WebSphere MQ

IBM

# Using the Component Object Model Interface

*Version 7.0*

WebSphere MQ

**IBM**

# Using the Component Object Model Interface

*Version 7.0*

**Second edition (January 2009)**

This edition of the book applies to the following:

- IBM WebSphere MQ for Windows, Version 7.0

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

# Contents

**Chapter 7. About the WebSphere MQ Automation Classes for ActiveX**

# Figures

**vii**

WebSphere MQ: Using the Component Object Model Interface

# Chapter 1. Introduction

This chapter gives an overview of the WebSphere® MQ Automation Classes for ActiveX, the Component Object Model (COM) interface, and ActiveX scripting. The supported WebSphere MQ environment is also described.

## WebSphere MQ Automation Classes for ActiveX overview

The WebSphere MQ Automation Classes for ActiveX (MQAX) are ActiveX components that provide classes that you can use in your application to access WebSphere MQ. It requires a WebSphere MQ environment and a corresponding WebSphere MQ application with which to communicate.

It gives your ActiveX application the ability to run transactions and access data on any of your enterprise systems that you can access through WebSphere MQ.

WebSphere MQ Automation Classes for ActiveX:
- Give you access to the functions and features of the WebSphere MQ API, permitting full interconnectivity to other WebSphere MQ platforms.
- Conform to the normal conventions expected of an ActiveX component.
- Conform to the WebSphere MQ object model, also available for .NET, C++, Java™, and LotusScript®.

MQAX starter samples are provided. You are recommended to use these initially to check that your installation of the MQAX is successful and that you have the basic WebSphere MQ environment in place. The samples also demonstrate how MQAX can be used.

### COM and ActiveX scripting

The Component Object Model (COM) is an object-based programming model defined by Microsoft®. It specifies how software components can be provided in a way that allows them to locate and communicate with each other irrespective of the computer language in which they are written or their location.

ActiveX is a set of technologies, based on COM, that integrates application development, reusable components, and Internet technologies on the Microsoft Windows® platforms. ActiveX components provide interfaces that can be accessed dynamically by applications. An ActiveX scripting client is an application, for example a compiler, that can build or execute a program or script that uses the interfaces provided by ActiveX (or COM) components.

### WebSphere MQ environment support

WebSphere MQ Automation Classes for ActiveX can only be used with **32-bit** ActiveX scripting clients.

The COM component can only be used for **32-bit** applications. If you want to write 64-bit COM application, you can use the .NET interface.

**1**

To run the MQAX in a WebSphere MQ server environment you must have Windows 2000 or later installed on your system.

To run the MQAX in a WebSphere MQ client environment you need the following installed on your system:

- WebSphere MQ client on Windows 2000 or later

The WebSphere MQ client requires access to at least one WebSphere MQ server. When both the WebSphere MQ server and client are installed on your system MQAX applications will always run against the server. The ActiveX interface to the MQAI and Active Directory Services is only available in WebSphere MQ server environments.

# Chapter 2. Designing and programming using WebSphere MQ Automation Classes for ActiveX

This chapter includes:
- Designing MQAX applications that access non-ActiveX applications.
- "Programming hints and tips" on page 4
- "Using data conversion" on page 5
- "Threading" on page 7
- "Error handling" on page 8

## Designing MQAX applications that access non-ActiveX applications

The WebSphere MQ Automation Classes provide access to the functions of the WebSphere MQ API, so you can benefit from all the advantages that using WebSphere MQ can bring to your Windows application. The overall design of your application will be the same as for any WebSphere MQ application, so you should first consider all of the design aspects described in the WebSphere MQ Application Programming Guide.

To use the WebSphere MQ Automation Classes, you code the Windows programs in your application using a language that supports the creation and use of COM objects; for example, Visual Basic, Java, and other ActiveX scripting clients. The classes can then be easily integrated into your application because the WebSphere MQ objects you need can be coded using the native syntax of the implementation language.

### Using WebSphere MQ Automation Classes for ActiveX

When designing an ActiveX application that uses WebSphere MQ Automation Classes for ActiveX, the most important item of information is the message that is sent or received from the remote WebSphere MQ system. Therefore you must know the format of the items that are inserted into the message. For an MQAX script to a work, both it and the WebSphere MQ application that picks up or sends the message must know the message structure.

If you are sending a message with an MQAX application and you want to perform data conversion at the MQAX end, you must also know:
- The code page used by the remote system
- The encoding used by the remote system

To help you keep your code portable it is always good practice to set the code page and encoding, even if these are currently the same in both the sending and receiving systems.

When considering how to structure the implementation of the system you design, remember that your MQAX scripts run on the same machine as the one on which you have either the WebSphere MQ queue manager or the WebSphere MQ client installed.

# Programming hints and tips

The following hints and tips are in no significant order. They are subjects that, if relevant to the work you are doing, might save you time.

## Message Descriptor properties

Where a WebSphere MQ application is the originator of a message and WebSphere MQ generates the

- AccountingToken
- CorrelationId
- GroupId
- MessageId

you are recommended to use the AccountingTokenHex, CorrelationIdHex, GroupIdHex, and MessageIdHex properties if you want to look at their values, or manipulate them in any way - including passing them back in a message to WebSphere MQ. The reason for this is that WebSphere MQ generated values are strings of bytes that have any value from 0 through 255 inclusive, they are not strings of printable characters.

Where your MQAX script is the originator of a message and you generate the

- AccountingToken
- CorrelationId
- GroupId
- MessageId

you can use either the AccountingToken, CorrelationId, GroupId, or MessageId properties or their Hex equivalents.

## WebSphere MQ constants

WebSphere MQ constants are provided as members of the enum MQ in library MQAX200.

## WebSphere MQ string constants

WebSphere MQ string constants and their corresponding character strings.

WebSphere MQ string constants are not available when using WebSphere MQ Automation Classes for ActiveX. You must use the explicit character string for those shown below and any others you may need. The commands must be padded to eight characters using spaces:

| | |
|---|---|
| MQFMT_NONE | " " |
| MQFMT_ADMIN | "MQADMIN " |
| MQFMT_CHANNEL_COMPLETED | "MQCHCOM " |
| MQFMT_CICS | "MQCICS " |
| MQFMT_COMMAND_1 | "MQCMD1 " |
| MQFMT_COMMAND_2 | "MQCMD2 " |
| MQFMT_DEAD_LETTER_HEADER | "MQDEAD " |
| MQFMT_DIST_HEADER | "MQHDIST " |
| MQFMT_EVENT | "MQEVENT " |

| | |
|---|---|
| MQFMT_IMS | ″MQIMS ″ |
| MQFMT_IMS_VAR_STRIsNG | ″MQIMSVS ″ |
| MQFMT_MD_EXTENSION | ″MQHMDE ″ |
| MQFMT_PCF | ″MQPCF ″ |
| MQFMT_REF_MSG_HEADER | ″MQHREF ″ |
| MQFMT_RF_HEADER | ″MQHRF ″ |
| MQFMT_STRING | ″MQSTR ″ |
| MQFMT_TRIGGER | ″MQTRIG ″ |
| MQFMT_WORK_INFO_HEADER | ″MQHWIH ″ |
| MQFMT_XMIT_Q_HEADER | ″MQXMIT ″ |

## Null string constants

The WebSphere MQ constants, used for the initialization of four MQMessage properties, MQMI_NONE (24 NULL characters), MQCI_NONE (24 NULL characters), MQGI_NONE (24 NULL characters), and MQACT_NONE (32 NULL characters), are not supported by WebSphere MQ Automation Classes for ActiveX. Setting them to empty strings has the same effect.

For example, to set the various ids of an MQMessage to these values: *mymessage*.**MessageId** = ″″ *mymessage*.**CorrelationId** = ″″ *mymessage*.**AccountingToken** = ″″

## Receiving a message from WebSphere MQ

There are several ways of receiving a message from WebSphere MQ:
- Polling by issuing a GET followed by a Wait, using the Visual Basic TIMER function.
- Issuing a GET with the Wait option; you specify the wait duration by setting the WaitInterval property. This is recommended when, even though you set your system up to run in multithreaded environment, the software running at the time may run only singlethreaded. This avoids your system locking up indefinitely.

  Other threads operate unaffected. However, if your other threads require access to WebSphere MQ, they require a second connection to WebSphere MQ using additional MQAX queue manager and queue objects.

Issuing a GET with the Wait option and setting the WaitInterval to MQWI_UNLIMITED causes your system to lock up until the GET call completes, if the process is singlethreaded.

## Using data conversion

Two forms of data conversion are supported by WebSphere MQ Automation Classes for ActiveX.

## Numeric encoding

If you set the MQMessage Encoding property, the following methods convert between different numeric encoding systems:
- ReadDecimal2 method
- ReadDecimal4 method

- ReadDouble method
- ReadDouble4 method
- ReadFloat method
- ReadInt2 method
- ReadInt4 method
- ReadLong method
- ReadShort method
- ReadUInt2 method
- WriteDecimal2 method
- WriteDecimal4 method
- WriteDouble method
- WriteDouble4 method
- WriteFloat method
- WriteInt2 method
- WriteInt4 method
- WriteLong method
- WriteShort method
- WriteUInt2 method

The Encoding property can be set and interpreted using the supplied WebSphere MQ constants.

