queue manager by using the call MQDISC(*Hconn*, CompCode, Reason), where *Hconn* is a handle to the queue manager.

In the .NET interface, the queue manager is represented by an object of class MQQueueManager. You disconnect from the queue manager by calling the Disconnect() method on that class.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

Compiling WebSphere MQ .NET applications

Before you can compile any applications that you write, you must have access to a .NET Framework, as detailed in “Prerequisites” on page 4.

WebSphere MQ classes for .NET class library

WebSphere MQ classes for .NET is a set of classes that enable .NET applications to interact with WebSphere MQ.

The following classes are provided:
- MQChannelDefinition
- MQEnvironment
- MQException

- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQPutMessageOptions
- MQQueue
- MQQueueManager

The following structure is provided:
- MQC

Chapter 5. Writing WebSphere MQ .NET programs

To use WebSphere MQ classes for .NET to access WebSphere MQ queues, you write programs in any language supported by .NET containing calls that put messages onto, and get messages from, WebSphere MQ queues.

This chapter provides information to assist with writing applications to interact with WebSphere MQ systems. For details of individual classes, see Chapter 6, "The WebSphere MQ .NET classes and interfaces," on page 27.

Connection differences

The way you program for WebSphere MQ .NET has some dependencies on the connection modes you want to use.

Client connections

When WebSphere MQ classes for .NET is used as a client, it is similar to the WebSphere® MQ C client, but has the following differences:

- It supports only TCP/IP.
- It does not support connection tables.
- Information that would be stored in a channel definition and in environment variables is stored in a class called Environment. Alternatively, this information can be passed as parameters when the connection is made.

For general information on WebSphere MQ clients, see the *WebSphere MQ Clients* book.

Defining which connection to use

The connection is determined by the setting of variables in the MQEnvironment class.

Either set the value of MQEnvironment.Hostname and MQEnvironment.Channel as follows:

- For client connections, set to the host name of the WebSphere MQ server and server connection channel to which you want to connect
- For bindings mode, set both variables to null

or set the environment variable NMQ_MQ_LIB to explicitly choose connection type as shown in the following table

NMQ_MQ_LIB value	Connection type
mqm.dll	Server bindings connection
mqic32.dll	Client bindings connection
mqic32xa.dll	XA Client bindings connection

Example code fragments

This section includes two example code fragments; Figure 1 and Figure 2 on page 18. Each one uses a particular connection and includes notes to describe the changes needed to use alternative connections.

Example code (client connection)

The following code fragment demonstrates an application that uses a client connection to:

1. Connect to a queue manager
2. Put a message onto SYSTEM.DEFAULT.LOCAL.QUEUE
3. Get the message back

```
// ===========================================================================
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 2003
// ===========================================================================

using System;
using IBM.WMQ;

class MQSample
{

  private String hostname = "your_hostname"; //define the name of your
                        //host to connect to

  private String channel = "server_channel"; //define name of channel
                        //for client to use
                        //Note:assumes WebSphere MQ Server
                        //is listening on the default
                        //TCP/IP port of 1414

  private String qManager = "your_Q_manager";   //define name of queue
                        //manager object to
                        //connect to.

  //When the class is called,this initialization is done first.

  public void init()
  {
    //Set up WebSphere MQ environment
    MQEnvironment.Hostname = hostname;           //Could have put the
                                                 //hostname and channel
    MQEnvironment.Channel = channel;             //string directly here!
     }//end of init
```

Figure 1. WebSphere MQ classes for .NET example code (client connection) (Part 1 of 2)

```
public void start()
  {
    try
    {
   //Create a connection to the queue manager
   MQQueueManager qMgr =new MQQueueManager(qManager);

   //Set up the options on the queue we wish to open...
   int openOptions =MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT ;

   //Now specify the queue that we wish to open,and the open options...
   MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
      openOptions);

   //Define a simple WebSphere MQ message,and write some text in UTF format..
   MQMessage hello_world =new MQMessage();
   hello_world.WriteUTF("Hello World!");

   //specify the message options...
   MQPutMessageOptions pmo =new MQPutMessageOptions();//accept the defaults,
                              //same as MQPMO_DEFAULT constant

   //put the message on the queue
   system_default_local_queue.Put(hello_world,pmo);

   //get the message back again...
   //First define a WebSphere MQ message buffer to receive the message into..
   MQMessage retrievedMessage =new MQMessage();
   retrievedMessage.MessageId =hello_world.MessageId;

   //Set the get message options..
   MQGetMessageOptions gmo =new MQGetMessageOptions();//accept the defaults
                              //same as MQGMO_DEFAULT
   //get the message off the queue..
   system_default_local_queue.Get(retrievedMessage,gmo);

   //And prove we have the message by displaying the UTF message text
   String msgText =retrievedMessage.ReadUTF();
   Console.WriteLine("The message is:"+msgText);

   //Close the queue
   system_default_local_queue.Close();

   //Disconnect from the queue manager
   qMgr.Disconnect();
  }
  //If an error has occurred in the above,try to identify what went wrong.
  //Was it a WebSphere MQ error?

  catch (MQException ex)
  {
   Console.WriteLine("A WebSphere MQ error occurred :Completion code "+
   ex.CompletionCode +
   "Reason code "+ex.ReasonCode);
  }
  catch (System.Exception ex)
  {
   Console.WriteLine("A System error occurred:"+ex);
  }
  }//end of start
}//end of sample
```

Figure 1. WebSphere MQ classes for .NET example code (client connection) (Part 2 of 2)

Example code (server bindings connection)

The following code fragment demonstrates an application that uses server bindings mode to:

1. Connect to a queue manager
2. Put a message onto SYSTEM.DEFAULT.LOCAL.QUEUE
3. Get the message back again

```
// ======================================================================
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 2003
// ======================================================================

using System;
using IBM.WMQ;

public class MQSample1
{
  private String qManager ="your_Q_manager"; //define name of queue
                                             //manager to connect to.
  private MQQueueManager qMgr;      //define a queue manager object

static void Main(string[] args)
  {
   new MQSample1();
  }
```

Figure 2. WebSphere MQ classes for .NET example code (server bindings connection) (Part 1 of 2)

```
  public MQSample1()
  {
    try
    {
      //Create a connection to the queue manager
   qMgr =new MQQueueManager(qManager);

   //Set up the options on the queue we wish to open...
   //Note.All WebSphere MQ Options are prefixed with MQC
   int openOptions =MQC.MQOO_INPUT_AS_Q_DEF |MQC.MQOO_OUTPUT ;

   //Now specify the queue that we wish to open,
   //and the open options...
   MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
      openOptions);

   //Define a simple WebSphere MQ message,and write some text in UTF format..
   MQMessage hello_world =new MQMessage();
   hello_world.WriteUTF("Hello World!");

   //specify the message options...
   MQPutMessageOptions pmo =new MQPutMessageOptions();//accept the defaults,
                              //same as MQPMO_DEFAULT
   //put the message on the queue
   system_default_local_queue.Put(hello_world,pmo);
   //get the message back again...
   //First define a WebSphere MQ message buffer to receive the message into..
   MQMessage retrievedMessage =new MQMessage();
   retrievedMessage.MessageId =hello_world.MessageId;
   //Set the get message options...
   MQGetMessageOptions gmo =new MQGetMessageOptions();//accept the defaults
                              //same as MQGMO_DEFAULT

   //get the message off the queue...
   system_default_local_queue.Get(retrievedMessage,gmo);
   //And prove we have the message by displaying the UTF message text
   String msgText =retrievedMessage.ReadUTF();
   Console.WriteLine("The message is:"+msgText);
   //Close the queue...
   system_default_local_queue.Close();
   //Disconnect from the queue manager
   qMgr.Disconnect();
   }
   //If an error has occurred in the above,try to identify what went wrong

   //Was it a WebSphere MQ error?
   catch (MQException ex)
   {
   Console.WriteLine("A WebSphere MQ error occurred :Completion code "+
   ex.CompletionCode +"Reason code "+ex.ReasonCode);
   }
   //Was it a System error?
   catch (System.Exception ex)
   {
     Console.WriteLine("A System error occurred:"+ex);
   }
  }
}//end of sample
```

Figure 2. WebSphere MQ classes for .NET example code (server bindings connection) (Part 2 of 2)

Operations on queue managers

This section describes how to connect to, and disconnect from, a queue manager using WebSphere MQ classes for .NET.

Setting up the WebSphere MQ environment

Note: This step is not necessary when using WebSphere MQ classes for .NET in server bindings mode. In that case, go directly to "Connecting to a queue manager." Before you use the client connection to connect to a queue manager, you must set up the MQEnvironment.

The C based WebSphere MQ clients rely on environment variables to control the behavior of the MQCONN call. The .NET programming interface allows you to use the NMQ_MQ_LIB environment variable but also includes a class MQEnvironment. This class allows you to specify the following details that are to be used during the connection attempt:

- Channel name
- Host name
- Port number

To specify the channel name and host name, use the following code:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

By default, the clients attempt to connect to a WebSphere MQ listener at port 1414. To specify a different port, use the code:

```
MQEnvironment.Port = nnnn;
```

Connecting to a queue manager

You are now ready to connect to a queue manager by creating a new instance of the MQQueueManager class:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

To disconnect from a queue manager, call the Disconnect() method on the queue manager:

```
queueManager.Disconnect();
```

If you call the Disconnect method, all open queues and processes that you have accessed through that queue manager are closed. However, it is good programming practice to close these resources explicitly when you finish using them. To do this, use the Close() method.

