WebSphere MQ

IBM

Using .NET

Version 7.0

GC34-6949-01

WebSphere MQ

Using .NET

Version 7.0

GC34-6949-01

Note

Before using this information and the product it supports, be sure to read the general information under notices at the back of this book.

Second edition (January 2009)

This edition of the book applies to the following:

- IBM WebSphere MQ, Version 7.0

and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Chapter 1. Guidance for users

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 or to connect directly to a WebSphere MQ server.

Connection options

There are three modes of connecting WebSphere MQ classes for .NET to a queue manager. Consider which type of connection best suits your requirements.

Client bindings connection

To use WebSphere MQ classes for .NET as a WebSphere MQ client, you can install it, with the WebSphere MQ Client, either on the WebSphere MQ server machine, or on a separate machine. A client bindings connection can use XA or non-XA transactions

Server bindings connection

When used in server bindings mode, WebSphere MQ classes for .NET use 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.

Managed client connection

A connection made in this mode connects as a WebSphere MQ client to a WebSphere MQ server running either on the local or a remote machine.

The WebSphere MQ classes for .NET connecting in this mode remain in .NET managed code and make no calls to native services. For more information about managed code, refer to Microsoft® documentation.

There are a number of limitations to using the managed client. For more information about these, see “Managed client connections” on page 8.

Installation

WebSphere MQ classes for .NET, including samples, is installed with WebSphere MQ. There is a prerequisite of Microsoft .NET Framework.

The latest version of WebSphere MQ classes for .NET is installed as part of the standard WebSphere MQ installation. You might need to override default installation options to make sure this is done. For installation instructions, see *WebSphere MQ for Windows Quick Beginnings*.

Sample applications, including source, are also supplied; see "Sample applications."

To run WebSphere MQ classes for .NET on 32–bit or 64–bit platforms you must have installed Microsoft .NET Framework (v2.0) or a later version.

Using WebSphere MQ classes for .NET

This collection of topics describes how to configure your system to run the sample programs to verify your WebSphere MQ classes for .NET installation, and how to run your own programs.

Configuring your queue manager to accept TCP/IP client connections

To configure a queue manager to accept incoming connection requests from the clients:

1. Define a server connection channel:
   a. Start the queue manager.
   b. Define a sample channel called NET.CHANNEL[1]:
      ```
      DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for WebSphere MQ classes for .NET')
      ```
2. Start a listener:
   ```
   runmqlsr -t tcp [-m qmnqme] [-p portnum]
   ```

   Note: The square brackets indicate optional parameters; *qmname* is not required for the default queue manager, and the port number *portnum* is not required if you are using the default (1414).

Sample applications

Five sample applications are supplied:

- A put message application
- A get message application
- A 'hello world' application
- | A publish/subscribe application
- | An application using message properties

"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 three parameters:
- The name of a queue (required) for example SYSTEM.DEFAULT.LOCAL.QUEUE
- The name of a queue manager (optional)
- The definition of a channel (optional) for example SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

1. In this sample, we are not considering security implications. For a production system, consider using SSL or a security exit. See WebSphere MQ Security for more information.

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 three parameters:

- The name of a queue (required) for example SYSTEM.DEFAULT.LOCAL.QUEUE
- The name of a queue manager (optional)
- The definition of a channel (optional) for example 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 three parameters:

- The name of a queue (optional) for example SYSTEM.DEFAULT.LOCAL.QUEUE or SYSTEM.DEFAULT.MODEL.QUEUE
- The name of a queue manager (optional)
- A channel definition (optional) for example 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.

| **″Publish/subscribe″ program (MQPubSubSample.cs)**
| This program shows how to use WebSphere MQ publish/subscribe. It is
| supplied in C# only. The program has two parameters:

| - The name of a queue manager (optional)
| - A channel definition (optional)

| **″Message properties″ program (MQMessagePropertiesSample.cs)**
| This program shows how to use message properties. It is supplied in C#
| only. The program has two parameters:

| - The name of a queue manager (optional)
| - A channel definition (optional)

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

| *mqmtop*\Tools\dotnet\samples\cs\MQPubSubSample.cs

| *mqmtop*\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

Managed C++

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

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

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

Visual Basic

mqmtop\Tools\dotnet\samples\vb\vmqswrld.vb

mqmtop\Tools\dotnet\samples\vb\vmqsput.vb

mqmtop\Tools\dotnet\samples\vb\vmqsget.vb

mqmtop\Tools\dotnet\samples\vb\xmqswrld.vb

mqmtop\Tools\dotnet\samples\vb\xmqsput.vb

mqmtop\Tools\dotnet\samples\vb\xmqsget.vb

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

C#

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

The bldcssamp.bat file 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 bldmcpsamp.bat file contains a line for each sample, which is all that
| is necessary to build this sample program:

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

| If you want to compile these applications on Microsoft Visual Studio
| 2003/.NET SDKv1.1, replace the compile command:

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

| with

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

Visual Basic

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

The bldvbsamp.bat file 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 instructions 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 Chapter 2, "Programming with WebSphere MQ classes for .NET," on page 7.

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 “Using WebSphere MQ classes for .NET” on page 2.

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 for either amqmdnet.dll or amqmdxcs.dll, either ensure that both are registered in the ’Global Assembly Cache’ or create a configuration file that points to the amqmdnet.dll and amqmdxcs.dll assemblies. You can examine and change the contents of the assembly cache using mscorcfg.msc, which is supplied as part of the .NET framework.

If the .NET framework was not available when WebSphere MQ was installed, the classes might not be registered in the global assembly cache. You can manually rerun the registration process using the command

```
amqidnet -c mqmtop\bin\amqidotn.txt -l logfile.txt
```

Information about this installation is written to the specified log file (**logfile.txt** in this example).

Chapter 2. Programming with WebSphere MQ classes for .NET

Introduction for programmers

This topic contains general information for programmers.

For more detailed information about writing programs, see “Writing WebSphere MQ .NET programs” on page 8.

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

Rather than using the MQI verbs, the object-oriented WebSphere MQ .NET interface uses methods of objects.

The procedural WebSphere MQ application programming interface is built around verbs such as those listed below:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

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();
```

Prerequisite for compiling WebSphere MQ .NET applications

Before you can compile any applications that you write, you must have a .NET Framework installed.

For more information, see “Installation” on page 1.

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.

They represent the various components of WebSphere MQ which your application uses, such as queue managers, queues, channels and messages.

For details of these classes, see "The WebSphere MQ .NET classes and interfaces" on page 27

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 "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.

Managed client connections

When WebSphere MQ classes for .NET are used as a managed client, it is similar to a client bindings connection, but has a number of differences.

The following features are not available:

- Channel compression
- SSL support
- XA transactions
- | Channel exit chaining

If you try to use these features with a managed client, it will return an MQException. If the error is detected at the client end of a connection, it will use reason code MQRC_ENVIRONMENT_ERROR. If it is detected at the server end, the reason code returned by the server will be used.

| Channel exits written for a non-managed client do not work. You must write new
| exits specifically for the managed client. Check that there are no invalid channel
| exits specified in your client channel definition table (CCDT).

Communication is supported only over TCP/IP.

| When you stop a queue manager using the endmqm command, a
| server-connection channel to a .NET managed client can take longer to close than
| server-connection channels to other clients.

| If you use the trace facility, you cannot choose to trace specific threads or
| processes.

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

Defining which connection type to use

The connection type is determined by the setting of the connection name, channel name, the customization value NMQ_MQ_LIB and the property MQC.TRANSPORT_PROPERTY.

You can specify the connection name as follows:

- Explicitly on an MQQueueManager constructor:
  ```
  public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
  string ConnName)
  ```
  ```
  public MQQueueManager(String queueManagerName, string Channel, string ConnName)
  ```
- By setting the properties MQC.HOST_NAME_PROPERTY and, optionally, MQC.PORT_PROPERTY in a hashtable entry on an MQQueueManager constructor:
  ```
  public MQQueueManager(String queueManagerName, Hashtable properties)
  ```
- As explicit MQEnvironment values
  ```
  MQEnvironment.Hostname
  ```
  `MQEnvironment.Port`(optional).
- By setting the properties MQC.HOST_NAME_PROPERTY and, optionally, MQC.PORT_PROPERTY in the MQEnvironment.properties hashtable.

You can specify the channel name as follows:

- Explicitly on an MQQueueManager constructor:
  ```
  public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
  string ConnName)
  ```
  ```
  public MQQueueManager(String queueManagerName, string Channel, string ConnName)
  ```
- By setting the property MQC.CHANNEL_PROPERTY in a hashtable entry on an MQQueueManager constructor:
  ```
  public MQQueueManager(String queueManagerName, Hashtable properties)
  ```
- As an explicit MQEnvironment value
  ```
  MQEnvironment.Channel
  ```
- By setting the property MQC.CHANNEL_PROPERTY in the MQEnvironment.properties hashtable.

| You can specify the transport property as follows:

- | By setting the property MQC.TRANSPORT_PROPERTY in a hashtable entry on an MQQueueManager constructor:
  ```
  public MQQueueManager(String queueManagerName, Hashtable properties)
  ```
- | By setting the property MQC.TRANSPORT_PROPERTY in the MQEnvironment.properties hashtable.

| Select the connection type you require by using one of the following values:

| MQC.TRANSPORT_MQSERIES_BINDINGS - connect as server

| MQC.TRANSPORT_MQSERIES_CLIENT - connect as non-XA client

| MQC.TRANSPORT_MQSERIES_XACLIENT - connect as XA client

| MQC.TRANSPORT_MQSERIES_MANAGED - connect as non-XA managed client

You can set the customization value NMQ_MQ_LIB to explicitly choose the connection type as shown in the following table

NMQ_MQ_LIB value	Connection type
mqic.dll	Connect as a non-XA client
mqicxa.dll	Connect as an XA client
mqm.dll	Connect as a server
managed	Connect as a non-XA managed client
Note: Values of mqic32.dll and mqic32xa.dll are accepted as synonyms of mqic.dll and mqicxa.dll for comatibility with earlier releases.	

If you choose a connection type which is unavailable in your environment, for example you specify mqic32xa.dll and don't have XA support, WebSphere MQ .NET throws an exception.

Setting NMQ_MQ_LIB to ″managed″ causes the client to use managed WebSphere MQ problem diagnostics, .NET data conversion, and other managed low-level WebSphere MQ functions.

All other values for NMQ_MQ_LIB cause the .NET process to use unmanaged WebSphere MQ problem diagnostics and data conversion, and other unmanaged low-level WebSphere MQ functions (assuming a WebSphere MQ client or server is installed on the system).

WebSphere MQ .NET chooses the connection type as follows:

1. If MQC.TRANSPORT_PROPERTY is specified, it connects according to the value of MQC.TRANSPORT_PROPERTY.

   Note, however, that setting MQC.TRANSPORT_PROPERTY to MQC.TRANSPORT_MQSERIES_MANAGED does not guarantee that the client process runs managed. Even with this setting, the client is not managed in the following cases:
   - If another thread in the process has connected with MQC.TRANSPORT_PROPERTY set to something other than MQC.TRANSPORT_MQSERIES_MANAGED.
   - If NMQ_MQ_LIB is not set to ″managed″, problem diagnostics, data conversion and other low-level functions are not fully managed (assuming a WebSphere MQ client or server is installed on the system).
2. If a connection name has been specified without a channel name, or a channel name has been specified without a connection name, it throws an error.
3. If both a connection name and a channel name have been specified:
   - If NMQ_MQ_LIB is set to mqic32xa.dll, it connects as an XA client.
   - If NMQ_MQ_LIB is set to managed, it connects as a managed client.
   - Otherwise it connects as a non-XA client.
4. If NMQ_MQ_LIB is specified, it connects according to the value of NMQ_MQ_LIB.
5. If a WebSphere MQ server is installed, it connects as a server.
6. If a WebSphere MQ client is installed, it connects as a non-XA client.
7. Otherwise, it connects as a managed client.

Client configuration files

A WebSphere MQ classes for .NET client application can use a client configuration file in the same way as any other WebSphere MQ client.

This file is typically called mqclient.ini, but you can specify a different file name. For more information about the client configuration file, see WebSphere MQ client configuration file.

Only the following attributes in a WebSphere MQ client configuration file are relevant to WebSphere MQ classes for .NET. If you specify other attributes, it has no effect.

Stanza	**Attribute**
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ServerConnectionParms
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

You can override any of these attributes using the appropriate environment variable. You can also override the WebSphere MQ client configuration file and the equivalent environment variables using the .NET Application Configuration File. The format of the stanzas, variable names, and variable values in the .NET Application Configuration file is illustrated in the following example for TCP/IP KeepAlive:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

See your Microsoft documentation for further information.

Example code fragment

The following C# code fragment demonstrates an application that performs three actions:

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

It also shows how to change the connection type.