Figure 1 shows an example of these:

```
/* Encodings for Binary Integers */
        MQENC_INTEGER_UNDEFINED
        MQENC_INTEGER_NORMAL
        MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
        MQENC_DECIMAL_UNDEFINED
        MQENC_DECIMAL_NORMAL
        MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
        MQENC_FLOAT_UNDEFINED
        MQENC_FLOAT_IEEE_NORMAL
        MQENC_FLOAT_IEEE_REVERSED
        MQENC_FLOAT_S390
```

*Figure 1. Supplied WebSphere MQ constants for encoding*

For example, to send an integer from an Intel® system to a System/390® operating system in System/390 encoding:

```
Dim msg As New MQMessage    'Define a WebSphere MQ message for our use..
Print msg.Encoding    'Currently 546 (or X'222')
                                        'Set the encoding property
                                        to 785 (or X'311')
msg.Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                      OR MQENC_FLOAT_S390
Print msg.Encoding    'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                        'Set it
msg.WriteLong(local_num)        'Write the number into the message
```

# Character set conversion

Character set conversion is necessary when you send a message from one system to another system where the code pages are different. Code page conversion is used by:

- ReadString method
- ReadNullTerminatedString method
- WriteString method
- WriteNullTerminatedString method
- MessageData Property

You must set the MQMessage CharacterSet property to a supported character set value (CCSID).

WebSphere MQ Automation Classes for ActiveX uses conversion tables to perform character set conversion.

For example, to convert strings automatically to code page 437:

```
Dim msg As New MQMessage                'Define a WebSphere MQ message
msg.CharacterSet = 437                  'Set code page required
msg.WriteString "A character string"'Put character string in message
```

The WriteString method receives the string data (″A character string″ in the example above) as a Unicode string. It then converts this data from Unicode into code page 437 using the conversion table 34B001B5.TBL.

Characters in the Unicode string that are not supported by code page 437 are given the standard substitution character from code page 437.

In a similar manner, when you use the ReadString method, the incoming message has a character set established by the WebSphere MQ Message Descriptor (MQMD) value and there is a conversion from this code page into Unicode before it is passed back to your scripting language.

# Threading

WebSphere MQ Automation Classes for ActiveX implement a free-threading model where objects can be used between threads.

While MQAX permits the use of MQQueue and MQQueueManager objects, WebSphere MQ does not currently permit the sharing of handles between different threads.

Attempts to use these on another thread result in an error and WebSphere MQ returns a return code of MQRC_HCONN_ERROR.

**Note:** There is only one MQSession object per process. Using the MQSession CompletionCode and ReasonCode is not recommended in multithreaded environments. The MQSession error values may be overwritten by a second thread between an error being raised and checked on the first thread. Threads are serialized for the duration of each method call or property access. So, issuing a Get with the Wait option will cause other threads accessing MQAX objects to be suspended until the operation completes.

# Error handling

Each MQAX object includes properties to hold error information and a method to reset or clear them. The properties are:

- CompletionCode
- ReasonCode
- ReasonName

The method is:

- ClearErrorCodes

## How error handling works

Your MQAX script or application invokes an MQAX object's method, or accesses or updates a property of the MQAX object:

1. The ReasonCode and CompletionCode in the object concerned are updated.
2. The ReasonCode and CompletionCode in the MQSession object are also updated with the same information.

   **Note:** See "Threading" on page 7 for restrictions on the use of MQSession error codes in threaded applications.

   If the CompletionCode is greater than or equal to the ExceptionThreshold property of MQSession, MQAX throws an exception (number 32000). Use this within your script using the On Error (or equivalent) statement to process it.
3. Use the Error function to retrieve the associated error string, which will have the form:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   For more information on how to use the On Error statements, see the documentation for your ActiveX scripting language.

   Using the CompletionCode and ReasonCode in the MQSession object is very convenient for simple error handlers.

   ReasonName property returns the WebSphere MQ symbolic name for the current value of the ReasonCode.

### Raising exceptions

The following rules describe how raising exceptions are handled:

- Whenever a property or method sets the completion code to a value greater than or equal to the exception threshold (usually set to 2) an exception is raised.
- All method calls and property sets set the completion code.

## Getting a property

This is a special case because the CompletionCode and ReasonCode are not always updated:

- If a property get succeeds, the object and MQSession object ReasonCode and CompletionCode remain unchanged.
- If a property get fails with a CompletionCode of warning, the ReasonCode and CompletionCode remain unchanged.

- If a property get fails with a CompletionCode of error, the ReasonCode and CompletionCode are updated to reflect the true values, and error processing proceeds as described.

The MQSession class has a method *ReasonCodeName* which may be used to replace a WebSphere MQ reason code with a symbolic name. This is especially useful while developing programs where unexpected errors may occur. However, the name is not ideal for presentation to end users.

Each class also has a property *ReasonName*, that returns the symbolic name of the current reason code for that class.

# Chapter 3. WebSphere MQ Automation Classes for ActiveX reference

This chapter describes the classes of the WebSphere MQ Automation Classes for ActiveX (MQAX), developed for ActiveX. The classes enable you to write ActiveX applications that can access other applications running in your non-ActiveX environments, using WebSphere MQ.

## WebSphere MQ Automation Classes for ActiveX interface

The ActiveX automation classes consist of the following:
- "MQSession Class" on page 13
- "MQQueueManager class" on page 16
- "MQQueue class" on page 29
- "MQMessage class" on page 47
- "MQPutMessageOptions class" on page 74
- "MQGetMessageOptions class" on page 76
- "MQDistributionList class" on page 79
- "MQDistributionListItem class" on page 84

In addition, WebSphere MQ Automation Classes for ActiveX provides predefined numeric ActiveX constants (such as MQMT_REQUEST) needed to use the classes. These are provided in the enum MQ in library MQAX200. The constants are a subset of those defined in the WebSphere MQ C header files (cmqc*.h) with some additional WebSphere MQ Automation Classes for ActiveX Reason codes.

## About WebSphere MQ Automation Classes for ActiveX classes

This information should be read in conjunction with the WebSphere MQ Application Programming Referencemanual.

The MQSession class provides a root object that contains the status of the last action performed on any of the MQAX objects. See "Error handling" on page 8 for more information.

The MQQueueManager and MQQueue classes provide access to the underlying WebSphere MQ objects. Methods or property accesses for these classes in general result in calls being made across the WebSphere MQ MQI.

The MQMessage, MQPutMessageOptions, and MQGetMessageOptions classes encapsulate the MQMD, MQPMO, and MQGMO data structures respectively, and are used to help you send messages to queues and retrieve messages from them.

The MQDistributionList class encapsulates a collection of queues - local, remote, or alias for output. The MQDistributionListItem class encapsulates the MQOR, MQRR, and MQPMR structures and associates them with an owning distribution list.

These classes can be found in the MQAX200 library.

# Parameter passing

Parameters on method invocations are all passed by value, except where that parameter is an object, in which case it is a reference that is passed.

The class definitions provided list the Data Type for each parameter or property. For many ActiveX clients, such as Visual Basic, if the variable used is not of the required type, the value is automatically converted to or from the required type - providing such a conversion is possible. This follows standard rules of the client; MQAX provides no such conversion.

Many of the methods take fixed-length string parameters, or return a fixed-length character string. The conversion rules are as follows:

- If the user supplies a fixed-length string of the wrong length, as an input parameter or as a return value, the value is truncated or padded with trailing spaces as required.
- If the user supplies a variable-length string of the wrong length as an input parameter, the value is truncated or padded with trailing spaces.
- If the user supplies a variable-length string of the wrong length as a return value, the string is adjusted to the required length (because returning a value destroys the previous value in the string anyway).
- Strings provided as input parameters may contain embedded Nulls.

## Object access methods

These methods do not relate directly to any single WebSphere MQ call. Each of these methods creates an object in which reference information is then held, followed by connecting to or opening a WebSphere MQ object:

When a connection is made to a queue manager, it holds the 'connection handle' generated by WebSphere MQ.

When a queue is opened, it holds the 'object handle' generated by WebSphere MQ.

These WebSphere MQ attributes are explained in the WebSphere MQ Application Programming Reference manual. They are not directly available to the MQAX program.

## Errors

Syntactic errors on parameter passing may be detected at compile time and run time by the ActiveX client. Errors can be trapped using On Error in Visual Basic.

The WebSphere MQ ActiveX classes all contain two special read-only properties - ReasonCode and CompletionCode. These can be read at any time.

An attempt to access any other property, or to issue any method call could potentially generate an error from WebSphere MQ.

If a property set or method invocation succeeds, the owning object's ReasonCode and CompletionCode fields are set to MQRC_NONE and MQCC_OK respectively.

If the property access or method invocation does not succeed, reason and completion codes are set in these fields.