The Commit() and Backout() methods on a queue manager replace the MQCMIT and MQBACK calls that are used with the procedural interface.

Accessing queues and processes

To access queues, use the MQQueueManager class. The MQOD (object descriptor structure) is collapsed into the parameters of these methods. For example, to open a queue on a queue manager called queueManager, use the following code:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

The *options* parameter is the same as the Options parameter in the MQOPEN call.

The AccessQueue method returns a new object of class MQQueue.

When you have finished using the queue, use the Close() method to close it, as in the following example:

```
queue.Close();
```

With WebSphere MQ .NET, you can also create a queue by using the MQQueue constructor. The parameters are exactly the same as for the accessQueue method, with the addition of a queue manager parameter. For example:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

Constructing a queue object in this way enables you to write your own subclasses of MQQueue.

Handling messages

Put messages onto queues using the Put() method of the MQQueue class. You get messages from queues using the Get() method of the MQQueue class. Unlike the procedural interface, where MQPUT and MQGET put and get arrays of bytes, the WebSphere MQ classes for .NET put and get instances of the MQMessage class. The MQMessage class encapsulates the data buffer that contains the actual message data, together with all the MQMD (message descriptor) parameters that describe that message.

To build a new message, create a new instance of the MQMessage class and use the WriteXXX methods to put data into the message buffer.

When the new message instance is created, all the MQMD parameters are automatically set to their default values, as defined in the *WebSphere MQ Application Programming Reference*. The Put() method of MQQueue also takes an instance of the MQPutMessageOptions class as a parameter. This class represents the MQPMO structure. The following example creates a message and puts it onto a queue:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message!
queue.Put(myMessage,pmo);
```

The Get() method of MQQueue returns a new instance of MQMessage, which represents the message just taken from the queue. It also takes an instance of the MQGetMessageOptions class as a parameter. This class represents the MQGMO structure.

You do not need to specify a maximum message size, because the Get() method automatically adjusts the size of its internal buffer to fit the incoming message. Use the ReadXXX methods of the MQMessage class to access the data in the returned message.

The following example shows how to get a message from a queue:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

You can alter the number format that the read and write methods use by setting the *encoding* member variable.

You can alter the character set to use for reading and writing strings by setting the *characterSet* member variable.

See “MQMessage” on page 35 for more details.

Note: The WriteUTF() method of MQMessage automatically encodes the length of the string as well as the Unicode bytes it contains. When your message will be read by another .NET program (using ReadUTF()), this is the simplest way to send string information.

Handling errors

Methods in the .NET interface do not return a completion code and reason code. Instead, they throw an exception whenever the completion code and reason code resulting from a WebSphere MQ call are not both zero. This simplifies the program logic so that you do not have to check the return codes after each call to WebSphere MQ. You can decide at which points in your program you want to deal with the possibility of failure. At these points, you can surround your code with `try` and `catch` blocks, as in the following example:

```
try {
    myQueue.Put(messageA,PutMessageOptionsA);
    myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    Console.WriteLine("An error occurred during the put operation:" +
                      "CC = " + ex.CompletionCode +
                      "RC = " + ex.ReasonCode);
    Console.WriteLine("Cause exception:" + ex );
}
```

The WebSphere MQ call reason codes reported back in .NET exceptions are documented in a chapter called “Return Codes” in the *WebSphere MQ Application Programming Reference*.

Getting and setting attribute values

The classes MQManagedObject, MQQueue, and MQQueueManager contain properties that allow you to get and set their attribute values. Note that for MQQueue, the methods work only if you specify the appropriate inquire and set flags when you open the queue.

For less common attributes, the MQQueueManager and MQQueue classes all inherit from a class called MQManagedObject. This class defines the Inquire() and Set() interfaces.

When you create a new queue manager object by using the *new* operator, it is automatically opened for inquire. When you use the AccessQueue() method to access a queue object, that object is *not* automatically opened for either inquire or set operations. This is because adding these options automatically can cause problems with some types of remote queues. To use the Inquire and Set methods and to set properties on a queue, you must specify the appropriate inquire and set flags in the openOptions parameter of the AccessQueue() method.

The inquire and set methods take three parameters:

- selectors array
- intAttrs array
- charAttrs array

You do not need the SelectorCount, IntAttrCount, and CharAttrLength parameters that are found in MQINQ, because the length of an array is always known. The following example shows how to make an inquiry on a queue:

```
//inquire on a queue
const int MQIA_DEF_PRIORITY =6;
const int MQCA_Q_DESC =2013;
const int MQ_Q_DESC_LENGTH =64;
int [ ] selectors =new int [2 ] ;
int [ ] intAttrs =new int [1 ] ;
byte [ ] charAttrs =new byte [MQ_Q_DESC_LENGTH ];
selectors [0 ] =MQIA_DEF_PRIORITY;
selectors [1 ] =MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

Multithreaded programs

Multithreaded programs are hard to avoid. Consider a simple program that connects to a queue manager and opens a queue at startup. The program displays a single button on the screen. When a user presses that button, the program fetches a message from the queue.

The .NET runtime environment is inherently multithreaded. Therefore, your application initialization occurs in one thread, and the code that executes in response to the button press executes in a separate thread (the user interface thread).

The implementation of WebSphere MQ .NET ensures that, for a given connection (MQQueueManager object instance), all access to the target WebSphere MQ queue manager is synchronized. The default behaviour is that a thread that wants to issue a call to a queue manager is blocked until all other calls in progress for that connection are complete. If you require simultaneous access to the same queue manager from multiple threads within your program, create a new MQQueueManager object for each thread that requires concurrent access. (This is equivalent to issuing a separate MQCONN call for each thread.)

If the default connection options are overridden by MQC.MQCNO_HANDLE_SHARE_NONE or MQC.MQCNO_SHARE_NO_BLOCK then the queue manager is no longer synchronized.

Secure Sockets Layer (SSL) support

WebSphere MQ classes for .NET client applications support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

Enabling SSL

SSL is supported only for client connections. To enable SSL, you must specify the CipherSpec to use when communicating with the queue manager, and this must match the CipherSpec set on the target channel.

To enable SSL, specify the CipherSpec using the SSLCipherSpec static member variable of MQEnvironment. The following example attaches to a SVRCONN channel named SECURE.SVRCONN.CHANNEL, which has been set up to require SSL with a CipherSpec of NULL_MD5:

```
MQEnvironment.Hostname       = "your_hostname";
MQEnvironment.Channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec = "NULL_MD5";
MQEnvironment.SSLKeyRepository = @"C:\mqm\key"
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

See Appendix A, “SSL CipherSpecs supported by WebSphere MQ,” on page 75 for a list of CipherSpecs.

The SSLCipherSpec property can also be set using the MQC.SSL_CIPHER_SPEC_PROPERTY in the hash table of connection properties.

To successfully connect using SSL, the client key store must be set up with Certificate Authority root certificates chain from which the certificate presented by the queue manager can be authenticated. Similarly, if SSLClientAuth on the SVRCONN channel has been set to MQSSL_CLIENT_AUTH_REQUIRED, the client key store must contain an identifying personal certificate that is trusted by the queue manager.

Using the distinguished name of the queue manager

The queue manager identifies itself using an SSL certificate, which contains a *Distinguished Name* (DN). A WebSphere MQ .NET client application can use this DN to ensure that it is communicating with the correct queue manager. A DN pattern is specified using the sslPeerName variable of MQEnvironment. For example, setting:

```
  MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

allows the connection to succeed only if the queue manager presents a certificate with a Common Name beginning QMGR., and at least two Organizational Unit names, the first of which must be IBM and the second WEBSPHERE.

The SSLPeerName property can also be set using the MQC.SSL_PEER_NAME_PROPERTY in the hash table of connection properties. For more information about distinguished names and rules for setting peer names, refer to *WebSphere MQ Security*.

If SSLPeerName is set, connections succeed only if it is set to a valid pattern and the queue manager presents a matching certificate.

Error handling when using SSL

The following reason codes can be issued by WebSphere MQ classes for .NET when connecting to a queue manager using SSL:

MQRC_SSL_NOT_ALLOWED
: The SSLCipherSpec property was set, but bindings connect was used. Only client connect supports SSL.

MQRC_SSL_PEER_NAME_MISMATCH
: The DN pattern specified in the SSLPeerName property did not match the DN presented by the queue manager.

MQRC_SSL_PEER_NAME_ERROR
: The DN pattern specified in the SSLPeerName property was not valid.

Compiling and testing WebSphere MQ .NET programs

Before compiling WebSphere MQ .NET programs, you must ensure that your WebSphere MQ classes for .NET installation directory is in your CLASSPATH environment variable.

To build a C# application using WebSphere MQ classes for .NET, use the following command

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib:mqmtop\bin /out:MyProg.exe MyProg.cs
```

To build a Visual Basic application using WebSphere MQ classes for .NET, use the following command

```
vbc /r:System.dll /r:mqmtop\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

To build a Managed C++ application using WebSphere MQ classes for .NET, use the following command.

```
cl /clr mqmtop\bin Myprog.cpp
```

Before running WebSphere MQ .NET programs, you must ensure the CLR can locate the .NET Assembly. This can be done by either:

- registering amgmdnet.dll in the ’Global Assembly Cache’.

  The command to register the assembly is:

  ```
  gacutil -i mqmtop\bin\amqmdnet.dll.
  ```

or

- creating a ’configuration file’ for the WebSphere MQ .NET program.

  The configuration file should tell the CLR how to locate the amqmdnet.dll assembly.