```
// ===========================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2005
// ===========================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);
```

```
        // 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 WebSphere MQ message, writing 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
        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);

        // Prove we have the message by displaying the UTF message text
        String msgText = retrievedMessage.ReadUTF();
        Console.WriteLine("The message is: {0}", 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: {0}", ex.ToString());
      }

      catch (System.Exception ex)
      {
        Console.WriteLine("A System error occurred: {0}", ex.ToString());
      }

      return 0;
    }//end of start
  }//end of sample
```

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

Before you use the client connection to connect to a queue manager, you must set up the WebSphere MQ environment.

Note: This step is not necessary when using WebSphere MQ classes for .NET in server bindings mode.

The .NET programming interface allows you to use the NMQ_MQ_LIB customization value but also includes a class MQEnvironment. This class allows you to specify details that are to be used during the connection attempt, such as those in the following list:

- Channel name
- Host name
- Port number
- Channel exits
- SSL parameters
- User ID and password

For full information about the MQEnvironment class, see "MQEnvironment" on page 36

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 on the object associated with each resource.

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 topics

You can access queues and topics using methods of MQQueueManager or appropriate constructors.

To access queues, use the methods of 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 represented by an MQQueueManager object 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 specifying the instantiated MQQueueManager object to use. 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.

Similarly, you can also access topics using the methods of the MQQueueManager class. Use an AccessTopic() method to open a topic. This returns a new object of class MQTopic. When you have finished using the topic, use the Close() method of the MQTopic to close it.

You can also create a topic by using an MQTopic constructor. There are a number of constructors for topics; for more information see “Constructors for MQTopic” on page 134.

Handling messages

Messages are handled using the methods of the queue or topic classes. To build a new message, create a new MQMessageobject.

Put messages onto queues or topics using the Put() method of the MQQueue or MQTopic class. Get messages from queues or topics using the Get() method of the MQQueue or MQTopic 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 47 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 message properties

Message properties allow you to select messages, or to retrieve information about a message without accessing its headers. The MQMessage class contains methods to get and set properties.

You can use message properties to allow an application to select messages to process, or to retrieve information about a message without accessing MQMD or MQRFH2 headers. They also facilitate communication between Websphere MQ and

JMS applications. For a discussion of message properties in Websphere MQ, see the *WebSphere MQ System Administration Guide*.

The MQMessage class provides a number of methods to get and set properties, according to the data type of the property. The get methods have names of the format Get*Property, and the set methods have names of the format Set*Property, where the asterisk (*) represents one of the following strings:

- Boolean
- Byte
- Bytes
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- Object
- Short
- String

For example, to get the Websphere MQ property myproperty (a character string), use the call `message.GetStringProperty('myproperty')`. You can optionally pass a property descriptor, which WebSphere MQ will complete.

Handling errors

Handle errors arising from WebSphere MQ classes for .NET using `try` and `catch` blocks.

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 “Reason Codes” in *WebSphere MQ Messages*.

Getting and setting attribute values

The classes MQManagedObject, MQQueue, and MQQueueManager contain methods 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 common attributes, the MQQueueManager and MQQueue classes 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 could 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
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

All attributes of these objects can be inquired on. A subset of attributes is exposed as the properties of an object. For a list of object attributes, see *WebSphere MQ Application Programming Reference*. For object properties, see the appropriate class description.

Multithreaded programs

The .NET runtime environment is inherently multithreaded. WebSphere MQ classes for .NET allows a queue manager object to be shared across multiple threads but ensures that all access to the target queue manager is synchronized.

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. In this situation, the 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.

Using a client channel definition table

The .NET classes for WebSphere MQ support the use of client definition tables through the environment variables MQCHLLIB and MQCHLTAB.

MQCHLLIB specifies the directory where the table is located and MQCHLTAB specifies the actual filename of the table.

The client channel definition table is described in *WebSphere MQ Clients*.

Using channel exits in WebSphere MQ .NET

If you use client bindings, you can use channel exits as for any other client connection. If you use managed bindings, you must write an exit program that implements an appropriate interface.

Client bindings

If you use client bindings, you can use channel exits as described in *WebSphere MQ Clients*. You cannot use channel exits written for managed bindings.

Managed bindings

If you use a managed connection, to implement an exit, you define a new .NET class that implements the appropriate interface. Three exit interfaces are defined in the WebSphere MQ package:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Note: User exits written using these interfaces are not supported as channel exits in the unmanaged environment.

The following sample defines a class that implements all three:

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // fill in the body of the send exit here
  }

  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // fill in the body of the receive exit here
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // fill in the body of the security exit here
  }

}
```

Each exit is passed an MQChannelExit and an MQChannelDefinition object instance. These objects represent the MQCXP and MQCD structures defined in the procedural interface.

The data to be sent by a send exit, and the data received in a security or receive exit is specified using the exit's parameters.

On entry, the data at offset *dataOffset* with length *dataLength* in the byte array *dataBuffer* is the data that is about to be sent by a send exit, and the data received in a security or receive exit. The parameter *dataMaxLength* gives the maximum length (from *dataOffset*) available to the exit in *dataBuffer*. Note: For a security exit, it is possible for the dataBuffer to be null, if this is the first time the exit is called or the partner end elected to send no data.

On return, the value of *dataOffset* and *dataLength* should be set to point to the offset and length within the returned byte array that the .NET classes should then use. For a send exit, this indicates the data that it should send, and for a security or receive exit, the data that should be interpreted. The exit should normally return a byte array; exceptions are a security exit which could elect to send no data, and any exit called with the INIT or TERM reasons. The simplest form of exit that can be written therefore is one which does nothing more than return dataBuffer:

The simplest possible exit body is:

```
{
  return dataBuffer;
}
```

Specifying channel exits (managed client)

If you specify a channel name and connection name when creating your MQQueueManager object (either in the MQEnvironment or on the MQQueueManager constructor) you can specify channel exits in two ways.

In order of precedence, these are:

1. Passing hashtable properties MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY or MQC.RECEIVE_EXIT_PROPERTY on the MQQueueManager constructor.
2. Setting the MQEnvironment SecurityExit, SendExit or ReceiveExit properties.

If you do not specify a channel name and connection name, the channel exits to use come from the channel definition picked up from a client channel definition table. It is not possible to override the values stored in the channel definition. See WebSphere MQ Clients for more information about channel definition tables.

In each case, the specification takes the form of a string with the following format:

```
Assembly_name(Class_name)
```

Class_name is the fully qualified name, including namespace specification, of a .NET class that implements the IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit or IBM.WMQ.MQReceiveExit interface (as appropriate). *Assembly_name* is the fully qualified location, including file extension, of the assembly that houses the class. The length of the string is limited to 128 characters. When necessary, the .NET client code loads and creates an instance of the specified class by parsing the string specification.

Specifying channel exit user data (managed client)

Channel exits can have user data associated with them. If you specify a channel name and connection name when creating your MQQueueManager object (either in the MQEnvironment or on the MQQueueManager constructor) you can specify the user data in two ways.

In order of precedence, these are:

1. Passing hashtable properties MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY or MQC.RECEIVE_USERDATA_PROPERTY on the MQQueueManager constructor.
2. Setting the MQEnvironment SecurityUserData, SendUserData or ReceiveUserData properties.

If you do not specify a channel name and connection name, the exit user data values to use come from the channel definition picked up from the client channel definition table. It is not possible to override the values stored in the channel definition. See WebSphere MQ Clients for more information about channel definition tables.

In each case, the specification is a string, limited to 32 characters.

Secure Sockets Layer (SSL) support

The following section does not apply to the managed client.

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 *WebSphere MQ Security* 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.

Using the .NET Monitor

The .NET Monitor is an application similar to a WebSphere MQ trigger monitor. You can create .NET components which will be instantiated whenever a message is received on a monitored queue, and which will then process that message. The .NET Monitor is started by the **runmqdnm** command and stopped by the **endmqdnm** command. For details of these commands, see WebSphere MQ System Administration Guide.

To use the .NET Monitor, you write a component that implements the IMQObjectTrigger interface, which is defined in amqmdnm.dll.

Components can be either transactional or non-transactional. A transactional component must inherit from System.EnterpriseServices.ServicedComponent and be registered as either RequiresTransaction or SupportsTransaction. It must not be registered as RequiresNew as the .NET Monitor will already have initiated a transaction.

The component receives MQQueueManager, MQQueue, and MQMessage objects from **runmqdnm**. It may also receive a User Parameter string if one was specified, using the *–u* command line option, when **runmqdnm** was invoked. Note that your component receives the contents of a message that arrived on the monitored queue in an MQMessage object. It does not have to connect to the queue manager, open the queue or get the message itself. The component must then process the message as appropriate and return control to the .NET Monitor.

If your component has been written as a transactional component, it registers whether it wishes to commit or rollback the transaction using the facilities provided by System.EnterpriseServices.ServicedComponent.

As the component receives MQQueueManager and MQQueue objects as well as the message, it has complete context information for that message and can, for example, open another queue on the same queue manager without needing to separately connect to WebSphere MQ.

Example code fragments

This topic contains two examples of components which obtain a message from the .NET Monitor and print it, one using transactional processing and the other non-transactional processing. A third example shows common utility routines, applicable to both the first two examples. All the examples are in C#.

Example 1: Transactional processing

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005                                      */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
```

```
//
// run (with dotnet monitor)
//
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

Example 2: Non-transactional processing

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005                                      */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

Example 3: Common routines

```
/**********************************************************************/
/* Licensed materials, property of IBM                                */
/* 63H9336                                                            */
/* (C) Copyright IBM Corp. 2005                                       */
/**********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
 /// <summary>
 /// Summary description for Util.
 /// </summary>
 public class Util
 {
    /* ------------------------------------------------------------------ */
    /* Default prefix string of the namespace.                             */
    /* ------------------------------------------------------------------ */
    private string prefixText = "dnmsamp";

    /* ------------------------------------------------------------------ */
    /* Constructor that takes the replacement prefix string to use.       */
    /* ------------------------------------------------------------------ */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ------------------------------------------------------------------ */
    /* Display an arbitrary string to the console.                        */
    /* ------------------------------------------------------------------ */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
```

```
          {
            Print(ex.ToString());
          }
        }
        else
        {
          Print("UNRECOGNISED FORMAT");
        }
      }

      /* ------------------------------------------------------------------ */
      /* Convert the byte array into a hex string.                          */
      /* ------------------------------------------------------------------ */
      static public string ToHexString(byte[] byteArray)
      {
        string hex = "0123456789ABCDEF";

        string retString = "";

        for(int i = 0; i < byteArray.Length; i++)
        {
          int h = (byteArray[i] & 0xF0)>>4;
          int l = (byteArray[i] & 0x0F);

          retString += hex.Substring(h,1) + hex.Substring(l,1);
        }

        return retString;
      }

   }
  }
```

Compiling WebSphere MQ .NET programs

Specimen commands to compile .NET applications written in various languages.

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
```

Tracing WebSphere MQ .NET programs

In WebSphere MQ .NET, you start and control the trace facility as in WebSphere MQ programs using the MQI.

| However, the -i and -p parameters of the strmqtrc command, which allow you to
| specify process and thread identifiers, and named processes, have no effect.

You normally need to use the trace facility only at the request of IBM service.

See the *WebSphere MQ System Administration Guide* for information on trace commands.

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, interfaces, and structures are described:

MQAsyncStatus

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

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

This class encapsulates specific features of the MQSTS data structure. Objects of this class are used by applications inquiring on the status of previous MQI activity, for example inquiring on the success of previous asynchronous put operations.

Constructors

MQAsyncStatus

```
public MQAsyncStatus()
```

Throws MQException.

Constructor method, constructs an object with fields initialized to zero or blank as appropriate.

Properties

Properties for MQAsyncStatus

CompCode

```
public static int CompCode {get;}
```

The completion code from the first error or warning.

Reason

```
public static int Reason {get;}
```

The reason code from the first error or warning.

PutSuccessCount

```
public static int PutSuccessCount {get;}
```

The number of successful asynchronous MQI put calls.

PutWarningCount

```
public static int  PutWarningCount { get; }
```

The number of asynchronous MQI put calls that succeeded with a warning.

PutFailureCount

```
public static int PutFailureCount {get;}
```

The number of failed asynchronous MQI put calls.

ObjectType
```
public static int ObjectType {get;}
```

The object type for the first error. The following values are possible:
- MQC.MQOT_ALIAS_Q
- MQC.MQOT_LOCAL_Q
- MQC.MQOT_MODEL_Q
- MQC.MQOT_Q
- MQC.MQOT_REMOTE_Q
- MQC.MQOT_TOPIC
- zero, meaning that no object is returned

ObjectName
```
public static String ObjectName {get;}
```

The object name.

ObjectQMgrName
```
public static String  ObjectQMgrName {get;}
```

The object queue manager name.

ResolvedObjectName
```
public static String ResolvedObjectName {get;}
```

The resolved object name.

ResolvedObjectQMgrName
```
public static String ResolvedObjectQMgrName {get;}
```

The resolved object queue manager name.

For more detailed descriptions of these properties, see MQSTS Status reporting structure.

MQAuthenticationInformationRecord

The MQAuthenticationInformationRecord class encapsulates an authentication information record (MQAIR).

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

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

It allows an application running as a WebSphere MQ client to specify information about an authenticator that is to be used for the SSL client connection.

Constructors

Creates a new authentication information record.

MQAuthenticationInformationRecord
```
MQAuthenticationInformationRecord( );
```

Properties

Properties for MQAuthenticationInformationRecord

Version

```
public long Version {get; set;}
```

Structure version number.

AuthInfoType

```
public long AuthInfoType {get; set;}
```

The type of authentication information. The value must be CRLLDAP, meaning that Certificate Revocation List checking is done using LDAP servers.

AuthInfoConnName

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

The DNS name or IP address of the host on which the LDAP server is running, with an optional port number. This keyword is required.

LDAPPassword

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

The password associated with the Distinguished Name of the user who is accessing the LDAP server.

LDAPUserName

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

The Distinguished Name of the user who is accessing the LDAP server. When you set this property, LDAPUserNameLength and LDAPUserNamePtr are automatically set correctly.

| For more detailed descriptions of these properties, see Attributes for authentication
| information objects.

MQChannelDefinition

Use the MQChannelDefinition class to pass information concerning the connection to the queue manager to the send, receive, and security exits.

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

public class **MQChannelDefinition**
extends **Object**

Properties

Public variables in the MQChannelDefinition class.

ChannelName

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

The name of the channel through which the connection is established.

ClientChannelWeight

```
public String ClientChannelWeight
```

The client channel weight.

ConnectionAffinity

```
public String ConnectionAffinity
```

The connection affinity.

ConnectionName

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

The TCP/IP host name of the computer on which the queue manager resides.

MaxMessageLength

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

The maximum length of message that can be sent to the queue manager.

ReceiveExits

```
public String[] ReceiveExits {get; set;}
```

An array of the receive exit locations being used for the channel.

ReceiveUserDatas