# MQSession Class

This is the root class for WebSphere MQ Automation Classes for ActiveX.

There is always only one MQSession object per ActiveX client process.

An attempt to create a second object creates a second reference to the original object.

## Creation

New creates a new MQSession object.

### Syntax

**Dim** *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

### Properties
- CompletionCode property
- ExceptionThreshold property
- ReasonCode property
- ReasonName property

### Methods
- AccessGetMessageOptions method
- AccessMessage method
- AccessPutMessageOptions method
- AccessQueueManager method
- ClearErrorCodes method
- ReasonCodeName method

## CompletionCode property

Read-only. Returns the WebSphere MQ completion code set by the most recent method or property set issued against any WebSphere MQ object.

It is reset to MQCC_OK when a method or a property set is invoked successfully against any MQAX object.

An error event handler can inspect this property to diagnose the error, without having to know which object was involved.

Using the CompletionCode and ReasonCode in the MQSession object is very convenient for simple error handlers.

**Note:** See "Threading" on page 7 for restrictions on the use of MQSession error codes in threaded applications.

**Defined in:** MQSession class

**Data Type:** Long

**Values:**

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:**

To get: *completioncode& = MQSession*.CompletionCode

## ExceptionThreshold property

Read-write. Defines the level of WebSphere MQ error for which MQAX will throw an exception. Defaults to MQCC_FAILED. A value greater than MQCC_FAILED effectively prevents exception processing, leaving the programmer to perform checks on the CompletionCode and ReasonCode.

**Defined in:** MQSession class

**Data Type:** Long

**Values:**
- Any, but only MQCC_WARNING, MQCC_FAILED or greater are recommended.

**Syntax:**

To get: *ExceptionThreshold& = MQSession*.**ExceptionThreshold**

To set: *MQSession*.**ExceptionThreshold** = *ExceptionThreshold$*

## ReasonCode property

Read-only. Returns the reason code set by the most recent method or property set issued against any WebSphere MQ object.

An error event handler can inspect this property to diagnose the error, without having to know which object was involved.

Using the CompletionCode and ReasonCode in the MQSession object is very convenient for simple error handlers.

**Note:** See "Threading" on page 7 for restrictions on the use of MQSession error codes in threaded applications.

**Defined in:** MQSession class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference and the additional MQAX values listed under "Reason codes" on page 98

**Syntax:** To get: *reasoncode& = MQSession*.**ReasonCode**

# ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example, "MQRC_QMGR_NOT_AVAILABLE".

**Note:** See "Threading" on page 7 for restrictions on the use of MQSession error codes in threaded applications.

**Defined in:** MQSession class

**Data Type:** String

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQSession*.**ReasonName**

# AccessGetMessageOptions method

Creates a new MQGetMessageOptions object.

**Defined in:** MQSession class

**Syntax:** *gmo = MQSession*.**AccessGetMessageOptions()**

# AccessMessage method

Creates a new MQMessage object.

**Defined in:** MQSession class

**Syntax:** *msg = MQSession*.**AccessMessage()**

# AccessPutMessageOptions method

Creates a new MQPutMessageOptions object.

**Defined in:** MQSession class

**Syntax:** *pmo = MQSession*.**AccessPutMessageOptions()**

# AccessQueueManager method

Creates a new MQQueueManager object and connects it to a real queue manager by means of the WebSphere MQ client or server. As well as performing a connect, this method also performs an open for the queue manager object.

**Note:** When both the WebSphere MQ server and client are installed on your system MQAX applications will run against the server by default. To run MQAX against the client, the `mqic32.dll` path and filename must be specified in the `GMQ_MQ_LIB` environment variable, for example, set `GMQ_MQ_LIB=mqic32.dll`

If successful it sets the MQQueueManager's ConnectionStatus to TRUE.

A queue manager can be connected to at most one MQQueueManager object per ActiveX instance.

If the connection to the queue manager fails, an error event is raised, and the MQSession object's ReasonCode and CompletionCode are set.

**Defined in:** MQSession class

**Syntax:** *set qm = MQSession*.**AccessQueueManager** (*Name$*)

**Parameter:***Name$* String. Name of Queue Manager to be connected to.

# ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE.

**Defined in:** MQSession class

**Syntax:** Call *MQSession*.**ClearErrorCodes()**

# ReasonCodeName method

Returns the name of the reason code with the given numeric value. It is useful to give clearer indications of error conditions to users. The name is still somewhat cryptic (for example, ReasonCodeName(2059) is **MQRC_Q_MGR_NOT_AVAILABLE**), so where possible errors should be caught and replaced with descriptive text appropriate to the application.

**Defined in:** MQSession class

**Syntax:** *errname$ = MQSession*.**ReasonCodeName**(*reasonCode&*)

**Parameter:***reasoncode&* Long. The reason code for which the symbolic name is required.

# MQQueueManager class

This class represents a connection to a queue manager. The queue manager may be running locally (a WebSphere MQ server) or remotely with access provided by the WebSphere MQ client. An application must create an object of this class and connect it to a queue manager. When an object of this class is destroyed it is automatically disconnected from its queue manager.

## Containment

MQQueue class objects are associated with this class.

### Creation

**New** creates a new MQQueueManager object and sets all the properties to initial values. Alternatively use the AccessQueueManager method of the MQSession class.

### Syntax

**Dim** *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

## Properties

- AlternateUserId property
- AuthorityEvent property
- BeginOptions property
- ChannelAutoDefinition property
- ChannelAutoDefinitionEvent property
- ChannelAutoDefinitionExit property
- CharacterSet property
- CloseOptions property
- CommandInputQueueName property
- CommandLevel property
- CompletionCode property
- ConnectionHandle property
- ConnectionStatus property
- ConnectOptions property
- DeadLetterQueueName property
- DefaultTransmissionQueueName property
- Description property
- DistributionLists property
- InhibitEvent property
- IsConnected property
- IsOpen property
- LocalEvent property
- MaximumHandles property
- MaximumMessageLength property
- MaximumPriority property
- MaximumUncommittedMessages property
- Name property
- ObjectHandle property
- PerformanceEvent property
- Platform property
- ReasonCode property
- ReasonName property
- RemoteEvent property
- StartStopEvent property
- SyncPointAvailability property
- TriggerInterval property

## Methods

- AccessQueue method
- AddDistributionList method
- Backout method
- Begin method
- ClearErrorCodes method
- Commit method

- Connect method
- Disconnect method

## Property Access

The following properties can be accessed at any time
- AlternateUserId
- CompletionCode
- ConnectionStatus
- ReasonCode

The remaining properties can be accessed only if the object is connected to a queue manager, and the user ID is authorized to inquire against that queue manager. If an alternate user ID is set and the current user ID is authorized to use it, the alternate user ID is checked for authorization for inquire instead.

If these conditions do not apply, WebSphere MQ Automation Classes for ActiveX attempts to connect to the queue manager and open it for inquire automatically. If this is unsuccessful, the call sets a CompletionCode of MQCC_FAILED and one of the following ReasonCodes:
- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

The Backout, Begin, Commit, Connect, and Disconnect methods set errors matching those set by the MQI calls MQBACK, MQBEGIN, MQCMIT, MQCONN, and MQDISC respectively.

# AlternateUserId property

Read-write. The alternate user ID to be used to validate access to the queue manager attributes.

This property should not be set if IsConnected is TRUE.

This property cannot be set whilst the object is open.

**Defined in:** MQQueueManager class

**Data Type:** String of 12 characters

**Syntax:** To get: *altuser$ = MQQueueManager*.**AlternateUserId** To set: *MQQueueManager*.**AlternateUserId** = *altuser$*

# AuthorityEvent property

Read-only. The MQI AuthorityEvent attribute.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *authevent = MQQueueManager*.**AuthorityEvent**

## BeginOptions property

Read-write. These are the options that apply to the Begin method. Initially
MQBO_NONE.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQBO_NONE

**Syntax:** To get: *beginoptions&=MQQueueManager*.**BeginOptions**

To set: *MQQueueManager*.**BeginOptions**=*beginoptions&*

## ChannelAutoDefinition property

Read-only. This controls whether automatic channel definition is permitted.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQCHAD_DISABLED
- MQCHAD_ENABLED

**Syntax:** To get: *channelautodef&= MQQueueManager*.**ChannelAutoDefinition**

## ChannelAutoDefinitionEvent property

Read-only. This controls whether automatic channel definition events are
generated.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *channelautodefevent
&=MQQueueManager*.**ChannelAutoDefinitionEvent**

## ChannelAutoDefinitionExit property

Read-only. The name of the user exit used for automatic channel definition.

**Defined in:**
    MQQueueManager class

**Data Type:**
    String

**Syntax:** To get: *channelautodefexit$= MQQueueManager*.**ChannelAutoDefinitionExit**

## CharacterSet property

Read-only. The MQI CodedCharSetId attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *characterset&= MQQueueManager*.**CharacterSet**

## CloseOptions property

Read-write. Options used to control what happens when the queue manger is closed. The initial value is MQCO_NONE.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQCO_NONE

**Syntax:** To get: *closeopt& = MQQueueManager*.**CloseOptions**

To set: *MQQueueManager*.**CloseOptions** *=closeopt&*

## CommandInputQueueName property

Read-only. The MQI CommandInputQName attribute.