Tracing WebSphere MQ .NET programs

WebSphere MQ .NET uses the standard WebSphere MQ trace facility, which you can use to produce diagnostic messages if you suspect that there might be a problem with the code. (You normally need to use this facility only at the request of IBM service.)

Refer to WebSphere MQ Script (MQSC) Command Reference for information on trace commands.

Chapter 6. The WebSphere MQ .NET classes and interfaces

This topic describes all the WebSphere MQ .NET classes and interfaces. It includes details of the variables, constructors, and methods in each class and interface.

The following classes are described:
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQPutMessageOptions
- MQQueue
- MQQueueManager

The following structure is described:
- MQC

MQEnvironment

```
System.Object
  │
  └─ IBM.WMQ.MQEnvironment
```

public class **IBM.WMQ.MQEnvironment**
extends **System.Object**

The MQEnvironment class is used to control how the MQQueueManager constructor is called.

Constructors

MQEnvironment
```
public MQEnvironment()
```

Properties

Note: Variables marked with * do not apply when connecting directly to WebSphere MQ in server bindings mode.

Channel*
```
public static String Channel {get; set;}
```

The name of the channel to connect to on the target queue manager. You *must* set this property, before constructing an MQQueueManager instance for use in client mode.

Hostname*
```
public static String Hostname {get; set;}
```

The TCP/IP hostname of the machine on which the WebSphere MQ server resides. If the hostname is not set, and no overriding properties are set, server bindings mode is used to connect to the local queue manager.

Port* `public static int Port {get; set;}`

The port to connect to. This is the port on which the WebSphere MQ server is listening for incoming connection requests. The default value is 1414.

SSLCipherSpec*
`public static String SSLCipherSpec {get; set;}`

If set, SSL is enabled for the connection. Set the SSLCipherSpec to the value of the CipherSpec set on the SVRCONN channel. If set to null (default), no SSL encryption is performed.

SSLKeyRepository
`public static String SSLCipherSpec {get; set;}`

This property is set to the fully-qualified filename of the key repository.

The .sto extension is a mandatory part of the filename, but is not included as part of the value of the parameter. If set to null (default), the certificate MQSSLKEYR environment variable will be used to locate the key repository. This variable is ignored if sslCipherSpec is null.

Note:

The .sto extension is a mandatory part of the filename, but is not included as part of the value of the parameter. The directory you specify must exist.WebSphere MQ creates the file the first time it accesses the new key repository, unless the file already exists.

SSLPeerName*
`public static String sslPeerName {get; set;}`

A distinguished name pattern. If sslCipherSpec is set, this variable can be used to ensure the correct queue manager is used. For a description of the format for this value, see “Using the distinguished name of the queue manager” on page 24. If set to null (default), no checking of the queue manager’s DN is performed. This variable is ignored if sslCipherSpec is null.

MQException

```
System.Object
  |
  └─ System.Exception
      |
      └─ System.ApplicationException
          |
          └─ IBM.WMQ.WMQException
```

public class **IBM.WMQ.MQException**
extends **System.ApplicationException**

An MQException is thrown whenever a WebSphere MQ error occurs.

Constructors

MQException

```
public MQException(int completionCode,
                   int reasonCode)
```

Construct a new MQException object.

Parameters

completionCode
The WebSphere MQ completion code.

reasonCode
The WebSphere MQ reason code.

Properties

CompletionCode

```
public int CompletionCode {get; set;}
```

WebSphere MQ completion code giving rise to the error. The possible values are:
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

ReasonCode

```
public int ReasonCode {get; set;}
```

WebSphere MQ reason code describing the error. For a full explanation of the reason codes, refer to the *WebSphere MQ Application Programming Reference*.

MQGetMessageOptions

```
System.Object
│
└─ IBM.WMQ.MQBase
   │
   └─ IBM.WMQ.MQBaseObject
      │
      └─ IBM.WMQ.MQGetMessageOptions
```

public class **IBM.WMQ.MQGetMessageOptions**
extends **IBM.WMQ.MQBaseObject**

This class contains options that control the behavior of MQQueue.Get().

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with an asterisk (*).

Constructors

MQGetMessageOptions

```
public MQGetMessageOptions()
```

Construct a new MQGetMessageOptions object with options set to MQC.MQGMO_NO_WAIT, a wait interval of zero, and a blank resolved queue name.

Properties

GroupStatus*

```
public int GroupStatus {get;}
```

This is an output field that indicates whether the retrieved message is in a group, and if it is, whether it is the last in the group. Possible values are:

MQC.MQGS_LAST_MSG_IN_GROUP
Message is the last in the group. This is also the value returned if the group consists of only one message.

MQC.MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQC.MQGS_NOT_IN_GROUP
Message is not in a group.

MatchOptions*

```
public int MatchOptions {get; set;}
```

Selection criteria that determine which message is retrieved. The following match options can be set:

MQC.MQMO_MATCH_CORREL_ID
Correlation id to be matched.

MQC.MQMO_MATCH_GROUP_ID
Group id to be matched.

MQC.MQMO_MATCH_MSG_ID
Message id to be matched.

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
Match message sequence number.

MQC.MQMO_NONE
No matching required.

Options

```
public int Options {get; set;}
```

Options that control the action of MQQueue.get. Any or none of the following values can be specified. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
Allow truncation of message data.

MQC.MQGMO_BROWSE_FIRST
Browse from start of queue.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Browse message under browse cursor.

MQC.MQGMO_BROWSE_NEXT
Browse from the current position in the queue.

MQC.MQGMO_CONVERT
Request the application data to be converted, to conform to the characterSet and encoding attributes of the MQMessage, before the data is copied into the message buffer. Because data conversion is also applied as the data is retrieved from the message buffer, applications do not usually set this option.

Using this option can cause problems when converting from single byte character sets to double byte character sets. Instead, do the conversion using the readString, readLine, and writeString methods after the message has been delivered.

MQC.MQGMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQGMO_LOCK*
Lock the message that is browsed.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Allow a unit of work to be backed out without reinstating the message on the queue.

MQC.MQGMO_MSG_UNDER_CURSOR
Get message under browse cursor.

MQC.MQGMO_NONE
No other options have been specified; all options assume their default values.

MQC.MQGMO_NO_SYNCPOINT
Get message without syncpoint control.

MQC.MQGMO_NO_WAIT
Return immediately if there is no suitable message.

MQC.MQGMO_SYNCPOINT
Get the message under syncpoint control; the message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Get message with syncpoint control if message is persistent.

MQC.MQGMO_UNLOCK*
Unlock a previously locked message.

MQC.MQGMO_WAIT
Wait for a message to arrive.

Segmenting and grouping WebSphere MQ messages can be sent or received as a single entity, can be split into several segments for sending and receiving, and can also be linked to other messages in a group.

Each piece of data that is sent is known as a *physical* message, which can be a complete *logical* message, or a segment of a longer logical message.

Each physical message usually has a different `MsgId`. All the segments of a single logical message have the same `groupId` value and `MsgSeqNumber` value, but the `Offset` value is different for each segment. The `Offset` field gives the offset of the data in the physical message from the start of the logical message. The segments usually have different `MsgId` values, because they are individual physical messages.

Logical messages that form part of a group have the same `groupId` value, but each message in the group has a different `MsgSeqNumber` value. Messages in a group can also be segmented.

The following options can be used for dealing with segmented or grouped messages:

MQC.MQGMO_ALL_MSGS_AVAILABLE*
: Retrieve messages from a group only when all the messages in the group are available.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
: Retrieve the segments of a logical message only when all the segments in the group are available.

MQC.MQGMO_COMPLETE_MSG*
: Retrieve only complete logical messages.

MQC.MQGMO_LOGICAL_ORDER*
: Return messages in groups, and segments of logical messages, in logical order.

ResolvedQueueName

```
public String ResolvedQueueName {get;}
```

This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved. This is different from the name used to open the queue if an alias queue or model queue was opened.

Segmentation*

```
public char Segmentation {get;}
```

This is an output field that indicates whether or not segmentation is allowed for the retrieved message. Possible values are:

MQC.MQSEG_INHIBITED
: Segmentation not allowed.

MQC.MQSEG_ALLOWED
: Segmentation allowed.

SegmentStatus*

```
public byte SegmentStatus {get;}
```

This is an output field that indicates whether the retrieved message is a segment of a logical message. If the message is a segment, the flag indicates whether or not it is the last segment. Possible values are:

MQC.MQSS_LAST_SEGMENT
: Message is the last segment of the logical message. This is also the value returned if the logical message consists of only one segment.

MQC.MQSS_NOT_A_SEGMENT
: Message is not a segment.

MQC.MQSS_SEGMENT
: Message is a segment, but is not the last segment of the logical message.

WaitInterval

```
public int WaitInterval {get; set;}
```

The maximum time (in milliseconds) that an MQQueue.get call waits for a suitable message to arrive (used in conjunction with MQC.MQGMO_WAIT). A value of MQC.MQWI_UNLIMITED indicates that an unlimited wait is required.

MQManagedObject

```
System.Object
  │
  └─ IBM.WMQ.MQBase
     │
     └─ IBM.WMQ.MQBaseObject
        │
        └─ IBM.WMQ.MQManagedObject
```

public class **IBM.WMQ.MQManagedObject**
extends **IBM.WMQ.MQBaseObject**

MQManagedObject is a superclass for MQQueueManager and MQQueue. It provides the ability to inquire and set attributes of these resources.

Constructors

MQManagedObject

```
protected MQManagedObject()
```

Constructor method.

Methods

Close

```
public virtual void Close()
```

Throws MQException.