```
public String[] ReceiveUserDatas {get; set;}
```

An array of the user data strings associated with each receive exit for the channel.

SecurityExit

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

The security exit location being used for the channel.

SecurityUserData

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

A storage area for the security exit to use. Information placed here is preserved across invocations of the security exit, and is also available to the send and receive exits.

SendExits

```
public String[] SendExits {get; set;}
```

An array of the send exit locations being used for the channel.

SendUserDatas

```
public String[] SendUserDatas {get; set;}
```

An array of the user data strings associated with each send exit for the channel.

SharingConversations

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

Number of sharing conversations for this channel instance.

SSLCipherSpec

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

The SSL Cipher Specification defined for the channel.

SSLPeerName

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

If SSL is used to encrypt data on the wire, this is set to the Distinguished Name presented by the queue manager during connection. If SSL is not used, it is left as null.

For more detailed descriptions of these properties, see MQCD - Channel definition.

MQChannelExit

This class defines context information passed to the send, receive, and security exits when they are invoked. The exit must set the ExitResponse member variable to indicate what action the WebSphere MQ Client for .NET should take next.

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

public class **MQChannelExit**
extends **Object**

Note: This class does not apply when connecting directly to WebSphere MQ in bindings mode.

Properties of MQChannelExit

The properties of MQChannelExit are described.

CapabilityFlags

`public int CapabilityFlags {get; set;}`

Indicates the capability of the queue manager.

Only the MQC.MQCF_DIST_LISTS flag is supported.

CurHdrCompression

`public int CurHdrCompression {get; set;}`

The type of compression currently being employed on this channel for message header compression.

CurMsgCompression

`public int CurMsgCompression {get; set;}`

The type of compression currently being employed on this channel for message data compression.

ExitID

`public int ExitID {get; set;}`

The type of exit that has been invoked. For an MQSecurityExit this is always MQC.MQXT_CHANNEL_SEC_EXIT; for an MQSendExit this is always MQC.MQXT_CHANNEL_SEND_EXIT; for an MQReceiveExit this is always MQC.MQXT_CHANNEL_RCV_EXIT.

ExitNumber

`public int ExitNumber {get; set;}`

A zero based index indicating the index of this exit in the array of exits of the same type. For example, a value of 1 indicates that this is the second instance of a send exit.

ExitReason

`public int ExitReason {get; set;}`

The reason for invoking the exit. Possible values are:

MQC.MQXR_INIT
: Exit initialization; called after the channel connection conditions have been negotiated, but before any security flows have been sent.

MQC.MQXR_INIT_SEC
: Indicates that the exit is to initiate the security dialog with the queue manager.

MQC.MQXR_SEC_MSG
: Indicates to the security exit that a security message has been received from the queue manager.

MQC.MQXR_TERM
: Exit termination; called after the disconnect flows have been sent but before the socket connection is destroyed.

MQC.MQXR_XMIT
: For a send exit, indicates that data is to be transmitted to the queue manager.

  For a receive exit, indicates that data has been received from the queue manager.

ExitResponse

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

Set by the exit to indicate the action that WebSphere MQ classes for .NET must take next. Valid values are:

MQC.MQXCC_CLOSE_CHANNEL
: Set by any exit to indicate that the connection to the queue manager must be closed.

MQC.MQXCC_OK
: Set by the security exit to indicate that security exchanges are complete.

  Set by send exit to indicate that the returned data is to be transmitted to the queue manager.

  Set by the receive exit to indicate that the returned data is available for processing by the WebSphere MQ Client for .NET.

MQC.MQXCC_SEND_AND_REQUEST_SEC_MSG
: Set by the security exit to indicate that the returned data is to be transmitted to the queue manager, and that a response is expected from the queue manager.

MQC.MQXCC_SEND_SEC_MSG
: Set by the security exit to indicate that the returned data is to be transmitted to the queue manager, and that no response is expected.

MQC.MQXCC_SUPPRESS_EXIT
: Set by any exit to indicate that it must no longer be called.

MQC.MQXCC_SUPPRESS_FUNCTION
: Set by the security exit to indicate that communications with the queue manager must be shut down.

ExitUserArea

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

A storage area available for the exit to use.

Any data placed in the exitUserArea is preserved by the WebSphere MQ Client for .NET across exit invocations with the same exitID. (That is, the send, receive, and security exits each have their own, independent, user areas.)

FapLevel

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

The negotiated Format and Protocol (FAP) level.

Hconn

```
public MQHCONN HConn {get; set;}
```

Connection handle for the exit to use when making MQI calls.

MaxSegmentLength

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

The maximum length for any one transmission to a queue manager.

If the exit returns data that is to be sent to the queue manager, the length of the returned data must not exceed this value.

SharingConversations

```
public MQBOOL SharingConversations {get; set;}
```

Whether the conversation is sharing this channel instance.

UserData

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

The user data parameter specified on the channel definition for this specific exit instance.

For more detailed descriptions of these properties, see Fields.

MQDestination

MQDestination object for .NET

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

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

MQDestination is an abstract base class and so cannot be instantiated by itself. It is designed to contain the common functionality for any WebSphere MQ messaging destination. MQDestination is a super class for both MQQueue and MQTopic.

Constructors

Constructors for MQDestination.

MQDestination

```
protected MQDestination();
```

Default constructor. MQDestination is an abstract base class and cannot be instantiated by itself.

Methods

Methods for MQDestination object.

Put

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

Throws MQException.

Places a message onto a queue or publishes a message to a topic. This method uses a default instance of MQPutMessageOptions to perform the put or publish. The default MQPutMessageOptions instance differs depending upon the destination type.

Parameters

message
: An MQMessage object containing the Message Descriptor data (MQMD) and message to be sent. The Message Descriptor properties of this object can be altered as a consequence of this method. The values that they have immediately after the completion of this method are the values that were put to the queue or published to the topic.

Put

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

Throws MQException.

Places a message onto a queue or publishes a message to a topic.

Parameters

message
: An MQMessage object containing the Message Descriptor data (MQMD) and message to be sent. The Message Descriptor properties of this object can be altered as a consequence of this method. The values that they have immediately after the completion of this method are the values that were put to the queue or published to the topic.

putMessageOptions
: Options controlling the action of the put. See MQPutMessageOptions object “Properties” on page 82.

Get

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

Throws MQException.

Retrieves a message from the queue or topic. 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.

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

Parameters

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

Get

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

Throws MQException.

Retrieves a message from the queue or topic. 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.

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 object "Properties" on page 40.

Get

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

Throws MQException.

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

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 object “Properties” on page 40.

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

  - If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code.
  - If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue or topic and an exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

Properties

Properties for MQDestination.

CreationDateTime

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

The date and time that the queue or topic was created. Originally contained within MQQueue, this property has been moved into the base MQDestination class.

There is no default value.

DestinationType

```
public int DestinationType { get; }
```

Integer value describing the type of destination being used. Initialized from the sub classes constructor (MQQueue or MQTopic), this value can take one of these values:

- MQOT_Q
- MQOT_TOPIC

There is no default value.

MQEnvironment

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

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

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

Constructors

MQEnvironment

```
public MQEnvironment()
```

Properties

Properties of the MQEnvironment class.

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.

FipsRequired

```
public static MQLONG FipsRequired {get; set;}
```

Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere MQ. If cryptographic hardware is configured, the cryptographic modules used are those provided by the hardware product, and these might, or might not, be FIPS-certified to a particular level. This depends on the hardware product in use.

There are two constants available to use when setting this value:

- MQC.MQSSL_FIPS_NO - this equates to the numeric value 0
- MQC.MQSSL_FIPS_YES - this equates to the numeric value 1

HdrCompList

```
public static ArrayList HdrCompList {get; set;}
```

Header Data Compression List

Hostname*

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

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

KeyResetCount

```
public static MQLONG KeyResetCount {get; set;}
```

Indicates the number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.

MessageExit

```
public static String MsgExit {get; set;}
```

A message exit allows you to send the application data in a particular content and format. If MessageExit is set to null, no message exit will be called.

MQAIRArray

```
public static ArrayList MQAIRArray {get; set;}
```

An array of authentication information records.

MsgCompList

```
public static ArrayList MsgCompList {get; set;}
```

Message Data Compression List

Password

```
public static String Password {get; set;}
```

The password to be authenticated.

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.

ReceiveExit

```
public static String ReceiveExit {get; set;}
```

A receive exit allows you to examine and alter data received from a queue manager. It is normally used in conjunction with a corresponding send exit at the queue manager. If ReceiveExit is set to null, no receive exit will be called.

ReceiveUserData

```
public static String ReceiveUserData {get; set;}
```

The user data associated with a receive exit. Limited to 32 characters.

SecurityExit

```
public static String SecurityExit {get; set;}
```

A security exit allows you to customize the security flows that occur when an attempt is made to connect to a queue manager. If securityExit is set to null, no security exit will be called.

SecurityUserData

```
public static String SecurityUserData {get; set;}
```

The user data associated with a security exit. Limited to 32 characters.

SendExit

```
public static String SendExit {get; set;}
```

A send exit allows you to examine alter the data sent to a queue manager. It is normally used in conjunction with a corresponding receive exit at the queue manager. If SendExit is set to null, no send exit will be called.

SendUserData

```
public static String SendUserData {get; set;}
```

The user data associated with a send exit. Limited to 32 characters.

SharingConversations

```
public static String SharingConversations {get; set;}
```

The *SharingConversations* field is used on connections from .NET applications, when these applications are not using a client channel definition table.

SharingConversations determines the maximum number of conversations that can be shared on a socket associated with this connection.

A value of 0 means that the channel operates as it did before WebSphere MQ Version 7.0, with regard to conversation sharing, read ahead, and heartbeat.

The field is passed in the hashtable of properties as a SHARING_CONVERSATIONS_PROPERTY, when instantiating a WebSphere MQ queue manager.

If you do not specify *SharingConversations*, a default value of 10 is used.

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.

SSLCryptoHardware

```
public static String SSLCryptoHardware {get; set;}
```

Sets the name of the parameter string required to configure the cryptographic hardware present on the system. For a full description of this property, see WebSphere MQ Programmable Command Formats and Administration Interface. This variable is ignored if sslCipherSpec is null.

SSLKeyRepository

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

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

If this parameter is 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 . kdb extension is a mandatory part of the file name, 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 22. If set to null (default), no checking of the queue manager's DN is performed. This variable is ignored if sslCipherSpec is null.

UserId

```
public static String UserId {get; set;}
```

The UserId to be authenticated. The Userid field in the MQCSP structure gets populated by setting this Userid property. Authentication of this Userid peroperty can be performed using an API or Security exit.

| For more detailed descriptions of these properties, see MQSCO - SSL configuration
| options, MQAIR - Authentication information record, Fields.

MQException

An MQException is thrown whenever a WebSphere MQ error occurs.

```
System.Object
 │
 └ System.Exception
    │
    └ System.ApplicationException
       │
       └ IBM.WMQ.MQException
```

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

Constructors

Construct a new MQException object.

MQException

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

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

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

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

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

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

Properties for MQGetMessageOptions.

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 (*).

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 when the data is retrieved from the message buffer, applications do not typically 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 typically 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 typically 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.

Message properties options These options relate to the handling of message properties.

MQGMO_PROPERTIES_AS_Q_DEF
: Properties of the message, except those contained in the message descriptor (or extension), are represented as defined by the PropertyControl attribute of MQQueue.

MQGMO_PROPERTIES_IN_HANDLE
: Properties of the message are made available via the MsgHandle.

MQGMO_NO_PROPERTIES
: No properties of the message, except those contained in the message descriptor (or extension) are retrieved.

MQGMO_PROPERTIES_COMPATIBILITY
: If the message contains a property with a prefix of "mcd.", "jms.", "usr." or "mqext." then all message properties, except those contained in the message descriptor (or extension) should be represented using MQRFH2 headers. Otherwise no properties of the message, except those contained in the message descriptor (or extension) will be retrieved.

MQGMO_PROPERTIES_FORCE_MQRFH2
: Properties of the message, except those contained in the message descriptor (or extension) are represented using MQRFH2 headers. This provides backward compatibility for applications which are expecting to retrieve properties but cannot be changed to use message handles.

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 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.

| For more detailed descriptions of these properties, see MQGMO Get-message
| options.

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 MQDestination, MQProcess,
| MQQueueManager, and MQSubscription. It provides the ability to inquire and set
| attributes of these resources.

Constructors

MQManagedObject

```
protected MQManagedObject()
```

Constructor method. This object is an abstract base class which cannot be instantiated by itself.

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.

GetAttributeString

```
public String GetAttributeString(int selector,
                                 int length)
```

Throws MQException.

Gets an attribute string.

Throws MQException.

Parameters

length Integer indicating the length of the string required.

selector Integer indicating which attribute is being queried. Suitable selectors for character attributes are shown in MQCA_*.

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.

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.

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.

SetAttributeString

```
public void SetAttributeString(int selector,
                               String value,
                               int length);
```

Throws MQException.

Sets an attribute string.

Throws MQException.

Parameters

selector Integer indicating which attribute is being set. Suitable selectors for character attributes are shown in MQCA_*

value The string to set as the attribute value.

length Integer indicating the length of the string required.

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. This property is not valid for subscriptions and is ignored.

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, temporary dynamic queues, subscriptions and topics that are being accessed by the objects that created them.

For queues and topics 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.

MQC.MQCO_QUIESCE
Request the queue be closed, receiving a warning if any messages remain (allowing them to be retrieved before final closing).

For subscriptions the following additional values are permissible:

MQC.MQCO_KEEP_SUB
The subscription is not deleted. This option is valid only if the original subscription is durable. This is the default value if the resource is a durable topic.

MQC.MQCO_REMOVE_SUB
The subscription is deleted. This is the default value if the resource is a non-durable, unmanaged topic.

MQC.MQCO_PURGE_SUB
The subscription is deleted. This is the default value if the resource is a non-durable, managed topic.

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

The queue manager to which this resource belongs.

Description
```
public  String MQDescription {get;}
```

The description of the resource as held by the queue manager. This property will return an empty string for subscriptions and topics.

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. This property is not valid for subscriptions.

MQMessage

MQMessage represents both the message descriptor and the data for a WebSphere MQ message.

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

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

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 for MQMessage

ClearMessage

```
public void ClearMessage()
```

Throws IOException.

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

DeleteProperty

```
public void DeleteProperty(String name)
```

Throws MQException.

Deletes a property with the specified name from the message.

Parameters

name The name of the property to delete.

GetBooleanProperty

```
public boolean GetBooleanProperty(String name)
```

Throws MQException.

Returns the value of the boolean property with the specified name.

Parameters

name The name of the boolean property.

GetBooleanProperty

```
public boolean GetBooleanProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the boolean property with the specified name, completing the specified property descriptor.