**Defined in:** MQQueueManager class

**Data Type:** String of 48 characters

**Syntax:** To get: *commandinputqname$ = MQQueueManager*.**CommandInputQueueName**

## CommandLevel property

Read-only. Returns the version and level of the WebSphere MQ queue manager implementation (MQI CommandLevel attribute)

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *level& = MQQueueManager*.**CommandLevel**

## CompletionCode property

Read-only. Returns the completion code set by the last method or property access issued against the object.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode& = MQQueueManager*.**CompletionCode**

## ConnectionHandle property

Read-only. The connection handle for the WebSphere MQ queue manager object.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Syntax:** To get: *hconn& = MQQueueManager.***ConnectionHandle**

## ConnectionStatus property

Read-only. Indicates if the object is connected to its queue manager or not.

**Defined in:** MQQueueManager class

**Data Type:** Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *status = MQQueueManager*.**ConnectionStatus**

## ConnectOptions property

Read-Write. These options apply to the Connect method. Initially MQCNO_NONE.

**Defined in:**
    MQQueueManager class

**Data Type:**
    Long

**Values:**

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

**Syntax:** To get: *connectoptions&=MQQueueManager*.**ConnectOptions**

To set: *MQQueueManager*.**ConnectOptions**=*connectoptions&*

# DeadLetterQueueName property

Read-only. The MQI DeadLetterQName attribute.

**Defined in:** MQQueueManager class

**Data Type:** String of 48 characters

**Syntax:** to get: *dlqname$ = MQQueueManager*.**DeadLetterQueueName**

# DefaultTransmissionQueueName property

Read-only. The MQI DefXmitQName attribute.

**Defined in:** MQQueueManager class

**Data Type:** String of 48 characters

**Syntax:** To get: *defxmitqname$ = MQQueueManager*.**DefaultTransmissionQueueName**

# Description property

Read-only. The MQI QMgrDesc attribute.

**Defined in:** MQQueueManager class

**Data Type:** String of 64 characters

**Syntax:** To get: *description$ = MQQueueManager*.**Description**

# DistributionLists property

Read-Only. This is the capability of the queue manager to support distribution lists.

**Defined in:**
MQQueueManager class

**Data Type:**
Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *distributionlists= MQQueueManager*.**DistributionLists**

## InhibitEvent property

Read-only. The MQI InhibitEvent attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *inhibevent& = MQQueueManager*.**InhibitEvent**

## IsConnected property

Read-only. A value that indicates whether or not the queue manager is currently connected.

**Defined in:** MQQueueManager class

**Data Type:** Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *isconnected = MQQueueManager*.**IsConnected**

## IsOpen property

Read-only. A value that indicates whether or not the queue manager is currently open for inquire.

**Defined in:**
MQQueueManager class

**Data Type:**
Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *IsOpen = MQQueueManager*.**IsOpen**

## LocalEvent property

Read-only. The MQI LocalEvent attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**

- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *localevent& = MQQueueManager*.**LocalEvent**

## MaximumHandles property

Read-only. The MQI MaxHandles attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *maxhandles& = MQQueueManager*.**MaximumHandles**

## MaximumMessageLength property

Read-only. The MQI MaxMsgLength Queue Manager attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *maxmessagelength& = MQQueueManager*.**MaximumMessageLength**

## MaximumPriority property

Read-only. The MQI MaxPriority attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *maxpriority& = MQQueueManager*.**MaximumPriority**

## MaximumUncommittedMessages property

Read-only. The MQI MaxUncommittedMsgs attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *maxuncommitted& = MQQueueManager*.**MaximumUncommittedMessages**

## Name property

Read-write. The MQI QMgrName attribute. This property cannot be written once the MQQueueManager is connected.

**Defined in:** MQQueueManager class

**Data Type:** String of 48 characters

**Syntax:** To get: *name$ = MQQueueManager*.**name**

To set: *MQQueueManager*.**name** = *name$*

**Note:** Visual Basic reserves the "Name" property for use in the visual interface. Therefore, when using within Visual Basic use lower-case, that is, "name".

# ObjectHandle property

Read-only. The object handle for the WebSphere MQ queue manager object.

**Defined in:**
MQQueueManager class

**Data type**
Long

**Syntax:** To get: *hobj& = MQQueueManager.***ObjectHandle**

# PerformanceEvent property

Read-only. The MQI PerformanceEvent attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *perfevent& = MQQueueManager*.PerformanceEvent

# Platform property

Read-only. The MQI Platform attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQPL_WINDOWS_NT
- MQPL_WINDOWS

**Syntax:** To get: *platform& = MQQueueManager*.**Platform**

# ReasonCode property

Read-only. Returns the reason code set by the last method or property access issued against the object.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = MQQueueManager*.**ReasonCode**

## ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example, "MQRC_QMGR_NOT_AVAILABLE".

**Defined in:** MQQueueManager class

**Data Type:** String

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQQueueManager*.**ReasonName**

## RemoteEvent property

Read-only. The MQI RemoteEvent attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *remoteevent& = MQQueueManager*.**RemoteEvent**

## StartStopEvent property

Read-only. The MQI StartStopEvent attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *strstpevent& = MQQueueManager*.**StartStopEvent**

## SyncPointAvailability property

Read-only. The MQI SyncPoint attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Values:**
- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

**Syntax:** To get: *syncpointavailability& = MQQueueManager*.**SyncPointAvailability**

# TriggerInterval property

Read-only. The MQI TriggerInterval attribute.

**Defined in:** MQQueueManager class

**Data Type:** Long

**Syntax:** To get: *trigint& = MQQueueManager*.**TriggerInterval**

# AccessQueue method

Creates a new MQQueue object and associates it with this MQQueueManager object by setting the queue's connection reference property. It sets the Name, OpenOptions, DynamicQueueName, and AlternateUserId properties of the MQQueue object to the values provided, and attempts to open it.

If the open is unsuccessful the call fails. An error event is raised against the object, the object's ReasonCode and CompletionCode are set, and the MQSession ReasonCode and CompletionCode are set.

The DynamicQueueName, QueueManagerName, and AlternateUserId parameters are optional and default to "".

The OpenOption MQOO_INQUIRE should be specified in addition to other options if queue properties are to be read.

Do not set the QueueManagerName, or set it to "" if the queue to be opened is local. Otherwise, it should be set to the name of the remote queue manager that owns the queue, and an attempt is made to open a local definition of the remote queue. See the WebSphere MQ Application Programming Guide for more information on remote queue name resolution and queue manager aliasing.

If the Name property is set to a model queue name, specify the name of the dynamic queue to be created in the DynamicQueueName$ parameter. If the value provided in the DynamicQueueName$ parameter is "", the value set into the queue object and used on the open call is "AMQ.*". See the WebSphere MQ Application Programming Guide for more information on naming dynamic queues.

## Defined in:

MQQueueManager class.

## Syntax:

set *queue = MQQueueManager*.**AccessQueue**(*Name$, OpenOptions &,QueueManagerName$, DynamicQueueName$, AlternateUserId$*)

### Parameters

*Name$* String. Name of WebSphere MQ queue.

*OpenOptions:* Long. Options to be used when queue is opened. See the WebSphere MQ Application Programming Reference.

*QueueManagerName$* String. Name of the queue manager that owns the queue to be opened. A value of ″″ implies the queue manager is local.

*DynamicQueueName$* String. The name assigned to the dynamic queue at the time the queue is opened when the Name$ parameter specifies a model queue.

*AlternateUserId$* String. The alternate user ID used to validate access when opening the queue.

## AddDistributionList method

Creates a new MQDistributionList object and sets its connection reference to the owning queue manager.

**Defined in:**
    MQQueueManager class

**Syntax:** *set distributionlist* = **MQQueueManager.***AddDistributionList*

## Backout method

Backs out any uncommitted message puts and gets that have occurred as part of a unit of work since the last syncpoint.

**Defined in:** MQQueueManager class

**Syntax:** Call *MQQueueManager.***Backout()**

## Begin method

Begins a unit of work that is coordinated by the queue manager. The begin options affect the behavior of this method.

**Defined in:**
    MQQueueManager class

**Syntax:** Call *MQQueueManager*.**Begin()**

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQQueueManager class and the MQSession class.

**Defined in:** MQQueueManager class

**Syntax: Call** *MQQueueManager*.**ClearErrorCodes()**

# Commit method

Commits any message puts and gets that have occurred as part of a unit of work since the last syncpoint.

**Defined in:** MQQueueManager class

**Syntax:** Call *MQQueueManager*.**Commit()**

# Connect method

Connects the MQQueueManager object to a real queue manager via the WebSphere MQ client or server. As well as performing a connect, this method also performs an open for queries to be made against the queue manager object.

Sets IsConnected to TRUE.