Closes the object. No further operations against this resource are permitted after this method has been called. To change the behavior of the Close method, set the closeOptions attribute.

Throws MQException if the WebSphere MQ call fails.

Inquire

```
public void Inquire(int[] selectors,
                    int[] intAttrs,
                    byte[] charAttrs)
```

Throws MQException.

Returns an array of integers and a set of character strings containing the attributes of an object (queue, process, or queue manager).

The attributes to be queried are specified in the selectors array. Refer to the *WebSphere MQ Application Programming Reference* for details of the permissible selectors and their corresponding integer values.

Many of the more common attributes can be queried using the GetXXX() methods defined in MQManagedObject, MQQueue and MQQueueManager.

Parameters

selectors
: Integer array identifying the attributes with values to be inquired on.

intAttrs
: The array in which the integer attribute values are returned. Integer attribute values are returned in the same order as the integer attribute selectors in the selectors array.

charAttrs
: The buffer in which the character attributes are returned, concatenated. Character attributes are returned in the same order as the character attribute selectors in the selectors array. The length of each attribute string is fixed for each attribute.

Throws MQException if the inquire fails.

Set

```
public void Set(int[] selectors,
                int[] intAttrs,
                byte[] charAttrs)
```

Throws MQException.

Sets the attributes defined in the selector's vector.

The attributes to be set are specified in the selectors array. Refer to the *WebSphere MQ Application Programming Reference* for details of the permissible selectors and their corresponding integer values.

Parameters

selectors
: Integer array identifying the attributes with values to be set.

intAttrs
: The array of integer attribute values to be set. These values must be in the same order as the integer attribute selectors in the selectors array.

charAttrs
: The buffer in which the character attributes to be set are concatenated. These values must be in the same order as the character attribute selectors in the selectors array. The length of each character attribute is fixed.

Throws MQException if the set fails.

Properties

AlternateUserId

`public String AlternateUserId {get; set;}`

The alternate user ID (if any) specified when this resource was opened. Setting this attribute has no effect.

CloseOptions

`public int CloseOptions {get; set;}`

Set this attribute to control the way the resource is closed. The default value is MQC.MQCO_NONE, and this is the only permissible value for all resources other than permanent dynamic queues, and temporary dynamic queues that are being accessed by the objects that created them. For these queues, the following additional values are permissible:

MQC.MQCO_DELETE
Delete the queue if there are no messages.

MQC.MQCO_DELETE_PURGE
Delete the queue, purging any messages on it.

ConnectionReference
```
public  MQQueueManager ConnectionReference {get;}
```

The queue manager to which this resource belongs.

IsOpen
```
public boolean IsOpen {get;}
```

Indicates whether this resource is currently open.

Name
```
public String Name {get;}
```

The name of this resource (either the name supplied on the access method, or the name allocated by the queue manager for a dynamic queue).

OpenOptions
```
public int OpenOptions {get; set;}
```

The options specified when this resource was opened. Setting this attribute has no effect.

MQMessage

```
System.Object
 │
 └─ IBM.WMQ.MQBase
    │
    └─ IBM.WMQ.MQBaseObject
       │
       └─ IBM.WMQ.MQMessage
```

public class **IBM.WMQ.MQMessage**
extends **IBM.WMQ.MQBaseObject**
implements **DataInput**, **DataOutput**

MQMessage represents both the message descriptor and the data for a WebSphere MQ message. There is group of readXXX methods for reading data from a message, and a group of writeXXX methods for writing data into a message. The format of numbers and strings used by these read and write methods can be controlled by the Encoding and CharacterSet properties. The remaining properties contain control information that accompanies the application message data when a message travels between sending and receiving applications. The application can set values into the property before putting a message to a queue and can read values after retrieving a message from a queue.

Constructors

MQMessage
```
public MQMessage()
```

Creates a new message with default message descriptor information and an empty message buffer.

Methods

ClearMessage

```
public void ClearMessage()
```

Throws IOException.

Discards any data in the message buffer and sets the data offset back to zero.

ReadBoolean

```
public bool ReadBoolean()
```

Throws IOException.

Reads a (signed) byte from the current position in the message buffer.

ReadByte

```
public byte ReadByte()
```

Throws IOException.

Reads a byte from the current position in the message buffer.

ReadBytes

```
public byte[] ReadBytes(int count)
```

Throws IOException.

Reads byte[’count’] (’count’ bytes) from the buffer starting at the data pointer. After the data has been read the data pointer is incremented by ’count’.

ReadChar

```
public char ReadChar()
```

Throws IOException, EndOfStreamException.

Reads a Unicode character from the current position in the message buffer.

ReadDecimal2

```
public short ReadDecimal2()
```

Throws IOException, EndOfStreamException.

Reads a 2-byte packed decimal number (-999 to 999). The behavior of this method is controlled by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

ReadDecimal4

```
public int readDecimal4()
```

Throws IOException, EndOfStreamException.

Reads a 4-byte packed decimal number (-9999999 to 9999999). The behavior of this method is controlled by the value of the encoding member variable.

A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

ReadDecimal8

```
public long ReadDecimal8()
```

Throws IOException, EndOfStreamException.

Reads an 8-byte packed decimal number (-999999999999999 to 999999999999999). The behavior of this method is controlled by the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

ReadDouble

```
public double ReadDouble()
```

Throws IOException, EndOfStreamException.

Reads a double from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard doubles in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 reads a System/390® format floating point number.

ReadFloat

```
public float ReadFloat()
```

Throws IOException, EndOfStreamException.

Reads a float from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 reads a System/390 format floating point number.

ReadFully

```
public void ReadFully(ref byte[] b)
```

Throws Exception, EndOfStreamException.

Fills the byte array `b` with data from the message buffer.

ReadFully

```
public void ReadFully(ref sbyte[] b)
```

Throws Exception, EndOfStreamException.

Fills the sbyte array b with data from the message buffer.

ReadFully

```
public void ReadFully(ref byte[] b,
                      int off,
                      int len)
```

Throws IOException, EndOfStreamException.

Fills *len* elements of the byte array b with data from the message buffer, starting at offset *off*.

ReadFully

```
public void ReadFully(ref sbyte[] b,
                      int off,
                      int len)
```

Throws IOException, EndOfStreamException.

Fills *len* elements of the sbyte array b with data from the message buffer, starting at offset *off*.

ReadInt

```
public int ReadInt()
```

Throws IOException, EndOfStreamException.

Reads an integer from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian integer; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian integer.

ReadInt2

```
public short ReadInt2()
```

Throws IOException, EndOfStreamException.

Synonym for ReadShort(), provided for cross-language WebSphere MQ API compatibility.

ReadInt4

```
public int ReadInt4()
```

Throws IOException, EndOfStreamException.

Synonym for ReadInt(), provided for cross-language WebSphere MQ API compatibility.

ReadInt8

```
public long ReadInt8()
```

Throws IOException, EndOfStreamException.

Synonym for ReadLong(), provided for cross-language WebSphere MQ API compatibility.

ReadLine

```
public String ReadLine()
```

Throws IOException.

Converts from the codeset identified in the characterSet member variable to Unicode, and then reads in a line that has been terminated by \n, \r, \r\n, or EOF.

ReadLong

```
public long ReadLong()
```

Throws IOException, EndOfStreamException.

Reads a long from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian long; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian long.

ReadObject

```
public Object ReadObject()
```

Throws SerialisationException, IOException.

Reads an object from the message buffer. The class of the object, the signature of the class, and the value of the non-transient and non-static fields of the class are all read.

ReadShort

```
public short ReadShort()
```

Throws IOException, EndOfStreamException.

Reads a short from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian short; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian short.

ReadString

```
public String ReadString(int length)
```

Throws IOException, EndOfStreamException.

Reads a string in the codeset identified by the characterSet member variable, and convert it into Unicode.

Parameters:

length	The number of characters to read (which may differ from the number of bytes according to the codeset, because some codesets use more than one byte per character).

ReadUInt2

```
public ushort ReadUInt2()
```

Throws IOException, EndOfStreamException.

Synonym for ReadUnsignedShort(), provided for cross-language WebSphere MQ API compatibility.

ReadUnsignedByte

```
public byte ReadUnsignedByte()
```

Throws IOException, EndOfStreamException.

Reads an unsigned byte from the current position in the message buffer.

ReadUnsignedShort

```
public ushort ReadUnsignedShort()
```

Throws IOException, EndOfStreamException.

Reads an unsigned short from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian unsigned short; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian unsigned short.

ReadUTF

```
public String ReadUTF()
```

Throws IOException.

Reads a UTF string, prefixed by a 2-byte length field, from the current position in the message buffer.

ResizeBuffer

```
public void ResizeBuffer(int size)
```

Throws IOException.

A hint to the MQMessage object about the size of buffer that might be required for subsequent get operations. If the message currently contains message data, and the new size is less than the current size, the message data is truncated.

Seek

```
public void Seek(int pos)
```

Throws IOException, ArgumentOutOfRangeException ArgumentException.

Moves the cursor to the absolute position in the message buffer given by *pos*. Subsequent reads and writes act at this position in the buffer.

SkipBytes

```
public int SkipBytes(int n)
```

Throws IOException, EndOfStreamException.

Moves forward `n` bytes in the message buffer.

This method blocks until one of the following occurs:

- All the bytes are skipped

- The end of message buffer is detected
- An exception is thrown

Returns the number of bytes skipped, which is always n.

Write

```
public void Write(int b)
```

Throws IOException.