Parameters

name The name of the boolean property.

pd The attributes of the property.

GetByteProperty

```
public sbyte GetByteProperty(String name)
```

Throws MQException.

Returns the value of the byte property with the specified name.

Parameters

name The name of the byte property.

GetByteProperty

```
public sbyte GetByteProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the byte property with the specified name, completing the specified property descriptor.

Parameters

name The name of the byte property.

pd The attributes of the property.

GetBytesProperty

```
public sbyte[] GetBytesProperty(String name)
```

Throws MQException.

Returns the value of the signed byte array property with the specified name.

Parameters

name The name of the byte array property.

GetBytesProperty

```
public sbyte[] GetBytesProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the signed byte array property with the specified name, completing the specified property descriptor.

Parameters

name The name of the byte array property.

pd The attributes of the property.

GetDoubleProperty

```
public double GetDoubleProperty(String name)
```

Throws MQException.

Returns the value of the double property with the specified name.

Parameters

name The name of the double property.

GetDoubleProperty

```
public double GetDoubleProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the double property with the specified name, completing the specified property descriptor.

Parameters

name The name of the double property.

pd The attributes of the property.

GetFloatProperty

```
public float GetFloatProperty(String name)
```

Throws MQException.

Returns the value of the float property with the specified name.

Parameters

name The name of the float property.

GetFloatProperty

```
public float GetFloatProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the float property with the specified name, completing the specified property descriptor.

Parameters

name The name of the float property.

pd The attributes of the property.

GetInt2Property

```
public short GetInt2Property(String name)
```

Throws MQException.

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

Parameters

name The name of the short property.

GetInt2Property

```
public short GetInt2Property(String name, MQPropertyDescriptor pd)
```

Throws MQException.

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

Parameters

name The name of the short property.

pd The attributes of the property.

GetInt4Property

```
public int GetInt4Property(String name)
```

Throws MQException.

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

Parameters

name The name of the int property.

GetInt4Property

```
public int GetInt4Property(String name, MQPropertyDescriptor pd)
```

Throws MQException.

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

Parameters

name The name of the int property.

pd The attributes of the property.

GetInt8Property

```
public long GetInt8Property(String name)
```

Throws MQException.

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

Parameters

name The name of the long property.

GetInt8Property

```
public long GetInt8Property(String name, MQPropertyDescriptor pd)
```

Throws MQException.

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

Parameters

name The name of the long property.

pd The attributes of the property.

GetLongProperty

```
public long GetLongProperty(String name)
```

Throws MQException.

Returns the value of the long property with the specified name.

Parameters

name The name of the long property.

GetLongProperty

```
public long GetLongProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the long property with the specified name, completing the specified property descriptor.

Parameters

name The name of the long property.

pd The attributes of the property.

GetObjectProperty

```
public Object GetObjectProperty(String name)
```

Throws MQException.

Returns the value of the .NET object property with the specified name.

You can use this method to return, in objectified format, an object that has been stored as a property in the message with the SetObjectProperty method call, or its equivalent primitive Set*type*Property method.

Parameters

name The name of the String property.

GetObjectProperty

```
public Object GetObjectProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the .NET object property with the specified name, completing the specified property descriptor.

You can use this method to return, in objectified format, an object that has been stored as a property in the message with the SetObjectProperty method call, or its equivalent primitive Set*type*Property method.

Parameters

name The name of the String property.

pd The attributes of the property.

GetPropertyNames

```
public System.Collectoins.IEnumerator GetPropertyNames(String name)
```

Throws MQException.

Returns an IEnumerator of all the property names matching the specified name. The percent sign (%) can be used at the end of the name as a wildcard character to filter the properties of the message, matching on zero or more characters, including the period (.).

Parameters

name The name of the property to match on.

GetShortProperty

```
public short GetShortProperty(String name)
```

Throws MQException.

Returns the value of the short property with the specified name.

Parameters

name The name of the short property.

GetShortProperty

```
public short GetShortProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the short property with the specified name, completing the specified property descriptor.

Parameters

name The name of the short property.

pd The attributes of the property.

GetStringProperty

```
public String GetFloatProperty(String name)
```

Throws MQException.

Returns the value of the String property with the specified name.

Parameters

name The name of the String property.

GetStringProperty

```
public String GetFloatProperty(String name, MQPropertyDescriptor pd)
```

Throws MQException.

Returns the value of the String property with the specified name, completing the specified property descriptor.

Parameters

name The name of the String property.

pd The attributes of the property.

PutForwardMessage

```
public void PutForwardMessage(MQMessage message)
```

Throws MQException.

Puts a message to be forwarded on the queue using a default instance of MQPutMessageOptions, with *message* containing the original message.

Parameters

message
The message to be forwarded

PutForwardMessage

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

Throws MQException.

Puts a message to be forwarded on the queue with *message* containing the original message.

Parameters

message
The message to be forwarded

putMessageOptions
Options controlling the action of the put. For more information, see "MQPutMessageOptions" on page 81.

PutReplyMessage

```
public void PutReplyMessage(MQMessage message)
```

Throws MQException.

Puts a reply message on the queue using a default instance of MQPutMessageOptions, with *message* containing the original message.

Parameters

message
: The request message to be replied to

PutReplyMessage

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

Throws MQException.

Puts a reply message on the queue with *message* containing the original message.

Parameters

message
: The request message to be replied to

putMessageOptions
: Options controlling the action of the put. For more information, see "MQPutMessageOptions" on page 81.

PutReportMessage

```
public void PutReportMessage(MQMessage message)
```

Throws MQException.

Puts a report message on the queue using a default instance of MQPutMessageOptions, with *message* containing the original message.

Parameters

message
: The message causing the report to be generated

PutReportMessage

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

Throws MQException.

Puts a report message on the queue with *message* containing the original message.

Parameters

message
: The message causing the report to be generated

putMessageOptions
: Options controlling the action of the put. For more information, see "MQPutMessageOptions" on page 81.

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 code set 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 code set identified by the characterSet member variable, and convert it into Unicode.

Parameters:

length	The number of characters to read (which can differ from the number of bytes according to the code set, because some code sets 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.

SetBooleanProperty

```
public void SetBooleanProperty(String name, boolean value)
```

Throws MQException.

Sets a boolean property value with the specified name into the message, with the default property descriptor.

Parameters

name	The name of the boolean property.
value	The boolean property value to set.

SetBooleanProperty

```
public void SetBooleanProperty(String name, MQPropertyDescriptor pd, boolean value)
```

Throws MQException.

Sets a boolean property value with the specified name into the message, with the specified property descriptor.

Parameters

name	The name of the boolean property.
pd	The attributes of the property.
value	The boolean property value to set.

SetByteProperty

```
public void SetByteProperty(String name, sbyte value)
```

Throws MQException.

Sets a signed byte property value with the specified name into the message, with the default property descriptor.

Parameters

name	The name of the byte property.
value	The byte property value to set.

SetByteProperty

```
public void SetByteProperty(String name, MQPropertyDescriptor pd, sbyte value)
```

Throws MQException.

Sets a signed byte property value with the specified name into the message, with the specified property descriptor.

Parameters

name	The name of the byte property.
pd	The attributes of the property.
value	The byte property value to set.

SetBytesProperty
```
public void SetBytesProperty(String name, sbyte[] value)
```

Throws MQException.

Sets a signed byte array property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the byte array property.

value The byte array property value to set.

SetBytesProperty
```
public void SetBytesProperty(String name, MQPropertyDescriptor pd, sbyte[] value)
```

Throws MQException.

Sets a signed byte array property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the byte array property.

pd The attributes of the property.

value The byte array property value to set.

SetDoubleProperty
```
public void SetDoubleProperty(String name, double value)
```

Throws MQException.

Sets a double property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the double property.

value The double property value to set.

SetDoubleProperty
```
public void SetDoubleProperty(String name, MQPropertyDescriptor pd, double value)
```

Throws MQException.

Sets a double property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the double property.

pd The attributes of the property.

value The double property value to set.

SetFloatProperty
```
public void SetFloatProperty(String name, float value)
```

Throws MQException.

Sets a float property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the float property.

value The float property value to set.

SetFloatProperty

```
public void SetFloatProperty(String name, MQPropertyDescriptor pd, float value)
```

Throws MQException.

Sets a float property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the float property.

pd The attributes of the property.

value The float property value to set.

SetIntProperty

```
public void SetIntProperty(String name, int value)
```

Throws MQException.

Sets an int property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the int property.

value The int property value to set.

SetIntProperty

```
public void SetIntProperty(String name, MQPropertyDescriptor pd, int value)
```

Throws MQException.

Sets an int property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the int property.

pd The attributes of the property.

value The int property value to set.

SetInt2Property

```
public void SetInt2Property(String name, short value)
```

Throws MQException.

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

Parameters

name The name of the short property.

value The short property value to set.

SetInt2Property

```
public void SetInt2Property(String name, MQPropertyDescriptor pd, short value)
```

Throws MQException.

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

Parameters

name The name of the short property.

pd The attributes of the property.

value The short property value to set.

SetInt4Property

```
public void SetInt4Property(String name, int value)
```

Throws MQException.

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

Parameters

name The name of the int property.

value The int property value to set.

SetInt4Property

```
public void SetInt4Property(String name, MQPropertyDescriptor pd, int value)
```

Throws MQException.

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

Parameters

name The name of the int property.

pd The attributes of the property.

value The int property value to set.

SetInt8Property

```
public void SetInt8Property(String name, long value)
```

Throws MQException.

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

Parameters

name The name of the long property.

value The long property value to set.

SetInt8Property

```
public void SetInt8Property(String name, MQPropertyDescriptor pd,long value)
```

Throws MQException.

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

Parameters

name The name of the long property.

pd The attributes of the property.

value The long property value to set.

SetLongProperty

```
public void SetLongProperty(String name, long value)
```

Throws MQException.

Sets a long property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the long property.

value The long property value to set.

SetLongProperty

```
public void SetLongProperty(String name, MQPropertyDescriptor pd, long value)
```

Throws MQException.

Sets a long property value with The specified name into the message, with the specified property descriptor.

Parameters

name The name of the long property.

pd The attributes of the property.

value The long property value to set.

SetObjectProperty

```
public void SetObjectProperty(String name, Object value)
```

Throws MQException.

Sets a .NET object property value with the specified name into the message, with the default property descriptor.

This method works only for the objectified primitive object types (Integer, Double, Long, and so on) and String objects. The property is set as if the Set*Property method had been called directly, for example, an Integer object leads to an int property value being set, and a Long object leads to a long property value being set.

Parameters

name The name of the .NET object property.

value The .NET object property value to set

SetObjectProperty

```
public void SetObjectProperty(String name, MQPropertyDescriptor pd, Object value)
```

Throws MQException.

Sets a .NET object property value with the specified name into the message, with the specified property descriptor.

This method works only for the objectified primitive object types (Integer, Double, Long, and so on) and String objects. The property is set as if the Set*Property method had been called directly, for example, an Integer object leads to an int property value being set, and a Long object leads to a long property value being set.

Parameters

name The name of the .NET object property.

pd The attributes of the property.

value The .NET object property value to set.

SetShortProperty

```
public void SetShortProperty(String name, short value)
```

Throws MQException.

Sets a short property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the short property.

value The short property value to set.

SetShortProperty

```
public void SetShortProperty(String name, MQPropertyDescriptor pd, short value)
```

Throws MQException.

Sets a short property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the short property.

pd The attributes of the property.

value The short property value to set.

SetStringProperty

```
public void SetBytesProperty(String name, String value)
```

Throws MQException.

Sets a String property value with the specified name into the message, with the default property descriptor.

Parameters

name The name of the string property.

value The string property value to set.

SetStringProperty

```
public void SetBytesProperty(String name, MQPropertyDescriptor pd, String value)
```

Throws MQException.

Sets a String property value with the specified name into the message, with the specified property descriptor.

Parameters

name The name of the string property.

pd The attributes of the property.

value The string property value to set.

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, therefore 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 code set 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

Properties for MQMessage.

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
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

Table 1. Character set identifiers (continued)

characterSet	Description
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_REVERSED | MQC.MQENC_DECIMAL_REVERSED | MQC.MQENC_FLOAT_IEEE_REVERSED For convenience, this value is also represented by MQC.MQENC_NATIVE. This setting causes WriteInt() to write a little-endian integer, and ReadInt() to read a little-endian integer. If you set the flag MQC.MQENC_INTEGER_NORMAL flag instead, WriteInt() writes a big-endian integer, and ReadInt() reads a big-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 field 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; set;}`

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

OriginalLength

`public int OriginalLength {get; set;}`

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.

PropertyValidation

`public int PropertyValidation {get; set;}`

Specifies whether validation of properties will take place when a property of the message is set. Possible values are:

- MQCMHO_DEFAULT_VALIDATION
- MQCMHO_VALIDATE
- MQCMHO_NO_VALIDATION

The default value is MQCMHO_DEFAULT_VALIDATION.

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; set;}`

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 will 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 will 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 can 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.

- MQC.MQRO_PAN
- MQC.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.

For more detailed descriptions of these properties, see MQMD - message descriptor fields.

MQProcess

MQProcess object for .NET

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

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

MQProcess provides inquire operations for WebSphere MQ processes. Use either the corresponding MQProcess constructors or the MQQueueManager::AccessProcess (...) methods to create an MQProcess object.

Constructors for MQProcess

Constructors for MQProcess.

MQProcess