A maximum of one MQQueueManager object per ActiveX instance is allowed to connect to a queue manager.

**Defined in:** MQQueueManager class

**Syntax:** Call *MQQueueManager*.**Connect()**

# Disconnect method

Disconnects the MQQueueManager object from the queue manager.

Sets IsConnected to FALSE.

All Queue objects associated with the MQQueueManager object are made unusable and cannot be re-opened.

Any uncommitted changes (message puts and gets) are committed.

**Defined in:** MQQueueManager class

**Syntax:** Call *MQQueueManager*.**Disconnect()**

# MQQueue class

This represents access to a WebSphere MQ queue. This connection is provided by an associated MQQueueManager object. When an object of this class is destroyed it is automatically closed.

# Containment

Contained by the MQQueueManager class.

### Creation

**New** creates a new MQQueue object and sets all the properties to initial values. Alternatively, use the AccessQueue Method of the MQQueueManager class.

### Syntax

**Dim** *que As* **New MQQueue Set** *que* = **New MQQueue**

### Properties

- AlternateUserId property
- BackoutRequeueName property
- BackoutThreshold property
- BaseQueueName property
- CloseOptions property
- CompletionCode property
- ConnectionReference property
- CreationDateTime property
- CurrentDepth property
- DefaultInputOpenOption property
- DefaultPersistence property
- DefaultPriority property
- DefinitionType property
- DepthHighEvent property
- DepthHighLimit property
- DepthLowEvent property
- DepthLowLimit property
- DepthMaximumEvent property
- Description property
- DynamicQueueName property
- HardenGetBackout property
- InhibitGet property
- InhibitPut property
- InitiationQueueName property
- IsOpen property
- MaximumDepth property
- MaximumMessageLength property
- MessageDeliverySequence property
- Name property
- ObjectHandle property
- OpenInputCount property
- OpenOptions property
- OpenOutputCount property
- OpenStatus property
- ProcessName property
- QueueManagerName property
- QueueType property
- ReasonCode property
- ReasonName property
- RemoteQueueManagerName property
- RemoteQueueName property

- ResolvedQueueManagerName property
- ResolvedQueueName property
- RetentionInterval property
- Scope property
- ServiceInterval property
- ServiceIntervalEvent property
- Shareability property
- TransmissionQueueName property
- TriggerControl property
- TriggerData property
- TriggerDepth property
- TriggerMessagePriority property
- TriggerType property
- Usage property

## Methods
- ClearErrorCodes method
- Close method
- Get method
- Open method
- Put method

## Property Access

If the queue object is **not** connected to a queue manager, you can read the following properties:
- CompletionCode
- OpenStatus
- ReasonCode

and you can read and write to:
- AlternateUserId
- CloseOptions
- ConnectionReference
- Name
- OpenOptions

If the queue object **is** connected to a queue manager, you can read all the properties.

## Queue Attribute properties

Properties not listed in the previous section are all attributes of the underlying WebSphere MQ queue. They can be accessed only if the object is connected to a queue manager, and the user's user ID is authorized for Inquire or Set against that queue. If an alternate user ID is set and the current user ID is authorized to use it, the alternate user ID is checked for authorisation instead.

The property must be an appropriate property for the given QueueType. See the *WebSphere MQ Application Programming Reference* manual.

If these conditions do not apply, the property access will set a CompletionCode of MQCC_FAILED and one of the following ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode is MQCC_WARNING)

### Opening a queue

The only way to create an MQQueue object is by using the MQQueueManager AccessQueue method or by New. An open MQQueue object remains open (OpenStatus=TRUE) until it is closed or deleted or until the creating queue manager object is deleted or connection is lost to the queue manager. The value of the MQQueue CloseOptions property controls the behavior of the close operation that takes place when the MQQueue object is deleted.

The MQQueueManager AccessQueue method opens the queue using the OpenOptions parameter. The MQQueue.Open method opens the queue using the OpenOptions property. WebSphere MQ validates the OpenOptions against the user authorization as part of the open queue process.

## AlternateUserId property

Read-write. The alternate user ID used to validate access to the queue when it is opened.

This property cannot be set while the object is open (that is, when IsOpen is TRUE).

**Defined in:** MQQueue class

**Data Type:** String of 12 characters

**Syntax:** To get: *altuser$ = MQQueue*.**AlternateUserId**

To set: *MQQueue.***AlternateUserId** = *altuser$*

## BackoutRequeueName property

Read-only. The MQI BackOutRequeueQName attribute.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *backoutrequeuename$ = MQQueue*.**BackoutRequeueName**

## BackoutThreshold property

Read-only. The MQI BackoutThreshold attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *backoutthreshold& = MQQueue.***BackoutThreshold**

## BaseQueueName property

Read-only. The queue name to which the alias resolves.

Valid only for alias queues.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *baseqname$ = MQQueue.***BaseQueueName**

## CloseOptions property

Read-Write. Options used to control what happens when the queue is closed.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE and MQCO_DELETE_PURGE are valid only for dynamic queues.

**Syntax:** To get: *closeopt& = MQQueue.***CloseOptions**

To set: *MQQueue.***CloseOptions** *= closeopt&*

## CompletionCode property

Read-only. Returns the completion code set by the last method or property access issued against the object.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode& = MQQueue.***CompletionCode**

## ConnectionReference property

Read-write. Defines the queue manager object to which a queue object belongs. The connection reference cannot be written while a queue is open.

**Defined in:** MQQueue class

**Data Type:** MQQueueManager

**Values:**
- A reference to an active WebSphere MQ Queue Manager object

**Syntax:** To set: *set MQQueue.***ConnectionReference** = *ConnectionReference*

To get: *set ConnectionReference = MQQueue.***ConnectionReference**

## CreationDateTime property

Read-only. Date and time this queue was created.

**Defined in:** MQQueue class

**Data Type:** Variant of type 7 (date/time) or EMPTY

**Syntax:** To get: *datetime = MQQueue.***CreationDateTime**

## CurrentDepth property

Read-only. The number of messages currently on the queue.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *currentdepth& = MQQueue.***CurrentDepth**

## DefaultInputOpenOption property

Read-only. Controls the way that the queue is opened if the OpenOptions specify MQOO_INPUT_AS_Q_DEF.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

**Syntax:** To get: *defaultinop& = MQQueue.***DefaultInputOpenOption**

## DefaultPersistence property

Read-only. The default persistence for messages on a queue.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *defpersistence& = MQQueue*.**DefaultPersistence**

## DefaultPriority property

Read-only. The default priority for messages on a queue.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *defpriority& = MQQueue*.**DefaultPriority**

## DefinitionType property

Read-only. Queue definition type.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

**Syntax:** To get: *deftype& = MQQueue*.**DefinitionType**

## DepthHighEvent property

Read-only. The MQI QDepthHighEvent attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *depthhighevent& = MQQueue*.**DepthHighEvent**

## DepthHighLimit property

Read-only. The MQI QDepthHighLimit attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *depthhighlimit& = MQQueue.***DepthHighLimit**

## DepthLowEvent property

Read-only. The MQI QDepthLowEvent attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *depthlowevent& = MQQueue.***DepthLowEvent**

## DepthLowLimit property

Read-only. The MQI QDepthLowLimit attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *depthlowlimit& = MQQueue.***DepthLowLimit**

## DepthMaximumEvent property

Read-only. The MQI QDepthMaxEvent attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQEVR_DISABLED
- MQEVR_ENABLED

**Syntax:** To get: *depthmaximumevent& = MQQueue.***DepthMaximumEvent**

## Description property

Read-only. A description of the queue.

**Defined in:** MQQueue class

**Data Type:** String of 64 characters

**Syntax:** To get: *description$ = MQQueue.***Description**

# DynamicQueueName property

Read-write, read-only when the queue is open.

This controls the dynamic queue name used when a model queue is opened. It may be set with a wildcard by the user either as a property set (only when the queue is closed) or as a parameter to MQQueueManager.AccessQueue().

The actual name of the dynamic queue is found by querying QueueName.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Values:**
• Any valid WebSphere MQ queue name.

**Syntax:** To set: *MQQueue*.**DynamicQueueName** = *dynamicqueuename$*

To get: *dynamicqueuename$* = *MQQueue*.**DynamicQueueName**

# HardenGetBackout property

Read-only. Whether to maintain an accurate back-out count.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
• MQQA_BACKOUT_HARDENED
• MQQA_BACKOUT_NOT HARDENED

**Syntax:** To get: *hardengetback&* = *MQQueue*.**HardenGetBackout**

# InhibitGet property

Read-write. The MQI InhibitGet attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
• MQQA_GET_INHIBITED
• MQQA_GET_ALLOWED

**Syntax:** To get: *getstatus&* = *MQQueue*.**InhibitGet**

To set: *MQQueue*.**InhibitGet** = *getstatus&*

## InhibitPut property

Read-write. The MQI InhibitPut attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

**Syntax:** To get: *putstatus& = MQQueue*.**InhibitPut**

To set: *MQQueue*.**InhibitPut** = *putstatus&*

## InitiationQueueName property

Read-only. Name of initiation queue.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *initqname$ = MQQueue*.**InitiationQueueName**

## IsOpen property

Read-only. Returns whether or not the queue is open.