Writes a byte into the message buffer at the current position.

Write

```
public void Write(byte[] b)
```

Throws IOException.

Writes an array of bytes into the message buffer at the current position.

Write

```
public void Write(sbyte[] b)
```

Throws IOException.

Writes an array of sbytes into the message buffer at the current position.

Write

```
public void Write(byte[] b,
                  int off,
                  int len)
```

Throws IOException.

Writes a series of bytes into the message buffer at the current position. *len* bytes are written, taken from offset *off* in the array b.

Write

```
public void Write(sbyte b[],
                  int off,
                  int len)
```

Throws IOException.

Writes a series of sbytes into the message buffer at the current position. *len* sbytes are written, taken from offset *off* in the array b.

WriteBoolean

```
public void WriteBoolean(boolean v)
```

Throws IOException.

Writes a boolean into the message buffer at the current position.

WriteByte

```
public void WriteByte(int v)
```

Throws IOException.

Writes a byte into the message buffer at the current position.

WriteByte

```
public void WriteByte(byte value)
```

Throws IOException.

Writes a byte into the message buffer at the current position.

WriteByte

```
public void WriteByte(sbyte value)
```

Throws IOException.

Writes an sbyte into the message buffer at the current position.

WriteBytes

```
public void WriteBytes(String s)
```

Throws IOException.

Writes the string to the message buffer as a sequence of bytes. Each character in the string is written in sequence by discarding its high eight bits.

WriteChar

```
public void WriteChar(int v)
```

Throws IOException.

Writes a Unicode character into the message buffer at the current position.

WriteChars

```
public void WriteChars(String s)
```

Throws IOException.

Writes a string as a sequence of Unicode characters into the message buffer at the current position.

WriteDecimal2

```
public void WriteDecimal2(short v)
```

Throws IOException, MQException.

Writes a 2-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters

v	can be in the range -999 to 999.

WriteDecimal4

```
public void WriteDecimal4(int v)
```

Throws IOException, MQException.

Writes a 4-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters

v can be in the range -9999999 to 9999999.

WriteDecimal8

```
public void WriteDecimal8(long v)
```

Throws IOException, MQException.

Writes an 8-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters:

v can be in the range -999999999999999 to 999999999999999.

WriteDouble

```
public void WriteDouble(double v)
```

Throws IOException, MQException.

Writes a double into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a System/390 format floating point number. Note that the range of IEEE doubles is greater than the range of S/390® double precision floating point numbers, so very large numbers cannot be converted.

WriteFloat

```
public void WriteFloat(float v)
```

Throws IOException, MQException.

Writes a float into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a System/390 format floating point number.

WriteInt

```
public void WriteInt(int v)
```

Throws IOException.

Writes an integer into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian integer; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian integer.

WriteInt2

```
public void WriteInt2(int v)
```

Throws IOException.

Synonym for WriteShort(), provided for cross-language WebSphere MQ API compatibility.

WriteInt4

```
public void WriteInt4(int v)
```

Throws IOException.

Synonym for WriteInt(), provided for cross-language WebSphere MQ API compatibility.

WriteInt8

```
public void WriteInt8(long v)
```

Throws IOException.

Synonym for WriteLong(), provided for cross-language WebSphere MQ API compatibility.

WriteLong

```
public void WriteLong(long v)
```

Throws IOException.

Writes a long into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian long; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian long.

WriteObject

```
public void WriteObject(Object obj)
```

Throws IOException.

Writes the specified object to the message buffer. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all its supertypes are all written.

WriteShort

```
public void WriteShort(int v)
```

Throws IOException.

Writes a short into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian short; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian short.

WriteString

```
public void WriteString(String str)
```

Throws IOException.

Writes a string into the message buffer at the current position, converting it to the codeset identified by the characterSet member variable.

WriteUTF

```
public void WriteUTF(String str)
```

Throws IOException.

Writes a UTF string, prefixed by a 2-byte length field, into the message buffer at the current position.

Properties

AccountingToken

```
public String AccountingToken {get; set;}
```

Part of the identity context of the message; it allows an application to charge for work done as a result of the message.

The default value is MQC.MQACT_NONE.

ApplicationIdData

```
public String ApplicationIdData {get; set;}
```

Part of the identity context of the message; it is information that is defined by the application suite, and can be used to provide additional information about the message or its originator.

The default value is "".

ApplicationOriginData

```
public String ApplicationOriginData {get; set;}
```

Information defined by the application that can be used to provide additional information about the origin of the message.

The default value is "".

BackoutCount

```
public int BackoutCount {get;}
```

A count of the number of times the message has previously been returned by an MQQueue.Get() call as part of a unit of work, and subsequently backed out.

The default value is zero.

CharacterSet

```
public int CharacterSet {get; set;}
```

The coded character set identifier of character data in the application message data. The behavior of the ReadString, ReadLine, and WriteString methods is altered accordingly.

The default value for this field is MQC.MQCCSI_Q_MGR. If the default value is used, CharacterSet 1200 (Unicode) is assumed. The following table shows coded character set identifiers and the characterSet values to use:

Table 1. Character set identifiers

characterSet	Description
37	ibm037
437	ibm437 / PC Original
500	ibm500
819	iso-8859-1 / latin1 / ibm819
1200	Unicode
1208	UTF-8
273	ibm273
277	ibm277
278	ibm278
280	ibm280
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
737	ibm737 / PC Greek
775	ibm775 / PC Baltic
813	iso-8859-7 / greek / ibm813
838	ibm838
850	ibm850 / PC Latin 1
852	ibm852 / PC Latin 2
855	ibm855 / PC Cyrillic
856	ibm856
857	ibm857 / PC Turkish
860	ibm860 / PC Portuguese
861	ibm861 / PC Icelandic
862	ibm862 / PC Hebrew
863	ibm863 / PC Canadian French
864	ibm864 / PC Arabic
865	ibm865 / PC Nordic
866	ibm866 / PC Russian
868	ibm868
869	ibm869 / PC Modern Greek
870	ibm870
871	ibm871
874	ibm874
875	ibm875
912	iso-8859-2 / latin2 / ibm912

Table 1. Character set identifiers (continued)

characterSet	Description
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cyrillic / ibm915
916	iso-8859-8 / hebrew / ibm916
918	ibm918
920	iso-8859-9 / latin5 / ibm920
921	ibm921
922	ibm922
930	ibm930
932	PC Japanese
933	ibm933
935	ibm935
937	ibm937
939	ibm939
942	ibm942
948	ibm948
949	ibm949
950	ibm950 / Big 5 Traditional Chinese
954	EUCJIS
964	ibm964 / CNS 11643 Traditional Chinese
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1089	iso-8859-6 / arabic / ibm1089
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1250	Windows Latin 2
1251	Windows Cyrillic
1252	Windows Latin 1
1253	Windows Greek
1254	Windows Turkish
1255	Windows Hebrew
1256	Windows Arabic
1257	Windows Baltic
1258	Windows Vietnamese
1381	ibm1381
1383	ibm1383
2022	JIS
5601	ksc-5601 Korean
33722	ibm33722

CorrelationId

```
public byte[] CorrelationId {get;set;}
```

For an MQQueue.Get() call, the correlation identifier of the message to be retrieved. Normally the queue manager returns the first message with a message identifier and correlation identifier that match those specified. The special value MQC.MQCI_NONE allows *any* correlation identifier to match.

For an MQQueue.Put() call, this specifies the correlation identifier to use.

The default value is MQC.MQCI_NONE.

DataLength

```
public int DataLength {get;}
```

The number of bytes of message data remaining to be read.

DataOffset

```
public int DataOffset {get; set;}
```

The current cursor position within the message data (the point at which read and write operations take effect).

Encoding

```
public int Encoding {get; set;}
```

The representation used for numeric values in the application message data; this applies to binary, packed decimal, and floating point data. The behavior of the read and write methods for these numeric formats is altered accordingly.

The following encodings are defined for binary integers:

MQC.MQENC_INTEGER_NORMAL
Big-endian integers.

MQC.MQENC_INTEGER_REVERSED
Little-endian integers, as used by PCs.

The following encodings are defined for packed-decimal integers:

MQC.MQENC_DECIMAL_NORMAL
Big-endian packed-decimal, as used by z/OS™.

MQC.MQENC_DECIMAL_REVERSED
Little-endian packed-decimal.

The following encodings are defined for floating-point numbers:

MQC.MQENC_FLOAT_IEEE_NORMAL
Big-endian IEEE floats.

MQC.MQENC_FLOAT_IEEE_REVERSED
Little-endian IEEE floats, as used by PCs.

MQC.MQENC_FLOAT_S390
z/OS format floating points.

Construct a value for the encoding field by adding together one value from each of these three sections (or using the bitwise OR operator). The default value is:

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

For convenience, this value is also represented by MQC.MQENC_NATIVE. This setting causes WriteInt() to write a big-endian integer, and ReadInt() to read a big-endian integer. If you set the flag

MQC.MQENC_INTEGER_REVERSED flag instead, WriteInt() writes a little-endian integer, and ReadInt() reads a little-endian integer.

A loss in precision can occur when converting from IEEE format floating points to zSeries® format floating points.

Expiry
`public int Expiry {get; set;}`

An expiry time expressed in tenths of a second, set by the application that puts the message. After a message's expiry time has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the MQC.MQRO_EXPIRATION flags, a report is generated when the message is discarded.

The default value is MQC.MQEI_UNLIMITED, meaning that the message never expires.

Feedback
`public int Feedback {get; set;}`

Used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. The following feedback codes are defined by the system:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used.