```
public MQProcess(MQQueueManager qMgr, String processName, int openOptions)
```

Throws MQException.

Establishes access to a WebSphere MQ process on the queue manager `qMgr` such that the process attributes can be inquired. The default user authority is used for connection to the queue manager.

See MQQueueManager.AccessProcess for details of the remaining parameters.

MQProcess

```
public MQProcess(MQQueueManager qMgr, String processName, int openOptions,
                 String queueManagerName, String alternateUserId)
```

Throws MQException.

Establishes access to a WebSphere MQ process on the queue manager `qMgr` such that the process attributes can be inquired. The specified alternative user authority is used for connection to the queue manager.

See MQQueueManager.AccessProcess for details of the remaining parameters.

Properties

Properties for MQProcess.

ApplicationId

```
public String ApplicationId { get; }
```

Gets the character string that identifies the application to be started. This information is used by the trigger monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The default value is null.

ApplicationType

```
public int ApplicationType { get; }
```

Identifies the nature of the process to be started in response to a trigger message. The following standard types have already been defined but others can be used:

- MQAT_AIX
- MQAT_CICS
- MQAT_IMS
- MQAT_MVS
- MQAT_NATIVE
- MQAT_OS400
- MQAT_UNIX
- MQAT_WINDOWS
- MQAT_JAVA
- MQAT_USER_FIRST
- MQAT_USER_LAST

The default value is MQAT_NATIVE.

EnvironmentData

```
public String EnvironmentData { get; }
```

Gets information on the environment of the application that is to be started.

The default value is null.

UserData

```
public String UserData { get; }
```

Gets information pertaining to the application to be started.

The default value is null.

MQPropertyDescriptor

This class encapsulates a property descriptor structure (MQPD). An MQPD instance describes an MQMessage property.

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

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

This class is an input parameter on the MQMessage.set*Property() calls and an output parameter on the MQMessage.get*Property() calls.

Constructors

Constructors for the property descriptor (MQPD).

ImqPropertyDescriptor( );
Create a new property descriptor.

Properties

Properties for MQPropertyDescriptor

Context

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

The message context the property belongs to. Possible values are:

CMQC.MQPD_NO_CONTEXT
The property is not associated with a message context.

CMQC.MQPD_USER_CONTEXT
The property is associated with the user context.

A property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT call with MQPMO_PASS_ALL_CONTEXT specified, causes the property to be copied from the saved context into the new message.

CopyOptions

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

This describes which type of message the property should be copied into.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect. the queue manager corrects the value of the CopyOptions field.

Any of the following can be specified. If more than one is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

You can specifiy one or more of these options:

CMQC.MQCOPY_ALL
This property is copied into all types of subsequent messages.

CMQC.MQCOPY_FORWARD
This property iscopied into a message being forwarded.

CMQC.MQCOPY_PUBLISH
This property is copied into the message received by a subscriber when a message is being published.

CMQC.MQCOPY_REPLY
This property is copied into a reply message.

CMQC.MQCOPY_REPORT
This property is copied into a report message.

CMQC.MQCOPY_DEFAULT
Use this value to indicate that no other copy options have been specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

CMQC.MQCOPY_NONE
Use this value to indicate that no other copy options have been specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

Options

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

Message property's options. This is always an input field. The default value is CMQC.MQPD_NONE

Support

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

This field describes what level of support for the message property is required of the queue manager in order for the message containing this property to be put to a queue. This only applies to WebSphere MQ-defined properties; support for all other properties is optional. Any or none of the following values can be specified

CMQC.MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported. The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not WebSphere MQ-defined.

CMQC.MQPD_SUPPORT_REQUIRED
Support for the property is required. The message is rejected by a queue manager that does not support the WebSphere MQ-defined property. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

CMQC.MQPD_SUPPORT_REQUIRED_IF_LOCAL
The message is rejected by a queue manager that does not support the WebSphere MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

Version
This is the structure version number; the initial value is MQPD_VERSION_1.

MQPD_VERSION_1
Version-1 property descriptor structure.

MQPD_CURRENT_VERSION
Current version of property descriptor structure.

MQPutMessageOptions

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

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

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

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

MQPutMessageOptions

```
public MQPutMessageOptions()
```

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

Properties

Properties of MQPutMessageOptions.

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 and 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_ASYNC_RESPONSE
This option causes the MQPUT or MQPUT1 call to be made asynchronously, with some response data.

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_RESPONSE_AS_Q_DEF

For an MQPUT call, this option takes the put response type from DEFPRESP attribute of the queue.

For an MQPUT1 call, this option causes the call to be made synchronously.

| **MQC.MQPMO_RESPONSE_AS_TOPIC_DEF**
| This is a synonym for MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

| **MQC.MQPMO_RETAIN**
| The publication being sent is to be retained by the queue manager. This allows a subscriber to request a copy of this publication after the time it was published, by using the MQSUBRQ call. It also allows a publication to be sent to applications which make their subscription after the time this publication was made (unless they choose not to be sent it by using the option MQSO_NEW_PUBLICATIONS_ONLY). If an application is sent a publication which was retained, this will be indicated by the MQIsRetained message property of that publication.

| Only one publication can be retained at each node of the topic tree. That means if there already is a retained publication for this topic, published by any other application, it is replaced with this publication. It is recommended that you do not have more than one publisher retaining messages on the same topic.

| When retained publications are requested by a subscriber, the subscription used may contain a wildcard in the topic, in which case a number of retained publications may match (at various nodes in the topic tree) and several publications may be sent to the requesting application. See MQSUBRQ - Subscription Request for more details.If this option is used and the publication cannot be retained, the message will not be published and the call will fail with MQRC_PUT_NOT_RETAINED.

MQC.MQPMO_SET_ALL_CONTEXT
: Set all context from the application.

MQC.MQPMO_SET_IDENTITY_CONTEXT
: Set identity context from the application.

MQC.MQPMO_SYNC_RESPONSE
This option causes the MQPUT or MQPUT1 call to be made synchronously, with full response data.

MQC.MQPMO_SUPPRESS_REPLYTO
Any information filled into the ReplyToQ and ReplyToQMgr fields of the MQMD of this publication will not be passed on to subscribers. If this option is used in combination with a report option that requires a ReplyToQ, the call will fail with MQRC_MISSING_REPLY_TO_Q.

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.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

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.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

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.

For more detailed descriptions of these properties, see MQPMO Put-message options.

MQQueue

MQQueue object for .NET

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

public class **IBM.WMQ.MQQueue**
extends **IBM.WMQ.MQDestination**. (See "MQDestination" on page 33.)

In WebSphere MQ V7.0 MQQueue has been modified to be a sub class of MQDestination (it was previously a sub class of MQManagedObject). Some of the methods and properties originally available on the MQQueue object have been moved into the parent class (MQDestination). This does not affect any existing WebSphere MQ .NET applications.

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

See also MQQueueManager.AccessQueue.

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 for details of the remaining parameters.

Methods

Close

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

Overrides MQManagedObject.Close.

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 40.)

  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 40 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 the MQQueueManager object. For this you do not need to have an MQQueue object. See MQQueueManager.Put.

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 81)

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.

PutForwardMessage

```
public void PutForwardMessage(MQMessage message)
```

Put a message being forwarded onto the queue using default put message options and message as the original message.

Parameters

MQMessage
: The message for forwarding.

Throws MQException if the put fails.

PutForwardMessage

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

Throws MQException.

Put a message being forwarded onto the queue using message as the original message.

Parameters

MQMessage
The message for forwarding.

MQPutMessageOptions
Options controlling the action of the put. (See "MQPutMessageOptions" on page 81)

Throws MQException if the put fails.

PutReplyMessage

```
public void PutReplyMessage(MQMessage message)
```

Put a reply message onto the queue using default put message options and message as the original message.

Parameters

MQMessage
The request message to be replied to.

Throws MQException if the put fails.

PutReplyMessage

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

Throws MQException.

Put a reply message onto the queue using message as the original message.

Parameters

MQMessage
The request message to be replied to.

MQPutMessageOptions
Options controlling the action of the put. (See "MQPutMessageOptions" on page 81)

Throws MQException if the put fails.

PutReportMessage

```
public void PutReportMessage(MQMessage message)
```

Put a report message onto the queue using default put message options and message as the original message.

Parameters

MQMessage
The message that caused the report to be generated.

Throws MQException if the put fails.

PutReportMessage

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

Throws MQException.

Put a message being forwarded onto the queue using message as the original message.

Parameters

MQMessage
: The message that caused the report to be generated.

MQPutMessageOptions
: Options controlling the action of the put. (See "MQPutMessageOptions" on page 81)

Throws MQException if the put fails.

Properties

Properties of MQQueue.

ClusterWorkLoadPriority

```
public int ClusterWorkLoadPriority {get;}
```

Specifies the priority of the queue. This parameter is valid only for local, remote, and alias queues.

ClusterWorkLoadRank

```
public int ClusterWorkLoadRank {get;}
```

Specifies the rank of the queue. This parameter is valid only for local, remote, and alias queues.

ClusterWorkLoadUseQ

```
public int ClusterWorkLoadUseQ {get;}
```

Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. This parameter does not apply if the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

CreationDateTime

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

Throws MQException.

The date and time that this queue was created.

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.

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

InhibitPut

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

Throws MQException.

Controls whether put operations are allowed for this queue or topic. The possible values are:
- MQQA_PUT_INHIBITED
- 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.

NonPersistentMessageClass

```
public int NonPersistentMessageClass {get;}
```

The level of reliability for non persistent messages put to this queue.

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).

QueueAccounting

```
public int QueueAccounting {get;}
```

Specifies whether the collection of accounting information is enabled for the queue.

QueueMonitoring

```
public int QueueMonitoring {get;}
```

Specifies whether monitoring is enabled for the queue.

QueueStatistics

```
public int QueueStatistics {get;}
```

Specifies whether collection of statistics is enabled for the queue.

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

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

TPIPE

```
public string TPIPE {get;}
```

The TPIPE name used for communication with OTMA via the WebSphere MQ IMS™ bridge.

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 must 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 must 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 will 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

For more detailed descriptions of these properties, see Attribute descriptions for queues.

MQQueueManager

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

```
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 44.)

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 36.

MQQueueManager

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

Throws MQException.

This version of the constructor is intended for use only in bindings mode, that is, when connecting to a local server. 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,
                      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 for any queue manager. The properties which may be specified are as follows:

- MQC.CONNECT_OPTIONS_PROPERTY
- MQC.CONNNAME_PROPERTY
- MQC.HOST_NAME_PROPERTY
- MQC.PORT_PROPERTY
- MQC.CHANNEL_PROPERTY
- MQC.SSL_CIPHER_SPEC_PROPERTY
- MQC.SSL_PEER_NAME_PROPERTY
- MQC.SSL_CERT_STORE_PROPERTY
- MQC.SSL_CRYPTO_HARDWARE_PROPERTY
- MQC.SECURITY_EXIT_PROPERTY
- MQC.SECURITY_USERDATA_PROPERTY
- MQC.SEND_EXIT_PROPERTY
- MQC.SEND_USERDATA_PROPERTY
- MQC.RECEIVE_EXIT_PROPERTY
- MQC.RECEIVE_USERDATA_PROPERTY
- MQC.MSG_EXIT_PROPERTY
- MQC.USER_ID_PROPERTY
- MQC.PASSWORD_PROPERTY
- MQC.MQAIR_ARRAY
- MQC.KEY_RESET_COUNT
- MQC.FIPS_REQUIRED
- MQC.HDR_CMP_LIST
- MQC.MSG_CMP_LIST
- MQC.TRANSPORT_PROPERTY

For descriptions of these properties, see the corresponding property description in “MQEnvironment” on page 36. The following example shows program code to create a queue manager with its user ID and password defined in a hash table.

```
Hashtable properties = new Hashtable();

properties.Add( MQC.USER_ID_PROPERTY, "ExampleUserId" );
properties.Add( MQC.PASSWORD_PROPERTY, "ExamplePassword" );

try
{
    mqQMgr = new MQQueueManager("qmgrname", properties);
}
catch (MQException mqe)
```

```
{
    System.Console.WriteLine("Connect failed with " + mqe.Message);
    return((int)mqe.Reason);
}
```

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.

MQQueueManager

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

Throws MQException.

Connects to the named Queue Manager, using the supplied 'Server Connection Channel' and connection, and passing the supplied options.

Methods

Methods for MQQueueManager

AccessProcess

```
public MQProcess AccessProcess(String processName,
                               int openOptions);
```

Throws MQException.

Establishes access to a WebSphere MQ process on this queue manager using the default queue manager name and default user ID values, to inquire about the process attributes.

Parameters

processName
The name of the process to open.

openOptions
Options that control the opening of the process. Valid options are:
- MQOO_FAIL_IF_QUIESCING
- MQOO_INQUIRE
- MQOO_SET

Returns
MQProcess that has been successfully opened.

AccessProcess

```
public MQProcess AccessProcess(String processName,
                               int openOptions,
                               String queueManagerName,
                               String alternateUserId);
```

Throws MQException.

Establishes access to a WebSphere MQ process on this queue manager using the specified queue manager name and specified alternate user ID values, in order to inquire about the process attributes.

Parameters

processName
The name of the process to open.

openOptions
Options that control the opening of the process. Valid options are:
- MQOO_ALTERNATE_USER_AUTHORITY
- MQOO_FAIL_IF_QUIESCING
- MQOO_INQUIRE
- MQOO_SET

queueManagerName
Name of the queue manager on which the process is defined. A name that is entirely blank or null denotes the queue manager to which the object is associated.

alternateUserId
If MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternative user ID to be used to check the authorization for the action. Otherwise this parameter can be blank or null.