**Defined in:** MQQueue class

**Data Type:** Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *open = MQQueue*.**IsOpen**

## MaximumDepth property

Read-only. Maximum queue depth.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *maxdepth& = MQQueue*.**MaximumDepth**

## MaximumMessageLength property

Read-only. Maximum permitted message length in bytes for this queue.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *maxmlength& = MQQueue.***MaximumMessageLength**

## MessageDeliverySequence property

Read-only. Message delivery sequence.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQMDS_PRIORITY
- MQMDS_FIFO

**Syntax:** To get: *messdelseq& = MQQueue.***MessageDeliverySequence**

## Name property

Read-write. The MQI Queue attribute. This property cannot be written after the MQQueue is open.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *name$ = MQQueue.***name**

To set: *MQQueue.***name** *= name$*

**Note:** Visual Basic reserves the "Name" property for use in the visual interface. Therefore, when using within Visual Basic use lower-case, that is "name".

## ObjectHandle property

Read-only. The object handle for the WebSphere MQ queue object.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *hobj& = MQQueue.***ObjectHandle**

## OpenInputCount property

Read-only. Number of opens for input.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *openincount& = MQQueue*.**OpenInputCount**

## OpenOptions property

Read-write. Options to be used for opening the queue.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *openopt& = MQQueue*.**OpenOptions**

To set: *MQQueue*.**OpenOptions** = *openopt&*

## OpenOutputCount property

Read-only. Number of opens for output.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *openoutcount& = MQQueue*.**OpenOutputCount**

## OpenStatus property

Read-only. Indicates if the queue is opened or not. Initial value is TRUE after AccessQueue method or FALSE after New.

**Defined in:** MQQueue class

**Data Type:** Boolean

**Values:**
- TRUE (-1)
- FALSE (0)

**Syntax:** To get: *status& = MQQueue*.**OpenStatus**

## ProcessName property

Read-only. The MQI ProcessName attribute.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *procname$ =MQQueue*.**ProcessName**

## QueueManagerName property

Read-write. The WebSphere MQ queue manager name.

**Defined in:** MQQueue class

**Data Type:** String

**Syntax:** To get: *QueueManagerName$ = MQQueue*.**QueueManagerName**

To set: *MQQueue*.**QueueManagerName** = *QueueManagerName$*

## QueueType Property

Read-only. The MQI QType attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

**Syntax:** To get: *queuetype& = MQQueue*.**QueueType**

## ReasonCode property

Read-only. Returns the reason code set by the last method or property access issued against the object.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = MQQueue*.**ReasonCode**

## ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example, "MQRC_QMGR_NOT_AVAILABLE".

**Defined in:** MQQueue class

**Data Type:** String

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQQueue*.**ReasonName**

## RemoteQueueManagerName property

Read-only. Name of remote queue manager. Valid for remote queues only.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *remqmanname$ = MQQueue*.**RemoteQueueManagerName**

## RemoteQueueName property

Read-only. The name of the queue as it is known on the remote queue manager. Valid for remote queues only.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *remqname$ = MQQueue*.**RemoteQueueName**

## ResolvedQueueManagerName property

Read-only. The name of the final destination queue manager as known to the local queue manager.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *resqmanname$ = MQQueue*.**ResolvedQueueManagerName**

## ResolvedQueueName property

Read-only. The name of the final destination queue manager as known to the local queue manager.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *resqmanname$ = MQQueue*.**ResolvedQueueName**

## RetentionInterval property

Read-only. The period of time for which the queue should be retained.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *retinterval& = MQQueue*.**RetentionInterval**

# Scope property

Read-only. Controls whether an entry for this queue also exists in a cell directory.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQSCO_Q_MGR
- MQSCO_CELL

**Syntax:** To get: *scope& = MQQueue*.**Scope**

# ServiceInterval property

Read-only. The MQI QServiceInterval attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *serviceinterval& = MQQueue.***ServiceInterval**

# ServiceIntervalEvent property

Read-only. The MQI QServiceIntervalEvent attribute.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

**Syntax:** To get: *serviceintervalevent& = MQQueue.***ServiceIntervalEvent**

# Shareability property

Read-only. Queue shareability.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

**Syntax:** To get: *shareability& = MQQueue*.**Shareability**

## TransmissionQueueName property

Read-only. Transmission queue name. Valid for remote queues only.

**Defined in:** MQQueue class

**Data Type:** String of 48 characters

**Syntax:** To get: *transqname$ = MQQueue*.**TransmissionQueueName**

## TriggerControl property

Read-write. Trigger control.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQTC_OFF
- MQTC_ON

**Syntax:** To get: *trigcontrol& = MQQueue*.**TriggerControl**

To set: *MQQueue*.**TriggerControl** = *trigcontrol&*

## TriggerData property

Read-write. Trigger data.

**Defined in:** MQQueue class

**Data Type:** String of 64 characters

**Syntax:** To get: *trigdata$ = MQQueue*.**TriggerData**

To set: *MQQueue*.**TriggerData** = *trigdata$*

## TriggerDepth property

Read-write. The number of messages that have to be on the queue before a trigger message is written.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *trigdepth& = MQQueue*.**TriggerDepth**

To set: *MQQueue*.**TriggerDepth** = *trigdepth&*

## TriggerMessagePriority property

Read-write. Threshold message priority for triggers.

**Defined in:** MQQueue class

**Data Type:** Long

**Syntax:** To get: *trigmesspriority& = MQQueue.***TriggerMessagePriority**

To set: *MQQueue.***TriggerMessagePriority** *= trigmesspriority&*

## TriggerType property

Read-write. Trigger type.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

**Syntax:** To get: *trigtype& = MQQueue.***TriggerType**

To set: *MQQueue.***TriggerType** *= Trigtype&*

## Usage property

Read-only. Indicates what the queue is used for.

**Defined in:** MQQueue class

**Data Type:** Long

**Values:**
- MQUS_NORMAL
- MQUS_TRANSMISSION

**Syntax:** To get: *usage& = MQQueue.***Usage**

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQQueue class and the MQSession class.

**Defined in:** MQQueue class

**Syntax:** Call *MQQueue.***ClearErrorCodes()**

## Close method

Closes a queue using the current values of CloseOptions.

**Defined in:** MQQueue class

**Syntax:** Call *MQQueue*.**Close()**

## Get method

Retrieves a message from the queue.

This method takes an MQMessage object as a parameter. It uses some of the fields in this object's MQMD as input parameters - in particular the MessageId and CorrelId, so it is important to ensure that these are set as required. See the WebSphere MQ Application Programming Reference for details.

If the method fails then the MQMessage object is unchanged. If it succeeds then the MQMD and Message Data portions of the MQMessage object are completely replaced with the MQMD and Message Data from the incoming message. The MQMessage control properties are set as follows

- **MessageLength** is set to length of the WebSphere MQ message
- **DataLength** is set to length of the WebSphere MQ message
- **DataOffset** is set to zero

**Defined in:**
    MQQueue class

**Syntax:** Call *MQQueue*.**Get**(*Message, GetMsgOptions, GetMsgLength*)

**Parameters**
    Message:

    MQMessage Object representing message to be retrieved.

    GetMsgOptions:

    Optional MQGetMessageOptions object to control the get operation. If these are not specified, default MQGetMessageOptions are used.

    GetMsgLength:

    Optional 2- or 4-byte length value to control the maximum length of the WebSphere MQ message that will be retrieved from the queue.

    If the MQGMO_ACCEPT_TRUNCATED_MSG option *is* specified, the GET succeeds with a completion code of MQCC_WARNING and a reason code of MQRC_TRUNCATED_MSG_ACCEPTED if the message size exceeds the specified length.

    The MessageData holds the first GetMsgLength bytes of data.

    If MQGMO_ACCEPT_TRUNCATED_MSG *is not* specified, and the message size exceeds the specified length, the completion code of MQCC_FAILED together with reason code MQRC_TRUNCATED_MESSAGE_FAILED is returned.

    If the contents of the message buffer are undefined, the total message length is set to the full length of the message that would have been retrieved.

If the message length parameter is not specified, the length of the message buffer is automatically adjusted to at least the size of the incoming message.

## Open method

Opens a queue using the current values of:

1. QueueName
2. QueueManagerName
3. AlternateUserId
4. DynamicQueueName

**Defined in:**
    MQQueue class

**Syntax:** Call *MQQueue*.**Open()**

## Put method

Places a message onto the queue.

This method takes an MQMessage object as a parameter. The Message Descriptor (MQMD) properties of this object may be altered as a result of this method. The values they have immediately after this method has run are the values that were put onto the WebSphere MQ.

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

**Defined in:**
    MQQueue class

**Syntax:** Call *MQQueue*.**Put**(*Message, PutMsgOptions)*

**Parameters**
    Message

    MQMessage object representing message to be put.