The default value of this field is MQC.MQFB_NONE, indicating that no feedback is provided.

Format
`public String Format {get; set;}`

A format name used by the sender of the message to indicate the nature of the data in the message to the receiver. You can use your own format names, but names beginning with the letters MQ have meanings that are defined by the queue manager. The queue manager built-in formats are:

MQC.MQFMT_ADMIN
Command server request/reply message.

MQC.MQFMT_COMMAND_1
Type 1 command reply message.

MQC.MQFMT_COMMAND_2
Type 2 command reply message.

MQC.MQFMT_DEAD_LETTER_HEADER
Dead-letter header.

MQC.MQFMT_EVENT
Event message.

MQC.MQFMT_NONE
No format name.

MQC.MQFMT_PCF
User-defined message in programmable command format.

MQC.MQFMT_STRING
Message consisting entirely of characters.

MQC.MQFMT_TRIGGER
Trigger message

MQC.MQFMT_XMIT_Q_HEADER
Transmission queue header.

The default value is MQC.MQFMT_NONE.

GroupId

```
public byte[] GroupId {get; set;}
```

A byte string that identifies the message group to which the physical message belongs.

The default value is MQC.MQGI_NONE.

MessageFlags

```
public int MessageFlags {get; set;}
```

Flags controlling the segmentation and status of a message.

MessageId

```
public byte[] MessageId {get; set;}
```

For an MQQueue.Get() call, this field specifies the message identifier of the message to be retrieved. Normally, the queue manager returns the first message with a message identifier and correlation identifier that match those specified. The special value MQC.MQMI_NONE allows *any* message identifier to match.

For an MQQueue.Put() call, this specifies the message identifier to use. If MQC.MQMI_NONE is specified, the queue manager generates a unique message identifier when the message is put. The value of this member variable is updated after the put, to indicate the message identifier that was used.

The default value is MQC.MQMI_NONE.

MessageLength

```
public int MessageLength {get;}
```

The number of bytes of message data in the MQMessage object.

MessageSequenceNumber

```
public int MessageSequenceNumber {get; set;}
```

The sequence number of a logical message within a group.

MessageType

```
public int MessageType {get; set;}
```

Indicates the type of the message. The following values are currently defined by the system:
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT
- MQC.MQMT_REQUEST

Application-defined values can also be used, in the range MQC.MQMT_APPL_FIRST to MQC.MQMT_APPL_LAST.

The default value of this field is MQC.MQMT_DATAGRAM.

Offset `public int Offset {get;}`

In a segmented message, the offset of data in a physical message from the start of a logical message.

OriginalLength
`public int OriginalLength {get;}`

The original length of a segmented message.

Persistence
`public int Persistence {get; set;}`

Message persistence. The following values are defined:
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

The default value is MQC.MQPER_PERSISTENCE_AS_Q_DEF, which takes the persistence for the message from the default persistence attribute of the destination queue.

Priority
`public int Priority {get; set;}`

The message priority. The special value MQC.MQPRI_PRIORITY_AS_Q_DEF can also be set in outbound messages, in which case the priority for the message is taken from the default priority attribute of the destination queue.

The default value is MQC.MQPRI_PRIORITY_AS_Q_DEF.

PutApplicationName
`public String PutApplicationName {get; set;}`

The name of the application that put the message. The default value is "".

PutApplicationType
`public int PutApplicationType {get; set;}`

The type of application that put the message. This can be a system-defined or user-defined value. The following values are defined by the system:
- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2

- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

The default value is the special value MQC.MQAT_NO_CONTEXT, which indicates that no context information is present in the message.

PutDateTime

```
public DateTime PutDateTime {get;}
```

The time and date that the message was put.

ReplyToQueueManagerName

```
public String ReplyToQueueManagerName {get; set;}
```

The name of the queue manager to which reply or report messages should be sent.

The default value is "".

If the value is "" on an MQQueue.put() call, the QueueManager fills in the value.

ReplyToQueueName

```
public String ReplyToQueueName {get; set;}
```

The name of the message queue to which the application that issued the get request for the message should send MQC.MQMT_REPLY and MQC.MQMT_REPORT messages.

The default value is "".

Report

```
public int Report {get; set;}
```

A report is a message about another message. This member variable enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and how to set the message and correlation identifiers in the report or reply. Any, all, or none of the following report types can be requested:

- Exception
- Expiration
- Confirm on arrival
- Confirm on delivery

For each type, only one of the three corresponding values below should be specified, depending on whether the application message data is to be included in the report message.

Note: Values marked with ** in the following list are not supported by z/OS queue managers; do not use them if your application is likely to access a z/OS queue manager, regardless of the platform on which the application is running.

The valid values are:

- MQC.MQRO_COA

- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**
- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**

You can specify one of the following to control how the message Id is generated for the report or reply message:
- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

You can specify one of the following to control how the correlation Id of the report or reply message is to be set:
- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

You can specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:
- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

If no report options are specified, the default is:
```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

You can specify one or both of the following to request that the receiving application sends a positive action or negative action report message.
- MQRO_PAN
- MQRO_NAN

TotalMessageLength
```
public int TotalMessageLength {get;}
```
The total number of bytes in the message as stored on the message queue from which this message was received.

UserId
```
public String UserId {get; set;}
```
Part of the identity context of the message; it identifies the user that originated this message.

The default value is "".

Version
```
public int Version {get; set;}
```
The version of the MQMD structure in use.

MQPutMessageOptions

```
System.Object
 │
 └─ IBM.WMQ.MQBase
     │
     └─ IBM.WMQ.MQBaseObject
         │
         └─ IBM.WMQ.MQPutMessageOptions
```

public class **IBM.WMQ.MQPutMessageOptions**
extends **IBM.WMQ.MQBaseObject**

This class contains options that control the behavior of MQQueue.put().

Constructors

MQPutMessageOptions
`public MQPutMessageOptions()`

Construct a new MQPutMessageOptions object with no options set, and a blank resolvedQueueName and resolvedQueueManagerName.

Properties

ContextReference
`public MQQueue ContextReference {get; set;}`

An input field that indicates the source of the context information.

If the `options` field includes MQC.MQPMO_PASS_IDENTITY_CONTEXT, or MQC.MQPMO_PASS_ALL_CONTEXT, set this field to refer to the MQQueue from which to take the context information.

The initial value of this field is null.

InvalidDestCount *
`public int InvalidDestCount {get;}`

An output field set by the queue manager to the number of messages that could not be sent to queues in a distribution list. The count includes queues that failed to open as well as queues that were opened successfully, but for which the put operation failed. This field is also set when opening a single queue that is not part of a distribution list.

KnownDestCount *
`public int KnownDestCount {get;}`

An output field set by the queue manager to the number of messages that the current call has sent successfully to queues that resolve to local queues. This field is also set when opening a single queue that is not part of a distribution list.

Options
`public int Options {get; set;}`

Options that control the action of MQQueue.put. Any or none of the following values can be specified. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

MQC.MQPMO_DEFAULT_CONTEXT
Associate default context with the message.

MQC.MQPMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQPMO_LOGICAL_ORDER*
Put logical messages and segments in message groups into their logical order.

MQC.MQPMO_NEW_CORREL_ID*
Generate a new correlation id for each sent message.

MQC.MQPMO_NEW_MSG_ID*
Generate a new message id for each sent message.

MQC.MQPMO_NONE
No options specified. Do not use in conjunction with other options.

MQC.MQPMO_NO_CONTEXT
No context is to be associated with the message.

MQC.MQPMO_NO_SYNCPOINT
Put a message without syncpoint control. Note that, if the syncpoint control option is not specified, a default of no syncpoint is assumed. This applies to all supported platforms.

MQC.MQPMO_PASS_ALL_CONTEXT
Pass all context from an input queue handle.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Pass identity context from an input queue handle.

MQC.MQPMO_SET_ALL_CONTEXT
Set all context from the application.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Set identity context from the application.

MQC.MQPMO_SYNCPOINT
Put a message with syncpoint control. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

RecordFields *

```
public int RecordFields {get; set;}
```

Flags indicating which fields are to be customized in each queue when putting a message to a distribution list. One or more of the following flags can be specified:

MQC.MQPMRF_ACCOUNTING_TOKEN
Use the accountingToken attribute in the MQDistributionListItem.

MQC.MQPMRF_CORREL_ID
Use the correlationId attribute in the MQDistributionListItem.

MQC.MQPMRF_FEEDBACK
Use the feedback attribute in the MQDistributionListItem.

MQC.MQPMRF_GROUP_ID
Use the groupId attribute in the MQDistributionListItem.

MQC.MQPMRF_MSG_ID
Use the messageId attribute in the MQDistributionListItem.

The special value MQC.MQPMRF_NONE indicates that no fields are to be customized.

ResolvedQueueManagerName

```
public String ResolvedQueueManagerName {get;}
```

An output field set by the queue manager to the name of the queue manager that owns the queue specified by the remote queue name. This might be different from the name of the queue manager from which the queue was accessed if the queue is a remote queue.

ResolvedQueueName

```
public String ResolvedQueueName {get;}
```

An output field that is set by the queue manager to the name of the queue on which the message is placed. This might be different from the name used to open the queue if the opened queue was an alias or model queue.

UnknownDestCount *

```
public int UnknownDestCount {get;}
```

An output field set by the queue manager to the number of messages that the current call has sent successfully to queues that resolve to remote queues. This field is also set when opening a single queue that is not part of a distribution list.

MQQueue

```
System.Object
  │
  └─ IBM.WMQ.MQBase
       │
       └─ IBM.WMQ.MQBaseObject
            │
            └─ IBM.WMQ.MQManagedObject
                 │
                 └─ IBM.WMQ.MQQueue
```

public class **IBM.WMQ.MQQueue**
extends **IBM.WMQ.MQManagedObject**. (See “MQManagedObject” on page 33.)

MQQueue provides inquire, set, put, and get operations for WebSphere MQ queues. The inquire and set capabilities are inherited from MQ.MQManagedObject.

See also “MQQueueManager.AccessQueue” on page 65.

Constructors

MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
               String alternateUserId )
```

Throws MQException.