Returns
MQProcess that has been successfully opened.

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 want 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 want 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 Guide* 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 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.

AccessTopic

```
public MQTopic AccessTopic(String topicName,
                           ref String topicObject,
                           int openAs,
                           int options);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. The MQTopic object can be opened for either publication or subscription depending upon the value of the openAs parameter. The value dictates the use of the options parameter – this can map to the equivalent MQOO options for publication or the equivalent MQSO options for subscription.

The options specified allow the MQTopic object to be used to get or browse messages, put messages, inquire about the attributes of the topic (those defined on the object), or set the attributes of the topic (those defined on the object).

An MQTopic object cannot be used for both publication and subscription simultaneously. Therefore, the method returns an MQTopic object for publication OR subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

Parameters

topicName
: The topic string to publish or subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
: This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

  The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

openAs
: Indicates whether the topic is being opened for either publication or subscription. The parameter can contain one of these options:

  - MQTOPIC_OPEN_AS_SUBSCRIPTION
  - MQTOPIC_OPEN_AS_PUBLICATION

  The topic object cannot be opened for both publication and subscription. Specifying more than a single option will result in an error condition.

options

: Options that control the opening of the topic for either publication or subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

  These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

  - MQOO_ALTERNATE_USER_AUTHORITY
  - MQOO_FAIL_IF_QUIESCING
  - MQOO_OUTPUT

- MQOO_PASS_ALL_CONTEXT
- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_SET_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

When opening the topic for subscription the following valid options apply.:

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options are specified, then MQSO_CREATE + MQSO_ALTER is assumed.

Other valid options are also available (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to a managed, non-durable subscription only. These options are therefore enforced:

- MQSO_NON_DURABLE
- MQSO_MANAGED

Returns
MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(String topicName,
                           ref String topicObject,
                           int openAs,
                           int options,
                           String alternateUserId);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. The MQTopic object can be opened for either publication or subscription depending upon the value of the openAs parameter. The value dictates the use of the options parameter – this can map to the equivalent MQOO options for publication or the equivalent MQSO options for subscription.

If either MQOO_ALTERNATE_USER_AUTHORITY or MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, put messages, inquire about the attributes of the topic (those defined on the object), or set the attributes of the topic (those defined on the object).

An MQTopic object cannot be used for both publication and subscription simultaneously. Therefore, the method returns an MQTopic object for publication OR subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the

resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

Parameters

topicName
The topic string to publish or subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

openAs Indicates whether the topic is being opened for either publication or subscription. The parameter can contain one of these options:

- MQTOPIC_OPEN_AS_SUBSCRIPTION
- MQTOPIC_OPEN_AS_PUBLICATION

The topic object cannot be opened for both publication and subscription. Specifying more than a single option will result in an error condition.

options

Options that control the opening of the topic for either publication or subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQOO_ALTERNATE_USER_AUTHORITY
- MQOO_FAIL_IF_QUIESCING
- MQOO_OUTPUT
- MQOO_PASS_ALL_CONTEXT

- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_SET_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

When opening the topic for subscription the following valid options apply:

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If no option is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to a managed, non-durable subscription only. These options are therefore enforced:

- MQSO_NON_DURABLE
- MQSO_MANAGED

alternateUserId

If either MQOO_ALTERNATE_USER_AUTHORITY or MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

Returns

MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(String topicName,
                           ref String topicObject,
                           int options,
                           String alternateUserId,
                           String subscriptionName);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. This method can be used for opening the topic for subscriptions only. The options parameter can map to the MQSO options for subscription only.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the

resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

Parameters

topicName
: The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
: This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

  The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

  Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

  These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

  - MQSO_CREATE
  - MQSO_RESUME
  - MQSO_ALTER

  If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

  When opening a topic for subscription, the method applies to a managed subscription only. This option is therefore enforced:

  - MQSO_MANAGED

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is mandatory, otherwise (MQSO_NON_DURABLE) this field is optional.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

Returns

MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(String topicName,
                           ref String topicObject,
                           int options,
                           String alternateUserId,
                           String subscriptionName,
                           ref Hashtable parameters);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. This method can be used for opening the topic for subscriptions only. The options parameter can map to the MQSO options for subscription only.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

Extra non-standard input and output parameters can also be specified using the parameters hash table. If a property is an output field it will be populated within the hash table only if it was originally specified on input. Essentially, no new key/value pairs will be added to the hash table – only existing ones updated.

Parameters

topicName
: The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
: This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

  The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

  Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

  These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

  - MQSO_CREATE
  - MQSO_RESUME
  - MQSO_ALTER

  If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

  When opening a topic for subscription, the method applies to a managed subscription only. This option is therefore enforced:

  - MQSO_MANAGED

alternateUserId

  If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is mandatory, otherwise (MQSO_NON_DURABLE) this field is optional.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

parameters

The hash table can be used to specify non-standard input and output parameters to the subscription request. If a property is an output field it will only be populated within the hash table if it was originally specified on input. Consequently, no new key/value pairs will be added to the hash table – only existing ones updated. The following key names are valid and can be specified:

- MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQSUB_PROP_PUBLICATION_PRIORITY
- MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

All are specified as String type properties. The corresponding language conversion routines can be used to convert the values to the relevant types.

Returns

MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(MQDestination destination,
                           String topicName,
                           ref String topicObject,
                           int options);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. This method can be used for opening the topic for subscriptions only. The options parameter can map to the MQSO options for subscription only.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the

resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscription, the subscription store (destination) is provided, managed and owned by the user. The queue manager takes no responsibility for this object and it is left to the user to correctly dispose of it. Any messages available for this subscription will be delivered to the specified destination.

The destination parameter must be valid and cannot be left blank or null. This method can be used to create an unmanaged, non-durable subscription only.

Parameters

destination

An existing MQDestination object which should receive the publications. For WebSphere MQ V7.0 this object maps to an MQQueue object. It cannot resolve to another MQTopic object.

The MQDestination (MQQueue) object can be created by calling an MQQueueManager::AccessQueue (...) method or an MQQueue constructor.

The corresponding destination is held as a reference within the MQTopic object as the UnmanagedDestinationReference property.

topicName

The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject

This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to an unmanaged, non-durable subscription only. These options are therefore enforced:

- MQSO_NON_DURABLE
- ~ MQSO_MANAGED

Returns
MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(MQDestination destination,
                           String topicName,
                           ref String topicObject,
                           int options,
                           String alternateUserId);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. The presence of the destination parameter indicates that this method can be used for opening the topic for subscriptions only. The options parameter therefore always maps to the equivalent MQSO values.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscription, the subscription store (destination) is provided, managed and owned by the user. The queue manager takes no responsibility for this object and it is left to the user to correctly dispose of it. Any messages available for this subscription will be delivered to the specified destination.

The destination parameter must be valid and cannot be left blank or null. This method can be used to create an unmanaged, non-durable subscription only.

Parameters

destination

An existing MQDestination object which should receive the publications. For WebSphere MQ V7.0 this object maps to an MQQueue object. It cannot resolve to another MQTopic object.

The MQDestination (MQQueue) object can be created by calling an MQQueueManager::AccessQueue (...) method or an MQQueue constructor.

The corresponding destination is held as a reference within the MQTopic object as the UnmanagedDestinationReference property.

topicName

The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject

This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to an unmanaged, non-durable subscription only. These options are therefore enforced:

- MQSO_NON_DURABLE
- ~ MQSO_MANAGED

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

Returns

MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(MQDestination destination,
                           String topicName,
                           ref String topicObject,
                           int options,
                           String alternateUserId,
                           String subscriptionName);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. The presence of the destination and subscriptionName parameters indicate that this method can be used for opening the topic for subscriptions only. The options parameter therefore always maps to the equivalent MQSO values.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscription, the subscription store (destination) is provided, managed and owned by the user. The queue manager takes no responsibility for this object and it is left to the user to correctly dispose of it. Any messages available for this subscription will be delivered to the specified destination.

The destination parameter must be valid and cannot be left blank or null. This method can be used to create an unmanaged subscription only.

Parameters

destination

An existing MQDestination object which should receive the publications. For WebSphere MQ V7.0 this object maps to an MQQueue object. It cannot resolve to another MQTopic object.

The MQDestination (MQQueue) object can be created by calling an MQQueueManager::AccessQueue (...) method or an MQQueue constructor.

The corresponding destination is held as a reference within the MQTopic object as the UnmanagedDestinationReference property.

topicName

The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject

This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to an unmanaged subscription only. This option is therefore enforced:

- ~ MQSO_MANAGED

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is mandatory, otherwise (MQSO_NON_DURABLE) this field is optional.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

Returns
MQTopic that has been successfully opened.

AccessTopic

```
public MQTopic AccessTopic(MQDestination destination,
                           String topicName,
                           ref String topicObject,
                           int options,
                           String alternateUserId,
                           String subscriptionName,
                           ref Hashtable parameters);
```

Throws MQException.

Establishes access to a WebSphere MQ topic. The presence of the destination and subscriptionName parameters indicate that this method can be used for opening the topic for subscriptions only. The options parameter therefore always maps to the equivalent MQSO values.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

The method returns an MQTopic object for subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a '/' character is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

When creating an MQTopic for subscription, the subscription store (destination) is provided, managed and owned by the user. The queue manager takes no responsibility for this object and it is left to the user to correctly dispose of it. Any messages available for this subscription will be delivered to the specified destination.

The destination parameter must be valid and cannot be left blank or null. This method can be used to create an unmanaged subscription only.

Extra non-standard input and output parameters can also be specified using the parameters hash table. If a property is an output field it will be populated within the hash table only if it was originally specified on input. Essentially, no new key/value pairs will be added to the hash table – only existing ones updated.

Parameters

destination

An existing MQDestination object which should receive the publications. For WebSphere MQ V7.0 this object maps to an MQQueue object. It cannot resolve to another MQTopic object.

The MQDestination (MQQueue) object can be created by calling an MQQueueManager::AccessQueue (...) method or an MQQueue constructor.

The corresponding destination is held as a reference within the MQTopic object as the UnmanagedDestinationReference property.

topicName

The topic string to subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject

This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method, the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to an unmanaged subscription only. This option is therefore enforced:

- ~ MQSO_MANAGED

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is mandatory, otherwise (MQSO_NON_DURABLE) this field is optional.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

parameters

The hash table can be used to specify non-standard input and output parameters to the subscription request. If a property is an output field it will only be populated within the hash table if it was originally specified on input. Consequently, no new key / value pairs will be added to the hash table – only existing ones updated. The following key names are valid and can be specified:

- MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQSUB_PROP_PUBLICATION_PRIORITY
- MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

All are specified as String type properties. The corresponding language conversion routines can be used to convert the values to the relevant types.

Returns

MQTopic that has been successfully opened.

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.

GetAsyncStatus

```
public MQAsyncStatus GetAsyncStatus()
```

Throws MQException;

Creates an MQAsyncStatus object that represents the asynchronous activity for the queue manager connection.

Returns

An asynchronous status object containing the values of the last asynchronous errors for the queue manager connection.

Exceptions

MQException – if there is a problem retrieving the asynchronous error status information.

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.

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 81 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) 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) 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) for details of the parameters.

Put

```
public void Put(int type,
                ref String destinationName,
                ref MQMessage message);
```

Throws MQException.

Places or publishes a single message onto a queue or topic without having to create an MQQueue or MQTopic object first.

When used to place messages to a queue allows you to omit the queue manager name, put message options, and alternative user ID. When used to publish messages to a topic allows you to omit the topic string, put message options and alternative user ID.

The destinationName parameter adopts different meanings depending upon the destination type specified in the type parameter.

The default options used for unspecified parameters might alter depending upon the destination type specified.

Parameters

type The options used to control the specified destination type. Valid values are:

- MQOT_Q

- MQOT_TOPIC

Only one option should be specified – the values should not be combined by addition or using the bitwise OR operator.

destinationName

The name of the queue or topic onto which to place or publish the message. Depending upon the value of the type parameter this parameter will adopt different meanings:

- If MQOT_Q is specified the destinationName parameter directly maps to the queue name, that is the ObjectName property of the MQOD.
- If MQOT_TOPIC is specified the destinationName parameter directly maps to the topic object, that is the ObjectName property of the MQSD (the topic object ObjectName property is left blank).

message

The message to send. Properties within the message might be altered as a result of placing or publishing the message to a queue or topic. It is therefore marked as both an input and output parameter.

Returns

There is no return value.

Put

```
public void Put(int type,
                ref String destinationName,
                ref MQMessage message
                MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Places or publishes a single message onto a queue or topic without having to create an MQQueue or MQTopic object first.

When used to place messages to a queue allows you to omit the queue manager name and alternative user ID. When used to publish messages to a topic allows you to omit the topic string and alternative user ID.

The destinationName parameter adopts different meanings depending upon the destination type being specified in the type parameter.

The options specified in the putMessageOptions parameter are used when putting the message to either the queue or topic. These options differ depending upon the destination type being specified.

Parameters

type

The options used to control the specified destination type. Valid values are:

- MQOT_Q
- MQOT_TOPIC

Only one option should be specified – the values should not be combined by addition or using the bitwise OR operator.

destinationName

The name of the queue or topic onto which to place or publish the message. Depending upon the value of the type parameter this parameter will adopt different meanings:

- If MQOT_Q is specified the destinationName parameter directly maps to the queue name, that is the ObjectName property of the MQOD.
- If MQOT_TOPIC is specified the destinationName parameter directly maps to the topic object, that is the ObjectName property of the MQSD (the topic object ObjectName property is left blank).

message
The message to send. Properties within the message might be altered as a result of placing or publishing the message to a queue or topic. It is therefore marked as both an input and output parameter.

putMessageOptions
Options controlling the action of the put or publish. See MQPutMessageOptions object “Properties” on page 82.

Returns
There is no return value.

Put

```
public void Put(int type,
              ref String destinationName,
              String queueManagerName,
              String topicString,
              ref MQMessage message);
```

Throws MQException.