    PutMsgOptions

    MQPutMessageOptions object containing options to control the put operation. If these are not specified, default PutMessageOptions are used.

## MQMessage class

This class represents a WebSphere MQ message. It includes properties to encapsulate the WebSphere MQ message descriptor (MQMD), and provides a buffer to hold the application-defined message data.

The class includes Write methods to copy data from an ActiveX application to an MQMessage object and similarly Read methods to copy data from an MQMessage object to an ActiveX application. The class manages the allocation and deallocation of memory for the buffer automatically. The application does not have to declare the size of the buffer when an MQMessage object is created because the buffer grows to accommodate data written to it.

You cannot place a message onto a WebSphere MQ queue if the buffer size exceeds the MaximumMessageLength property of that queue.

After it has been constructed, an MQMessage object may be Put onto a WebSphere MQ queue using the MQQueue.Put method. This method takes a copy of the MQMD and message data portions of the object and places that copy on the queue - so the application may modify or delete an MQMessage object after the Put, without affecting the message on the WebSphere MQ queue. The queue manager may adjust some of the fields in the MQMD when it copies the message on the WebSphere MQ queue.

An incoming message may be read into an MQMessage object using the MQQueue.Get method. This replaces any MQMD or message data that may already have been in the MQMessage object with values from the incoming message, adjusting the size of the MQMessage object's data buffer to match the size of the incoming message data.

## Containment

Messages are contained by the MQSession class.

### Creation

**New** creates a new MQMessage object. Its Message Descriptor properties are initially set to default values, and its Message Data buffer is empty.

### Syntax

**Dim** *msg* **As New MQMessage** or **Set** *msg* = **New MQMessage**

### Properties

The control properties are:
- CompletionCode property
- DataLength property
- DataOffset property
- MessageLength property
- ReasonCode property
- ReasonName property

The Message Descriptor properties are:
- AccountingToken property
- AccountingTokenHex property
- ApplicationIdData property
- ApplicationOriginData property
- BackoutCount property
- CharacterSet property
- CorrelationId property
- CorrelationIdHex property
- Encoding property
- Expiry property
- Feedback property

- Format property
- GroupId property
- GroupIdHex property
- MessageData property
- MessageFlags property
- MessageId property
- MessageIdHex property
- MessageSequenceNumber property
- MessageType property
- Offset property
- OriginalLength property
- Persistence property
- Priority property
- PutApplicationName property
- PutApplicationType property
- PutDateTime property
- ReplyToQueueManagerName property
- ReplyToQueueName property
- Report property
- TotalMessageLength property
- UserId property

## Methods
- ClearErrorCodes method
- ClearMessage method
- Read method
- ReadBoolean method
- ReadByte method
- ReadDecimal2 method
- ReadDecimal4 method
- ReadDouble method
- ReadDouble4 method
- ReadFloat method
- ReadInt2 method
- ReadInt4 method
- ReadLong method
- ReadNullTerminatedString method
- ReadShort method
- ReadString method
- ReadUTF method
- ReadUInt2 method
- ReadUnsignedByte method
- ResizeBuffer method
- Write method
- WriteBoolean method

- WriteByte method
- WriteDecimal2 method
- WriteDecimal4 method
- WriteDouble method
- WriteDouble4 method
- WriteFloat method
- WriteInt2 method
- WriteInt4 method
- WriteLong method
- WriteNullTerminatedString method
- WriteShort method
- WriteUTF method
- WriteString method
- WriteUInt2 method
- WriteUnsignedByte method

## Property access

All properties can be read at any time.

The control properties are read-only, except for DataOffset which is read-write. The Message Descriptor properties are all read-write, except BackoutCount and TotalMessageLength which are both read-only.

Note however that some of the MQMD properties may be modified by the queue manager when the message is put onto a WebSphere MQ queue. See the WebSphere MQ Application Programming Reference for details.

You can pass binary data to a WebSphere MQ message by setting the CharacterSet property to the Coded Character Set Identifier of the queue manager (MQCCSI_Q_MGR), and passing it a string. You can use the chr$ function to set non-character data into the string.

## Data conversion

The Read and Write methods perform data conversion. They convert between the ActiveX internal formats, and the WebSphere MQ message formats as defined by the Encoding and CharacterSet properties from the message descriptor. When writing a message you should, if possible, set values into Encoding and CharacterSet that match the characteristics of the recipient of the message before issuing a Write method. When reading a message, this is not normally required because these values will have been set from those in the incoming MQMD.

This is an additional data conversion step that happens after any conversion performed by the MQQueue.Get method.

# CompletionCode property

Read-only. Returns the WebSphere MQ completion code set by the most recent method or property access issued against this object.

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode& = MQMessage*.**CompletionCode**

# DataLength property

Read-only. This property returns the value:
```
MQMessage.MessageLength - MQMessage.DataOffset
```

It can be used before a Read method, to check that the expected number of characters are actually present in the buffer.

The initial value is zero.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *bytesleft& = MQMessage*.**DataLength**

# DataOffset property

Read-write. The current position within the Message Data portion of the message object.

The value is expressed as a byte offset from the start of the message data buffer; the first character in the buffer corresponds to a DataOffset value of zero.

A read or write method commences its operation at the character referenced by DataOffset. These methods process data in the buffer sequentially from this position, and update DataOffset to point to the byte (if any) immediately following the last byte processed.

DataOffset may take only values in the range zero to MessageLength inclusive. When DataOffset = MessageLength it is pointing to the end, that is the first invalid character of the buffer. Write methods are permitted in this situation - they extend the data in the buffer and increase MessageLength by the number of bytes added. Reading beyond the end of the buffer is not valid.

The initial value is zero.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *currpos& = MQMessage*.**DataOffset**

To set: *MQMessage*.**DataOffset** = *currpos&*

## MessageLength property

Read-only. Returns the total length of the Message Data portion of the message object in characters, irrespective of the value of DataOffset.

The initial value is zero. It is set to the incoming Message Length after a Get method invocation that referenced this message object. It is incremented if the application uses a Write method to add data to the object. It is unaffected by Read methods.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *msglength& = MQMessage*.**MessageLength**

## ReasonCode property

Read-only. Returns the reason code set by the most recent method or property access issued against this object.

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = MQMessage*.**ReasonCode**

## ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example, "MQRC_QMGR_NOT_AVAILABLE". **Defined in:** MQMessage class

**Data Type:** String

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQMessage*.**ReasonName**

## AccountingToken property

Read-write. The MQMD AccountingToken - part of the message Identity Context.

Its initial value is all nulls.

**Defined in:** MQMessage class

**Data Type:** String of 32 characters

**Syntax:** To get: *actoken$ = MQMessage*.**AccountingToken**

To set: *MQMessage.***AccountingToken** = *actoken$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use AccountingTokenHex in place of te AccountingToken property.

## AccountingTokenHex property

Read-write. The MQMD AccountingToken - part of the message Identity Context.

Every two characters represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B", and so on.

You must supply 64 valid hexadecimal characters.

Its initial value is "0...0"

**Defined in:** MQMessage class

**Data Type:** String of 64 hexadecimal characters representing 32 ASCII characters

**Syntax:** To get: *actokenh$ = MQMessage.***AccountingTokenHex**

To set: *MQMessage.***AccountingTokenHex** *= actokenh$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use AccountingTokenHex in place of the AccountingToken property.

## ApplicationIdData property

Read-write. The MQMD ApplIdentityData - part of the message Identity Context.

Its initial value is all blanks.

**Defined in:** MQMessage class

**Data Type:** String of 32 characters

**Syntax:** To get: *applid$ = MQMessage.***ApplicationIdData**

To set: *MQMessage.***ApplicationIdData** *= applid$*

## ApplicationOriginData property

Read-write. The MQMD ApplOriginData - part of the message origin context.

Its initial value is all blanks.

**Defined in:** MQMessage class

**Data Type:** String of 4 characters

**Syntax:** To get: *applor$ = MQMessage.***ApplicationOriginData**

To set: *MQMessage.***ApplicationOriginData** *= applor$*

## BackoutCount property

Read-only. The MQMD BackoutCount.

Its initial value is 0

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *backoutct& = MQMessage*.**BackoutCount**

## CharacterSet property

Read-write. The MQMD CodedCharSetId.

Its initial value is the special value MQCCSI_Q_MGR.

If CharacterSet is set to MQCCSI_Q_MGR, the WriteString method does not perform code-page conversion.

For example:
```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

where 'n' is greater than or equal to zero and less than or equal to 255, results in a single byte of value of 'n' being written to the buffer.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *ccid& = MQMessage*.**CharacterSet**

To set: *MQMessage*.**CharacterSe**t *= ccid&*

**Example**

If you want the string written out in code page 437, issue:
```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Set the value you want in the CharacterSet before issuing any WriteString calls.

## CorrelationId property

Read-write. The CorrelationId to be included in the MQMD of a message when put on a queue, also the Id to be matched against when getting a message from a queue.

Its initial value is null.