Accesses a queue on the queue manager `qMgr`.

See “MQQueueManager.AccessQueue” on page 65 for details of the remaining parameters.

Methods

Close

```
public override void Close()
```

Overrides “MQManagedObject.Close” on page 33.

Get

```
public void Get(MQMessage message,
                MQGetMessageOptions getMessageOptions,
                int MaxMsgSize)
```

Throws MQException.

Retrieves a message from the queue, up to a maximum specified message size.

This method takes an MQMessage object as a parameter. It uses some of the fields in the object as input parameters, in particular the messageId and correlationId, so it is important to ensure that these are set as required.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor (member variables) and message data portions of the MQMessage are completely replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a given MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the get completes. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

Parameters

message
: An input/output parameter containing the message descriptor information and the returned message data.

getMessageOptions
: Options controlling the action of the get. (See “MQGetMessageOptions” on page 29.)

  Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQException.MQRC_CONVERTED_STRING_TOO_BIG when converting from single byte character codes to double byte codes. In this case, the message is copied into the buffer but remains encoded using its original character set.

MaxMsgSize
: The largest message this call can receive. If the message on the queue is larger than this size, one of two things occurs:

  1. If the MQC.MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the options member variable of the MQGetMessageOptions object, the message is filled with as much of the message data as will fit in the specified buffer size, and an exception is thrown with completion code MQException.MQCC_WARNING

and reason code MQException.MQRC_TRUNCATED_MSG_ACCEPTED.

2. If the MQC.MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue and an MQException is raised with completion code MQException.MQCC_WARNING and reason code MQException.MQRC_TRUNCATED_MSG_FAILED.

Throws MQException if the get fails.

Get

```
public void Get(MQMessage message,
                MQGetMessageOptions getMessageOptions)
```

Throws MQException.

Retrieves a message from the queue, regardless of the size of the message. For large messages, the get method might have to issue two calls to WebSphere MQ on your behalf, one to establish the required buffer size and one to get the message data itself.

This method takes an MQMessage object as a parameter. It uses some of the fields in the object as input parameters, in particular the messageId and correlationId, so it is important to ensure that these are set as required.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor (member variables) and message data portions of the MQMessage are completely replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a given MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the get completes. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

Parameters

message
: An input/output parameter containing the message descriptor information and the returned message data.

getMessageOptions
: Options controlling the action of the get. (See "MQGetMessageOptions" on page 29 for details.)

Throws MQException if the get fails.

Get

```
public void Get(MQMessage message)
```

A simplified version of the Get method previously described.

Parameters

MQMessage
: An input/output parameter containing the message descriptor information and the returned message data.

This method uses a default instance of MQGetMessageOptions to do the get. The message option used is MQGMO_NOWAIT.

Put

```
public void Put(MQMessage message,
              MQPutMessageOptions putMessageOptions)
```

Throws MQException.

Places a message onto the queue.

Note: For simplicity and performance, if you want to put just a single message to a queue, use the Put() method on your MQQueueManager object. For this you do not need to have an MQQueue object. See "MQQueueManager.Put" on page 69.

This method takes an MQMessage object as a parameter. The message descriptor properties of this object can be altered as a result of this method. The values that they have immediately after the completion of this method are the values that were put onto the WebSphere MQ queue.

Modifications to the MQMessage object after the put has completed do not affect the actual message on the WebSphere MQ queue.

A Put updates the messageId and correlationId. Consider this when making further calls to Put/Get using the same MQMessage object. Also, calling Put does not clear the message data, so:

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

puts two messages. The first contains `a` and the second `ab`.

Parameters

message
: Message Buffer containing the Message Descriptor data and message to be sent.

putMessageOptions
: Options controlling the action of the put. (See "MQPutMessageOptions" on page 54)

Throws MQException if the put fails.

Put

```
public void Put(MQMessage message)
```

A simplified version of the Put method previously described.

Parameters

MQMessage
: Message Buffer containing the Message Descriptor data and message to be sent.

This method uses a default instance of MQPutMessageOptions to do the put.

Properties

CreationDateTime

```
public DateTime CreationDateTime {get;}
```

Throws MQException.

The date and time that this queue was created.

QueueType

```
public int QueueType {get;}
```

Throws MQException

Returns

The type of this queue with one of the following values:

- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

CurrentDepth

```
public int CurrentDepth {get;}
```

Throws MQException.

Gets the number of messages currently on the queue. This value is incremented during a put call, and during backout of a get call. It is decremented during a non-browse get and during backout of a put call.

DefinitionType

```
public int DefinitionType {get;}
```

Throws MQException.

How the queue was defined.

Returns

One of the following:

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

InhibitGet

```
public int InhibitGet {get; set;}
```

Throws MQException.

get

Whether get operations are allowed for this queue.

Returns

The possible values are:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

set

Controls whether get operations are allowed for this queue. The permissible values are:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

InhibitPut

```
public int InhibitPut {get; set;}
```

Throws MQException.

get

Whether put operations are allowed for this queue.

Returns

One of the following:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

set

Controls whether put operations are allowed for this queue. The permissible values are:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

MaximumDepth

```
public int MaximumDepth {get;}
```

Throws MQException.

The maximum number of messages that can exist on the queue at any one time. An attempt to put a message to a queue that already contains this many messages fails with reason code MQException.MQRC_Q_FULL.

MaximumMessageLength

```
public int MaximumMessageLength {get;}
```

Throws MQException.

The maximum length of the application data that can exist in each message on this queue. An attempt to put a message larger than this value fails with reason code MQException.MQRC_MSG_TOO_BIG_FOR_Q.

OpenInputCount

```
public int OpenInputCount {get;}
```

Throws MQException.

The number of handles that are currently valid for removing messages from the queue. This is the *total* number of such handles known to the local queue manager, not just those created by the WebSphere MQ classes for .NET (using accessQueue).

OpenOutputCount

```
public int OpenOutputCount {get;}
```

Throws MQException.

The number of handles that are currently valid for adding messages to the queue. This is the *total* number of such handles known to the local queue manager, not just those created by the WebSphere MQ classes for .NET (using accessQueue).

Shareability

```
public int Shareability {get;}
```

Throws MQException.

Whether the queue can be opened for input multiple times.

Returns

One of the following:

- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

TriggerControl

```
public int TriggerControl {get; set;}
```

Throws MQException.

get

Whether trigger messages are written to an initiation queue, to start an application to service the queue.

Returns

The possible values are:

- MQC.MQTC_OFF
- MQC.MQTC_ON

set

Controls whether trigger messages are written to an initiation queue to start an application to service the queue. The permissible values are:

- MQC.MQTC_OFF
- MQC.MQTC_ON

TriggerData

```
public String TriggerData {get; set;}
```

Throws MQException.

get

The free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

set

Sets the free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue. The maximum permissible length of the string is given by MQC.MQ_TRIGGER_DATA_LENGTH.

TriggerDepth

```
public int TriggerDepth {get; set;}
```

Throws MQException.

get

The number of messages that have to be on the queue before a trigger message is written when trigger type is set to MQC.MQTT_DEPTH.

set

Sets the number of messages that have to be on the queue before a trigger message is written when trigger type is set to MQC.MQTT_DEPTH.

TriggerMessagePriority

```
public int TriggerMessagePriority {get; set;}
```

Throws MQException.

get

The message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether to generate a trigger). A value of zero causes all messages to contribute to the generation of trigger messages.

set

Sets the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether a trigger should be generated). A value of zero causes all messages to contribute to the generation of trigger messages.

TriggerType

```
public int TriggerType {get; set;}
```

Throws MQException.

get

The conditions under which trigger messages are written as a result of messages arriving on this queue.

Returns

The possible values are:

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

set

Sets the conditions under which trigger messages are written as a result of messages arriving on this queue. The possible values are:

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

MQQueueManager

```
System.Object
│
└─ IBM.WMQ.MQBase
   │
   └─ IBM.WMQ.MQBaseObject
      │
      └─ IBM.WMQ.ManagedObject
         │
         └─ IBM.WMQ.MQQueueManager
```

public class **IBM.WMQ.MQQueueManager**
extends **IBM.WMQ.MQManagedObject**. (See "MQManagedObject" on page 33.)

The MQQueueManager encapsulates the MQCONN. It has an overloaded constructor that can be used to perform client/server connections to a QueueManager.

The MQQueueManager contains a method 'AccessQueue', which is used to instantiate an MQQueue object associated with the connected MQQueueManager object.

The MQQueueManager class also contains methods to begin, commit, and rollback transactions.

Constructors

MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

Throws MQException.

Creates a connection to the named queue manager.

Note: When using WebSphere MQ classes for .NET, the hostname, channel name, and port to use during the connection request are specified in the MQEnvironment class. This must be done *before* calling this constructor.