Places or publishes a single message onto a queue or topic without having to create an MQQueue or MQTopic object first.

When used to place messages to a queue or a topic allows you to omit the put message options, and alternative user ID.

The destinationName parameter adopts different meanings depending upon the destination type being specified in the type parameter.

Other parameters are optional.

Parameters

type The options used to control the specified destination type. Valid values are:

- MQOT_Q
- MQOT_TOPIC

Only one option should be specified – the values should not be combined by addition or using the bitwise OR operator.

destinationName

The name of the queue or topic onto which to place or publish the message. Depending upon the value of the type parameter this parameter will adopt different meanings:

- If MQOT_Q is specified the destinationName parameter directly maps to the queue name, that is the ObjectName property of the MQOD.

- If MQOT_TOPIC is specified the destinationName parameter directly maps to the topic object, that is the ObjectName property of the MQSD (the topic object ObjectName property is left blank).

queueManagerName
: The name of the queue manager onto which to place the message. If type MQOT_TOPIC is specified this parameter is ignored.

topicString
: The name of the topic string onto which to publish the message. If type MQOT_Q is specified this parameter is ignored.

message
: The message to send. Properties within the message might be altered as a result of placing or publishing the message to a queue or topic. It is therefore marked as both an input and output parameter.

Returns
: There is no return value.

Put

```
public void Put(int type,
                ref String destinationName,
                String queueManagerName,
                String topicString,
                ref MQMessage message,
                MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Places or publishes a single message onto a queue or topic without having to create an MQQueue or MQTopic object first.

When used to place messages to a queue or a topic allows you to omit the alternative user ID.

The destinationName parameter adopts different meanings depending upon the destination type being specified in the type parameter.

The options specified in the putMessageOptions parameter are used when putting the message to either the queue or topic. These options differ depending upon the destination type being specified.

Other parameters are optional.

Parameters

type The options used to control the specified destination type. Valid values are:

- MQOT_Q
- MQOT_TOPIC

Only one option should be specified – the values should not be combined by addition or using the bitwise OR operator.

destinationName

The name of the queue or topic onto which to place or publish the message. Depending upon the value of the type parameter this parameter will adopt different meanings:

- If MQOT_Q is specified the destinationName parameter directly maps to the queue name, that is the ObjectName property of the MQOD.
- If MQOT_TOPIC is specified the destinationName parameter directly maps to the topic object, that is the ObjectName property of the MQSD (the topic object ObjectName property is left blank).

queueManagerName
: The name of the queue manager onto which to place the message. If type MQOT_TOPIC is specified this parameter is ignored.

topicString
: The name of the topic string onto which to publish the message. If type MQOT_Q is specified this parameter is ignored.

message
: The message to send. Properties within the message might be altered as a result of placing or publishing the message to a queue or topic. It is therefore marked as both an input and output parameter.

putMessageOptions
: Options controlling the action of the put or publish. See MQPutMessageOptions object "Properties" on page 82.

Returns
: There is no return value.

Put

```
public void Put(int type,
                ref String destinationName,
                String queueManagerName,
                String topicString,
                ref MQMessage message,
                MQPutMessageOptions putMessageOptions
                String alternateUserId);
```

Throws MQException.

Places or publishes a single message onto a queue or topic without having to create an MQQueue or MQTopic object first.

The destinationName parameter adopts different meanings depending upon the destination type being specified in the type parameter.

The options specified in the putMessageOptions parameter are used when putting the message to either the queue or topic. These options differ depending upon the destination type being specified.

The alternateUserId parameter is an alternative user identifier used to check authorization when placing or publishing the message onto a queue or topic.

Other parameters are optional.

Parameters

type
: The options used to control the specified destination type. Valid values are:
- MQOT_Q
- MQOT_TOPIC

Only one option should be specified – the values should not be combined by addition or using the bitwise OR operator.

destinationName

The name of the queue or topic onto which to place or publish the message. Depending upon the value of the type parameter this parameter will adopt different meanings:

- If MQOT_Q is specified the destinationName parameter directly maps to the queue name, that is the ObjectName property of the MQOD.
- If MQOT_TOPIC is specified the destinationName parameter directly maps to the topic object, that is the ObjectName property of the MQSD (the topic object ObjectName property is left blank).

queueManagerName
The name of the queue manager onto which to place the message. If type MQOT_TOPIC is specified this parameter is ignored.

topicString
The name of the topic string onto which to publish the message. If type MQOT_Q is specified this parameter is ignored.

message
The message to send. Properties within the message might be altered as a result of placing or publishing the message to a queue or topic. It is therefore marked as both an input and output parameter.

putMessageOptions
Options controlling the action of the put or publish. See MQPutMessageOptions object "Properties" on page 82.

alternateUserId
Specifies an alternative user identifier used to check authorization when placing or publishing the message onto a queue or topic. If you do not specify MQPMO_ALTERNATE_USER, this parameter is ignored.

Returns
There is no return value.

Properties

Properties for MQQueueManager.

AccountingConnOverride

```
public int AccountingConnOverride {get;}
```

Allows applications to override the setting of the mqi accounting and queue accounting values.

AccountingInterval

```
public int AccountingInterval {get;}
```

How long before intermediate accounting records are written (in seconds).

ActivityRecording

```
public int ActivityRecording {get;}
```

Controls the generation of activity reports.

AdoptNewMCACheck

```
public int AdoptNewMCACheck {get;}
```

Specifies which elements are checked to determine whether an MCA will be adopted when a new inbound channel is detected with the same name as an already active MCA.

AdoptNewMCAInterval

```
public int AdoptNewMCAInterval {get;}
```

The amount of time, in seconds, that the new channel waits for the orphaned channel to end.

AdoptNewMCAType

```
public int AdoptNewMCAType {get;}
```

Whether an orphaned MCA instance is to be adopted (restarted) when a new inbound channel request is detected matching the AdoptNewMCACheck value.

BridgeEvent

```
public int BridgeEvent {get;}
```

Whether IMS Bridge events are generated.

ChannelEvent

```
public int ChannelEvent {get;}
```

Whether channel events are generated.

ChannelInitiatorControl

```
public int ChannelInitiatorControl {get;}
```

Whether the channel initiator starts automatically when the queue manager starts.

ChannelInitiatorAdapters

```
public int ChannelInitiatorAdapters {get;}
```

The number of adapter subtasks to process WebSphere MQ calls.

ChannelInitiatorDispatchers

```
public int ChannelInitiatorDispatchers {get;}
```

The number of dispatchers to use for the channel initiator.

ChannelInitiatorTraceAutoStart

```
public int ChannelInitiatorTraceAutoStart {get;}
```

Specifies whether the channel initiator trace starts automatically.

ChannelInitiatorTraceTableSize

```
public int ChannelInitiatorTraceTableSize {get;}
```

The size, in megabytes, of the channel initiator's trace data space.

ChannelMonitoring

```
public int ChannelMonitoring {get;}
```

Whether channel monitoring is enabled.

ChannelStatistics

```
public int ChannelStatistics {get;}
```

Controls the collection of statistics data for channels.

CharacterSet

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

Throws MQException.

Returns the CCSID (Coded Character Set Identifier) of the queue manager's code set. 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.

ClusterSenderMonitoring

```
public int ClusterSenderMonitoring {get;}
```

Controls the collection of online monitoring data for automatically-defined cluster sender channels.

ClusterSenderStatistics

```
public int ClusterSenderStatistics {get;}
```

Controls the collection of statistics data for automatically defined cluster sender channels.

ClusterWorkLoadMRU

```
public int ClusterWorkLoadMRU {get;}
```

The maximum number of outbound cluster channels.

ClusterWorkLoadUseQ

```
public int ClusterWorkLoadUseQ {get;}
```

The default value of the MQQueue property, ClusterWorkLoadUseQ, if it specifies a value of QMGR.

CommandEvent

```
public int CommandEvent {get;}
```

Specifies whether command events are generated.

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

CommandServer

```
public int CommandServer {get;}
```

Whether the command server starts automatically when the queue manager starts.

DNSGroup

```
public string DNSGroup {get;}
```

The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group must join when using Workload Manager for Dynamic Domain Name Services support (DDNS).

DNSWLM

```
public int DNSWLM {get;}
```

Whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with Workload Manager for DDNS.

IPAddressVersion

```
public int IPAddressVersion {get;}
```

Which IP protocol (IPv4 or IPv6) to use for a channel connection.

IsConnected

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

Returns the value of the isConnected variable.

KeepAlive

```
public int KeepAlive {get;}
```

Specifies whether the TCP KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is not available, the channel is closed.

ListenerTimer

```
public int ListenerTimer {get;}
```

The time interval, in seconds, between attempts by WebSphere MQ to restart the listener after an APPC or TCP/IP failure.

LoggerEvent

```
public int LoggerEvent {get;}
```

Whether logger events are generated.

LU62ARMSuffix

```
public string LU62ARMSuffix {get;}
```

The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command SET APPC=xx is issued.

LUGroupName

```
public string LUGroupName {get;}
```

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

LUName

```
public string LUName {get;}
```

The name of the LU to use for outbound LU 6.2 transmissions.

MaximumActiveChannels

```
public int MaximumActiveChannels {get;}
```

The maximum number of channels that can be active at any time.

MaximumCurrentChannels

```
public int MaximumCurrentChannels {get;}
```

The maximum number of channels that can be current at any time (including server-connection channels with connected clients).

MaximumLU62Channels

```
public int MaximumLU62Channels {get;}
```

The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

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.

MaximumTCPChannels

```
public int MaximumTCPChannels {get;}
```

The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

MQIAccounting

```
public int MQIAccounting {get;}
```

Controls the collection of accounting information for MQI data.

MQIStatistics

```
public int MQIStatistics {get;}
```

Controls the collection of statistics monitoring information for the queue manager.

OutboundPortMax

```
public int OutboundPortMax {get;}
```

The maximum value in the range of port numbers to be used when binding outgoing channels.

OutboundPortMin

```
public int OutboundPortMin {get;}
```

The minimum value in the range of port numbers to be used when binding outgoing channels.

QueueAccounting

```
public int QueueAccounting {get;}
```

Whether class 3 accounting (thread-level and queue-level accounting) data is to be enabled for all queues.

QueueMonitoring

```
public int QueueMonitoring {get;}
```

Controls the collection of online monitoring data for queues.

QueueStatistics

```
public int QueueStatistics {get;}
```

Controls the collection of statistics data for queues.

ReceiveTimeout

```
public int ReceiveTimeout {get;}
```

The length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

ReceiveTimeoutMin

```
public int ReceiveTimeoutMin {get;}
```

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state.

ReceiveTimeoutType

```
public int ReceiveTimeoutType {get;}
```

The qualifier to apply to the value in ReceiveTimeout.

SharedQueueQueueManagerName

```
public int SharedQueueQueueManagerName {get;}
```

Whether the ObjectQmgrName must be used or treated as the local queue manager on an MQOPEN call for a shared queue when the ObjectQmgrName is that of another queue manager in the queue-sharing group.

SSLEvent

```
public int SSLEvent {get;}
```

Whether SSL events are generated.

SSLFips

```
public int SSLFips {get;}
```

Whether only FIPS-certified algorithms are to be used if cryptography is executed in WebSphere MQ

itself.

SSLKeyResetCount

```
public int SSLKeyResetCount {get;}
```

Indicates the number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.

StatisticsInterval

```
public int ClusterSenderStatistics {get;}
```

Specifies the interval, in minutes, between consecutive gatherings of statistics.

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.

TCPName

```
public string TCPName {get;}
```

The name of either the only, or default, TCP/IP system to be used, depending on the value of TCPStackType.

TCPStackType

```
public int TCPStackType {get;}
```

Specifies whether the channel initiator may use only the TCP/IP address space specified in TCPNAME, or may optionally bind to any selected TCP/IP address.

TraceRouteRecording

```
public int TraceRouteRecording {get;}
```

Controls the recording of route tracing information.

For more detailed descriptions of these properties, see Attribute descriptions for the queue manager.

MQSubscription

MQSubscription object for .NET

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

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

MQSubscription is a helper object designed to encapsulate the HSUB reference. Under normal operating circumstances do not use or modify the object. It is a sub class of MQManagedObject.

Constructors

Constructors for MQSubscription.

MQSubscription

```
protected MQSubscription();
```

Default constructor. Although not an abstract base class, the MQSubscription object is set to inhibit construction of these objects. Instead an MQSubscription object is created automatically when an MQTopic object is created for subscriptions, and the reference to the MQSubscription object is held in the MQTopic (SubscriptionReference) object. The reference is available to you if you want to modify the close options or invoke any of the object's methods.

By default the close options of the MQSubscription object are set to MQCO_NONE, which means the queue manager decides which close options to use depending upon the subscription type.

Methods

Methods for MQSubscription object.

RequestPublicationUpdate

```
public int RequestPublicationUpdate(int options);
```

Throws MQException.

Requests an update publication to be sent for the current topic. This is normally used if the user specified the MQSO_PUBLICATIONS_ON_REQUEST option. If the queue manager has a retained publication for the topic, this is sent to the subscriber.

The method returns the number of retained publications to be sent to the subscription queue. There is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There might be more than one publication if the subscribed topic contained a wildcard. If no wildcards were present in the topic string when the subscription was made, then only one publication will be sent as a result of this call.

Parameters

options This parameter maps directly to the options field of the MQSRO structure. Any or none of these options can be specified:

MQSRO_FAIL_IF_QUIESCING
The method fails if the queue manager is in a quiescent state. On z/OS, for a CICS or IMS application, this option also forces the method to fail if the connection is in a quiescent state.

MQSRO_NONE
If none of the options described above are required, use this value to indicate that no other options have been specified.

Inquire

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

Although available on the MQManagedObject base class, this method has no relevance to MQSubscription and is inhibited.

Set

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

Although available on the MQManagedObject base class, this method has no relevance to MQSubscription and is inhibited.

GetAttributeString

```
public String GetAttributeString(int selector,
                                 int length);
```

Although available on the MQManagedObject base class, this method has no relevance to MQSubscription and is inhibited.

SetAttributeString

```
public String SetAttributeString(int selector,
                                 String value,
                                 int length);
```

Although available on the MQManagedObject base class, this method has no relevance to MQSubscription and is inhibited.

Properties

Properties for MQSubscription.

AlternateUserId

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

Although available on the MQManagedObject base class, this property has no relevance to MQSubscription and is disabled.

Description

```
public String AlternateUserId { get; }
```

Although available on the MQManagedObject base class, this property has no relevance to MQSubscription and is disabled.

OpenOptions

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

Although available on the MQManagedObject base class, this property has no relevance to MQSubscription and is disabled.

MQTopic

MQTopic object for .NET

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

public class **IBM.WMQ.MQTopic**
extends **IBM.WMQ.MQDestination**

MQTopic is a sub class of MQDestination and provides set, inquire, put (send/publish), and get (receive/subscribe) operations for WebSphere MQ topics. The set and inquire capabilities are inherited from MQManagedObject. The put and get capabilities are inherited from MQDestination.

Use either the MQTopic constructors or the MQQueueManager::AccessTopic(...) methods to gain access to an MQTopic object. An MQTopic object can be accessed for either publication or subscription, not both simultaneously.

When used for receiving messages the MQTopic object can be created with an unmanaged or managed subscription, and as a durable or non-durable subscriber – multiple overloaded constructors are provided for this.

Note: a subscription can be either managed or unmanaged, irrespective of whether you are using a managed client connection. For more information about managed subscriptions, see *Websphere MQ Publish/Subscribe User's Guide*. For more information about .NET managed code, refer to Microsoft documentation.

Constructors for MQTopic

Constructors for MQTopic.

MQTopic has eight constructors. Two can be used for publications or for managed subscriptions, two for managed subscriptions only, and four for unmanaged subscriptions only.

Publication or managed subscription

MQTopic

```
public MQTopic(MQQueueManager qMgr,
               String topicName,
               ref String topicObject,
               int openAs,
               int options);
```

```
public MQTopic(MQQueueManager qMgr,
               String topicName,
               ref String topicObject,
               int openAs,
               int options,
               String alternateUserId);
```

Either of these constructors establishes access to a topic on the specified queue manager. The MQTopic object can be opened for either publication or subscription depending upon the value of the openAs parameter. The value dictates the use of the options parameter that should contain MQOO options for publication or MQSO options for subscription.

An MQTopic object cannot be used for both publication and subscription simultaneously. The constructor creates an MQTopic object for either publication or subscription using the supplied topic name (topicName) and topic object (topicObject).

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a forward slash (/) is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

The options specified allow the MQTopic object to be used to get or browse messages, put messages, inquire about the attributes of the topic, or set the attributes of the topic.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

In the second variant, if either MQOO_ALTERNATE_USER_AUTHORITY or MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

Parameters

qMgr The object that represents the queue manager on which the topic resides.

topicName
The topic string to publish or subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

openAs Indicates whether the topic is being opened for either publication or subscription. The parameter can contain one of these options:

- MQTOPIC_OPEN_AS_SUBSCRIPTION
- MQTOPIC_OPEN_AS_PUBLICATION

The topic object cannot be opened for both publication and subscription. Specifying more than a single option will result in an error condition.

options

Options that control the opening of the topic for either publication or subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQOO_ALTERNATE_USER_AUTHORITY
- MQOO_FAIL_IF_QUIESCING
- MQOO_OUTPUT
- MQOO_PASS_ALL_CONTEXT
- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_SET_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

When opening the topic for subscription the following valid options apply:

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options are specified, then MQSO_CREATE + MQSO_ALTER is assumed.

Other valid options are also available (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to a managed, non-durable subscription only. These options are therefore enforced:

- MQSO_NON_DURABLE
- MQSO_MANAGED

alternateUserId

If either MQOO_ALTERNATE_USER_AUTHORITY or MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

Managed subscription

MQTopic

```
public MQTopic(MQQueueManager qMgr,
               String topicName,
               ref String topicObject,
               int options,
               String alternateUserId,
               String subscriptionName);
```

```
public MQTopic(MQQueueManager qMgr,
               String topicName,
               ref String topicObject,
               int options,
               String alternateUserId,
               String subscriptionName,
               ref Hashtable parameters);
```

Either of these constructors establishes access to a topic on the specified queue manager. These methods can be used for opening the topic for subscriptions only. The options parameter can map to the MQSO options for subscription only.

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a forward slash (/) is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

When creating an MQTopic for subscriptions, the store (MQQueue) will be managed and owned by the queue manager. This method can therefore be used to create a managed subscription only.

In the second variant, extra non-standard input and output parameters can also be specified using the parameters hash table. If a property is an output field it will be populated within the hash table only if it was originally specified on input. Essentially, no new key/value pairs will be added to the hash table – only existing ones updated.

Parameters

qMgr
: The object that represents the queue manager on which the topic resides.

topicName
: The topic string to publish or subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
: This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

  The parameter is both an input and output parameter. Upon successful completion of the method the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options
: Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

  When opening the topic for subscription the following valid options apply:
  - MQSO_CREATE
  - MQSO_RESUME
  - MQSO_ALTER

  If none of these options are specified, then MQSO_CREATE + MQSO_ALTER is assumed.

  Other valid options are also available (see MQOPEN – Open object Options).

  When opening a topic for subscription, the method applies to a managed subscription only. This option is therefore enforced:
  - MQSO_MANAGED

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is required, otherwise if this field is provided it will be used by the queue manager for MQSO_NON_DURABLE as well.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

parameters

The hash table can be used to specify non-standard input and output parameters to the subscription request. If a property is an output field it will only be populated within the hash table if it was originally specified on input. Consequently, no new key/value pairs will be added to the hash table – only existing ones updated. The following key names are valid and can be specified:

- MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQSUB_PROP_PUBLICATION_PRIORITY
- MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

All are specified as String type properties. The corresponding language conversion routines can be used to convert the values to the relevant types.

Unmanaged subscription

MQTopic

```
public MQTopic(MQQueueManager qMgr,
               MQDestination destination,
               String topicName,
               ref String topicObject,
               int options);
```

```
public MQTopic(MQQueueManager qMgr,
               MQDestination destination,
               String topicName,
               ref String topicObject,
               int options,
               String alternateUserId);
```

```
public MQTopic(MQQueueManager qMgr,
               MQDestination destination,
               String topicName,
               ref String topicObject,
               int options,
               String alternateUserId,
               String subscriptionName);
public MQTopic(MQQueueManager qMgr,
               MQDestination destination,
               String topicName,
               ref String topicObject,
               int options,
               String alternateUserId,
               String subscriptionName,
               ref Hashtable parameters);
```

These constructors establish access to a topic on the specified queue manager and can be used for opening the topic for subscriptions only. The options parameter can map to the MQSO options for subscription only.

The destination parameter must be valid and cannot be left blank or null.

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character. If both parts exist a forward slash (/) is inserted between them in the resultant combined topic. If only one of these parts exist it is used unchanged as the topic. The parts are concatenated in the sequence listed here:

- The value of the TOPICSTR parameter of the topic object named in topicObject.
- topicName, if the length provided for that string is non-zero.

If there are wildcards in the topicName, the interpretation of those wildcards can be controlled using the wildcard options specified in the options parameter.

The options specified allow the MQTopic object to be used to get or browse messages, inquire about the attributes of the topic, or set the attributes of the topic.

In the variants with a subscriptionName parameter, if MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, the alternateUserId parameter is used to check for the necessary authorization.

When creating an MQTopic for subscription, the subscription store (destination) is provided, managed and owned by the user. The queue manager takes no responsibility for this object and it is left to the user to correctly dispose of it. Any messages available for this subscription will be delivered to the specified destination. This method can therefore be used to create an unmanaged subscription only.

Variants of the constructor without the subscriptionName parameter can be used to create nondurable subscriptions only.

Extra nonstandard input and output parameters can also be specified using the parameters hash table. If a property is an output field it will be populated within the hash table only if it was originally specified on input. Essentially, no new key/value pairs will be added to the hash table – only existing ones updated.

Parameters

qMgr The object that represents the queue manager on which the topic resides.

destination

An existing MQDestination object which should receive the publications. For WebSphere MQ V7.0 this object maps to an MQQueue object. It cannot resolve to another MQTopic object.

The MQDestination (MQQueue) object can be created by calling an MQQueueManager::AccessQueue (...) method or an MQQueue constructor.

The corresponding destination is held as a reference within the MQTopic object as the UnmanagedDestinationReference property.

topicName
The topic string to publish or subscribe against. The topicName parameter directly maps to the ObjectString field of the MQSD. The full topic name used is the combination of the topicObject and topicName parameters as described above.

topicObject
This is the name of the topic object as defined on the local queue manager. If this property is specified in combination with a non-zero-length topicName, then the specified topicName is appended to the topic string contained in the topic object with a separator character. It is the full topic string that is published or subscribed against, as described above.

The parameter is both an input and output parameter. Upon successful completion of the method the closest matching administrative node is located within the topic hierarchy and returned. The contained topic object might therefore differ to that originally specified.

options

Options that control the opening of the topic for subscription. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

These options are valid (see MQOPEN – Open object Options for a full descriptive list of which options are valid when opening a topic):

- MQSO_CREATE
- MQSO_RESUME
- MQSO_ALTER

If none of these options is specified, then MQSO_CREATE + MQSO_ALTER is assumed. Other valid options are also available. (see MQOPEN – Open object Options).

When opening a topic for subscription, the method applies to an unmanaged subscription only. This option is therefore enforced:

- MQSO_MANAGED

If you are using a variant of the method without the subscriptionName parameter, this option is also enforced:

- MQSO_NON_DURABLE

alternateUserId

If MQSO_ALTERNATE_USER_AUTHORITY is specified in the options parameter, this parameter specifies the alternate user identifier that is used to check for the required authorization to complete the operation. Otherwise, this parameter can be left blank (or null).

subscriptionName

If the options parameter specified MQSO_DURABLE then this field is required, otherwise if this field is provided it will be used by the queue manager for MQSO_NON_DURABLE as well.

For an MQSO_DURABLE subscription it is the means by which you identify a subscription to be resumed after it has been created, if you have either closed the handle to the subscription or have been disconnected from the queue manager.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed.

parameters

The hash table can be used to specify non-standard input and output parameters to the subscription request. If a property is an output field it will only be populated within the hash table if it was originally specified on input. Consequently, no new key/value pairs will be added to the hash table – only existing ones updated. The following key names are valid and can be specified:

- MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQSUB_PROP_PUBLICATION_PRIORITY
- MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

All are specified as String type properties. The corresponding language conversion routines can be used to convert the values to the relevant types.

Methods

Methods for MQTopic object.

Put

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

Throws MQException.

Publishes a message to the topic. This method uses a default instance of MQPutMessageOptions to perform the put or publish. The default MQPutMessageOptions instance differs depending upon the destination type.

Parameters

message

An MQMessage object containing the Message Descriptor data (MQMD) and message to be sent. 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 published to the topic.

Put

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

Throws MQException.

Publishes a message to the topic.

Parameters

message
: An MQMessage object containing the Message Descriptor data (MQMD) and message to be sent. 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 published to the topic.

putMessageOptions
: Options controlling the action of the put (See "MQPutMessageOptions" on page 81).

Get

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

Throws MQException.

Retrieves a message from the topic. 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 make sure you set these 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.

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

Parameters

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

Get

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

Throws MQException.

Retrieves a message from the topic. 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 make sure you set these 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.

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 40).

Get

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

Throws MQException.

Retrieves a message from the topic, up to the 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 make sure you set these 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.

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 40).

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.

Properties

Properties for MQTopic.

IsDurable

```
public Boolean IsDurable { get; };
```

Read only property that returns True if the subscription is durable or False otherwise. If the topic was opened for output, (publication), the property is ignored and will always return False.

IsManaged

```
public Boolean IsManaged { get; };
```

Read only property that returns True if the subscription is managed by the queue manager, or False otherwise. If the topic was opened for output (publication), the property is ignored and will always return False.

IsSubscribed

```
public Boolean IsSubscribed { get; };
```

Read only property that returns True if the topic was opened for subscription and False if the topic was opened for publication.

SubscriptionReference

```
public MQSubscription SubscriptionReference { get; };
```

Read only property that returns the MQSubscription object associated with a topic object opened for subscription. The reference is available if you want to modify the close options or invoke any of the objects methods.

UnmanagedDestinationReference

```
public MQDestination UnmanagedDestinationReference { get; };
```

Read only property that returns the MQDestination (MQQueue) associated with an unmanaged subscription. This is the destination specified when the topic object was created. The property will return null for any topic objects opened for publication or with a managed subscription.

IMQObjectTrigger

To use the .NET Monitor, write a component that implements the IMQObjectTrigger interface.

```
IBM.WMQMonitor.IMQObjectTrigger
```

public interface IBM.WMQMonitor.IMQObjectTrigger

Methods of IMQObjectTrigger

IMQObjectTrigger defines a single method, Execute.

Execute

```
void Execute (MQQueueManager qmgr, MQQueue queue, MQMessage message, string param);
```

Passes the queue manager, queue, message, and the user parameter string supplied to the current instance of the .NET monitor by the -u command line option.

Parameters

qmgr
The name of the queue manager that hosts the application queue.

queue
The name of the application queue to monitor.

message
A message from the monitored queue.

param
User defined data. User data must be comprised of ASCII characters only, with no quotation marks (″), null characters, or carriage returns.

MQC

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

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

The MQC structure defines all the constants used by the MQI. 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
 ...
```

For a full description of these constants, see *WebSphere MQ Constants*.

Notices

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.

GC34-6949-01

Spine information:

IBM WebSphere MQ Using .NET Version 7.0