The following example shows a connection to a queue manager MYQM, running on a machine with hostname fred.mq.com.

```
MQEnvironment.Hostname = "fred.mq.com";  // host to connect to
MQEnvironment.Port     = 1414;           // port to connect to.
                                         // If I don't set this,
                                         // it defaults to 1414
                                         // (the default WebSphere MQ port)
MQEnvironment.Channel  = "channel.name"; // the CASE-SENSITIVE
```

```
                                          //  name of the
                                          // SVR CONN channel on
                                          // the queue manager
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

If the queue manager name is left blank (null or ""), a connection is made to the default queue manager.

See also “MQEnvironment” on page 27.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

Throws MQException.

This version of the constructor is intended for use only in bindings mode. It uses the extended connection API (MQCONNX) to connect to the queue manager. The *options* parameter allows you to choose fast or normal bindings. Possible values are:

- MQC.MQCNO_FASTPATH_BINDING for fast bindings *.
- MQC.MQCNO_STANDARD_BINDING for normal bindings.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

Throws MQException.

Performs an MQCONNX, passing the supplied options. The specified ConnectionManager manages the connection.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties)
```

Throws MQException.

The properties parameter takes a series of key/value pairs that describe the WebSphere MQ environment for this particular queue manager. These properties, where specified, override the values set by the MQEnvironment class, and allow the individual properties to be set on a queue manager by queue manager basis.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      String channel,
                      String connName)
```

Throws MQException.

Connects to the named Queue Manager, using the supplied ’Server Connection Channel’ and connection.

Methods

AccessQueue

```
public  MQQueue AccessQueue
               (String queueName, int openOptions,
                String queueManagerName,
                String dynamicQueueName,
                String alternateUserId)
```

Throws MQException.

Establishes access to a WebSphere MQ queue on this queue manager to get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue.

If the queue named is a model queue, a dynamic local queue is created. The name of the created queue can be determined from the `name` attribute of the returned MQQueue object.

Parameters

queueName
: Name of queue to open.

openOptions
: Options that control the opening of the queue. Valid options are:

MQC.MQOO_ALTERNATE_USER_AUTHORITY
: Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
: Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
: Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
: Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
: Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
: Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
: Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
: Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
: Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
: Open for inquiry - required if you wish to query properties.

MQC.MQOO_OUTPUT
: Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
: Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
: Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
: Save context when message retrieved*.

MQC.MQOO_SET
: Open to set attributes —required if you wish to set properties.

MQC.MQOO_SET_ALL_CONTEXT
: Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
: Allows identity context to be set.

If more than one option is required, the values can be added together or combined using the bitwise OR operator. See the *WebSphere MQ Application Programming Reference* for a fuller description of these options.

queueManagerName
: Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which this MQQueueManager object is connected.

dynamicQueueName
: This parameter is ignored unless queueName specifies the name of a model queue. If it does, this parameter specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last non-blank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique on this queue manager.

alternateUserId
: If MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternate user identifier that is used to check the authorization for the open. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, this parameter can be left blank (or null).

Returns
: MQQueue that has been successfully opened.

Throws MQException if the open fails.

AccessQueue

```
public MQQueue AccessQueue
            (String queueName,
             int openOptions)
```

Throws MQException if you call this method after disconnecting from the queue manager.

Parameters

queueName
: Name of queue to open

openOptions
: Options that control the opening of the queue

See the description of “MQQueueManager.AccessQueue” on page 65 for details of the parameters.

For this version of the method, *queueManagerName*, *dynamicQueueName*, and *alternateUserId* are set to `""`.

Returns
MQQueue that has been successfully opened.

Throws MQException if the open fails.

Backout

```
public  void Backout()
```

Throws MQException.

Calling this method indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are deleted; messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in the options field of MQGetMessageOptions) are reinstated on the queue.

See also the description of the commit method.

Begin*

```
public void Begin()
```

Throws MQException.

This method is supported only by the WebSphere MQ classes for .NET in server bindings mode. It signals to the queue manager that a new unit of work is starting.

Do not use this method for applications that use local one-phase transactions.

Commit

```
public void Commit()
```

Throws MQException.

Calling this method indicates to the queue manager that the application has reached a syncpoint, and that all the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are made available to other applications. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in the options field of MQGetMessageOptions) are deleted.

See also the description of the backout method.

Disconnect

```
public void Disconnect()
```

Throws MQException.

Terminates the connection to the queue manager. All open queues and processes accessed by this queue manager are closed, and become unusable. When you have disconnected from a queue manager, the only way to reconnect is to create a new MQQueueManager object.

Normally, any work performed as part of a unit of work is committed. However, if this connection is managed by a ConnectionManager, rather than an MQConnectionManager, the unit of work might be rolled back.

Put

```
public void Put(String qName,
              String qmName,
              MQMessage msg,
              MQPutMessageOptions pmo,
              String altUserId)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

The qName (queue name) and qmName (queue manager name) parameters identify where the message is placed. If the queue is a model queue, an MQException is thrown.

In other respects, this method behaves like the put method on the MQQueue object. It is an implementation of the MQPUT1 MQI call. See "MQQueue.Put" on page 59.

Parameters

qName The name of the queue onto which to place the message.

qmName
The name of the queue manager on which the queue is defined.

msg The message to send.

pmo Options controlling the actions of the put. See "MQPutMessageOptions" on page 54 for more details.

altUserid
Specifies an alternative user identifier used to check authorization when placing the message on a queue. If you do **not** specify MQPMO_ALTERNATE_USER, this parameter is ignored.

Put

```
public void Put(String qName,
              String qmName,
              MQMessage msg,
              MQPutMessageOptions pmo)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the altUserid parameter. See the fully-specified method ("MQQueueManager.Put") for details of the parameters.

Put

```
public void Put(String qName,
                String qmName,
                MQMessage msg)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the put message options (pmo) and altUserid parameters. See the fully-specified method ("MQQueueManager.Put" on page 69) for details of the parameters.

Put

```
public void Put(String qName,
                MQMessage msg,
                MQPutMessageOptions pmo)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the qmName and altUserid parameters. See the fully-specified method ("MQQueueManager.Put" on page 69) for details of the parameters.

Put

```
 public void Put(String qName,
                 MQMessage msg)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the qmName, put message options (pmo), and altUserid parameters. See the fully-specified method ("MQQueueManager.Put" on page 69) for details of the parameters.

Properties

CharacterSet

```
public int CharacterSet {get;}
```

Throws MQException.

Returns the CCSID (Coded Character Set Identifier) of the queue manager's codeset. This defines the character set used by the queue manager for all character string fields in the application programming interface.

Throws MQException if you call this method after disconnecting from the queue manager.

CommandInputQueueName

```
public String CommandInputQueueName {get;}
```

Throws MQException.

Returns the name of the command input queue defined on the queue manager. This is a queue to which applications can send commands, if authorized to do so.

Throws MQException if you call this method after disconnecting from the queue manager.

CommandLevel

```
public int CommandLevel {get;}
```

Throws MQException.

Indicates the level of system control commands supported by the queue manager. The set of system control commands that correspond to a particular command level varies according to the architecture of the platform on which the queue manager is running. See the WebSphere MQ documentation for your platform for further details.

Throws MQException if you call this method after disconnecting from the queue manager.

Returns
One of the MQC.MQCMDL_LEVEL_xxx constants

IsConnected

```
public boolean IsConnected {get;}
```

Returns the value of the isConnected variable.

MaximumMessageLength

```
public int MaximumMessageLength {get;}
```

Throws MQException.

Returns the maximum length of a message (in bytes) that can be handled by the queue manager. No queue can be defined with a maximum message length greater than this.

Throws MQException if you call this method after disconnecting from the queue manager.

MaximumPriority

```
public int MaximumPriority {get;}
```

Throws MQException.

Returns the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to this value.

Throws MQException if you call this method after disconnecting from the queue manager.

SyncpointAvailability

```
public int SyncpointAvailability {get;}
```

Throws MQException.

Indicates whether the queue manager supports units of work and syncpointing with the MQQueue.get and MQQueue.put methods.

Returns

- MQC.MQSP_AVAILABLE if syncpointing is available.
- MQC.MQSP_NOT_AVAILABLE if syncpointing is not available.

Throws MQException if you call this method after disconnecting from the queue manager.

MQC

```
System.Object
 │
 └─ IBM.WMQ.MQC
```

public interface **IBM.WMQ.MQC**
extends **System.Object**

The MQC interface defines all the constants used by the MQI (except for completion code constants and error code constants). To refer to one of these constants from within your programs, prefix the constant name with MQC.. For example, you can set the close options for a queue as follows:

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // delete the
                                      // queue when
                                      // it is closed
 ...
```

A full description of these constants is in the *WebSphere MQ Application Programming Reference*.

Completion code and error code constants are defined in the MQException class. See “MQException” on page 28.

Part 3. Appendixes

Appendix A. SSL CipherSpecs supported by WebSphere MQ

The following table lists the CipherSpecs supported by WebSphere MQ. Specify the CipherSpec name in the SSLCIPH property of the SVRCONN channel on the queue manager and in MQEnvironment.SSLCipherSpec

Table 2. Supported CipherSpecs

CipherSpec
DES_SHA_EXPORT
DES_SHA_EXPORT1024
NULL_MD5
NULL_SHA
RC2_MD5_EXPORT
RC4_56_SHA_EXPORT1024
RC4_MD5_US
RC4_MD5_EXPORT
RC4_SHA_US
TRIPLE_DES_SHA_US

Appendix B. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information 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 information. The furnishing of this information 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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information 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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

Index

Sending your comments to IBM

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 and the way in which the information is presented.

To make comments about the functions of IBM products or systems, 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, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

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