**Defined in:** MQMessage class

**Data Type:** String of 24 characters

**Syntax:** To get: *correlid$* = *MQMessage*.**CorrelationId** To set: *MQMessage*.**CorrelationId** = *correlid$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use CorrelationIdHex in place of the CorrelationId property.

## CorrelationIdHex property

Read-write. The CorrelationId to be included in the MQMD of a message when put on a queue, also the CorrelationId to be matched against when getting a message from a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B", and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0...0".

**Defined in:** MQMessage class

**Data Type:** String of 48 hexadecimal characters representing 24 ASCII characters

**Syntax:** To get: *correlidh$* = *MQMessage*.**CorrelationIdHex**

To set: *MQMessage*.**CorrelationIdHex** = *correlidh$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use CorrelationIdHex in place of the CorrelationId property.

## Encoding property

Read-write. The MQMD field that identifies the representation used for numeric values in the application message data.

Its initial value is the special value MQENC_NATIVE, which varies by platform.

This property is used by the following methods:
- ReadDecimal2 method
- ReadDecimal4 method
- ReadDouble method
- ReadDouble4 method
- ReadFloat method
- ReadInt2 method
- ReadInt4 method
- ReadLong method
- ReadShort method
- ReadUInt2 method
- WriteDecimal2 method
- WriteDecimal4 method

- WriteDouble method
- WriteDouble4 method
- WriteFloat method
- WriteInt2 method
- WriteInt4 method
- WriteLong method
- WriteShort method
- WriteUInt2 method

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *encoding& = MQMessage.**Encoding** To set: *MQMessage.**Encoding** = encoding&*

If you are preparing to write data to the message buffer, you should set this field to match the characteristics of the receiving queue manager platform if the receiving queue manager is incapable of performing its own data conversion.

## Expiry property

Read-write. The MQMD expiry time field, expected in tenths of a second.

Its initial value is the special value MQEI_UNLIMITED

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *expiry& = MQMessage.**Expiry**

To set: *MQMessage.**Expiry** = expiry&*

## Feedback property

Read-write. The MQMD feedback field.

Its initial value is the special value MQFB_NONE.

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *feedback& = MQMessage.**Feedback**

To set: *MQMessage.**Feedback** = feedback&*

# Format property

Read-write. The MQMD format field. Gives the name of a built-in or user-defined format that describes the nature of the Message Data.

Its initial value is the special value MQFMT_NONE.

**Defined in:** MQMessage class

**Data Type:** String of 8 characters

**Syntax:** To get: *format$ = MQMessage*.**Format**

To set: *MQMessage*.**Format** = *format$*

# GroupId property

Read-write. The GroupId to be included in the MQPMR of a message when put on a queue, also the Id to be matched against when getting a message from a queue. Its initial value is all nulls.

**Defined in:**
   MQMessage class

**Data Type:**
   String of 24 characters

**Syntax:** To get: *groupid$ = MQMessage*.**GroupId**

To set: *MQMessage*.**GroupId** = *groupid$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use GroupIdHex in place of the GroupId property.

# GroupIdHex property

Read-write. The GroupId to be included in the MQPMR of a message when put on a queue, also the Id to be matched against when getting a message from a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B" and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0...0".

**Defined in:**
   MQMessage class

**Data Type:**
   String of 48 hexadecimal characters representing 24 ASCII characters.

**Syntax:** To get: *groupidh$ = MQMessage*.**GroupIdHex**

To set: *MQMessage*.**GroupIdHex** = *groupidh$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use GroupIdHex in place of the GroupId property.

## MessageData property

Read-write. Retrieves or sets the entire contents of a message as a character string.

**Defined in:** MQMessage class

**Data Type:** Variant

**Note:** The data type used by this property is Variant but MQAX expects this to be a variant type of String. If you pass in a variant of other than this type then the error MQRC_OBJECT_TYPE_ERROR will be returned.

**Syntax:** To get: *String$ = MQMessage*.**MessageData**

To set: *MQMessage*.**MessageData** = *String$*

## MessageFlags property

Read-Write. Message flags specifying Segmentation control information. The initial value is 0.

**Defined in:**
        MQMessage class

**Data Type:**
        Long

**Values:**
        See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *messageflags& = MQMessage*.**MessageFlags**

To set: *MQMessage*.**MessageFlags** = *messageflags&*

## MessageId property

Read-write. The MessageId to be included in the MQMD of a message when put on a queue, also the Id to be matched against when getting a message from a queue.

Its initial value is all nulls.

**Defined in:** MQMessage class

**Data Type:** String of 24 characters

**Syntax:** To get: *messageid$ = MQMessage*.**MessageId**

To set: *MQMessage*.**MessageId** = *messageid$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use MessageIdHex in place of the MessageId property.

# MessageIdHex property

Read-write. The MessageId to be included in the MQMD of a message when put on a queue, also the MessageId to be matched against when getting a message from a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B", and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0...0".

**Defined in:** MQMessage class

**Data Type:** String of 48 hexadecimal characters representing 24 ASCII characters

**Syntax:** To get: *messageidh$ = MQMessage*.**MessageIdHex**

To set: *MQMessage*.**MessageIdHex** = *messageidh$*

**Note:** See "Message Descriptor properties" on page 4 for a discussion of when you must use MessageIdHex in place of the MessageId property.

# MessageSequenceNumber property

Read-Write. Sequence information identifying a message within a group. The initial value is 1.

**Defined in:**
  MQMessage class

**Data Type:**
  Long

**Values:**
  See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *sequencenumber& = MQMessage*.**SequenceNumber**

To set: *MQMessage*.**SequenceNumber** = *sequencenumber&*

# MessageType property

Read-write. The MQMD MsgType field.

Its initial value is MQMT_DATAGRAM.

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
• See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *msgtype&* = *MQMessage*.**MessageType**

To set: *MQMessage*.**MessageType** = *msgtype&*

## Offset property

Read-Write. The offset in a segmented message. The initial value is 0.

**Defined in:**
     MQMessage class

**Data Type:**
     Long

**Values:**
     See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *offset&* = *MQMessage*.**Offset**

To set: *MQMessage*.**Offset** = *offset&*

## OriginalLength property

Read-Write. The original length of a segmented message. The initial value is
MQOL_UNDEFINED

**Defined in:**
     MQMessage class

**Data Type:**
     Long

**Values:**
     See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *originallength&* = *MQMessage*.**OriginalLength**

To set: *MQMessage*.**OriginalLength** = *originallength&*

## Persistence property

Read-write. The message's persistence setting.

Its initial value is MQPER_PERSISTENCE_AS_Q_DEF.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *persist&* = *MQMessage*.**Persistence**

To set: *MQMessage*.**Persistence** = *persist&*

## Priority property

Read-write. The message's priority.

Its initial value is the special value MQPRI_PRIORITY_AS_Q_DEF

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *priority& = MQMessage*.**Priority**

To set: *MQMessage*.**Priority** = *priority&*

## PutApplicationName property

Read-write. The MQMD PutApplName - part of the Message Origin context.

Its initial value is all blanks.

**Defined in:** MQMessage class

**Data Type:** String of 28 characters

**Syntax:** To get: *putapplnm$ = MQMessage*.**PutApplicationName**

To set: *MQMessage*.**PutApplicationName** = *putapplnm$*

## PutApplicationType property

Read-write. The MQMD PutApplType - part of the Message Origin context.

Its initial value is MQAT_NO_CONTEXT

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
- See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *putappltp& = MQMessage*.**PutApplicationType**

To set: *MQMessage*.**PutApplicationType** = *putappltp&*

## PutDateTime property

Read-write. This property combines the MQMD PutDate and PutTime fields. These are part of the Message Origin context that indicate when the message was put.

The ActiveX Extension converts between ActiveX date/time format and the Date and Time formats used in a WebSphere MQ MQMD. If a message is received which has an invalid PutDate or PutTime, then the PutDateTime property after the get method will be set to EMPTY.

Its initial value is EMPTY.

**Defined in:** MQMessage class

**Data Type:** Variant of type 7 (date/time) or EMPTY.

**Syntax:** To get: *datetime = MQMessage*.**PutDateTime**

To set: *MQMessage*.**PutDateTime** = *datetime*

## ReplyToQueueManagerName property

Read-write. The MQMD ReplyToQMgr field.

Its initial value is all blanks

**Defined in:** MQMessage class

**Data Type:** String of 48 characters

**Syntax:** To get: *replytoqmgr$ = MQMessage*.**ReplyToQueueManagerName**

To set: *MQMessage*.**ReplyToQueueManagerName** = *replytoqmgr$*

## ReplyToQueueName property

Read-write. The MQMD ReplyToQ field.

Its initial value is all blanks

**Defined in:** MQMessage class

**Data Type:** String of 48 characters

**Syntax:** To get: *replytoq$ = MQMessage*.**ReplyToQueueName**

To set: *MQMessage*.**ReplyToQueueName** = *replytoq$*

## Report property

Read-write. The message's Report options.

Its initial value is MQRO_NONE.

**Defined in:** MQMessage class

**Data Type:** Long

**Values:**
• See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *report& = MQMessage*.**Report**

To set: *MQMessage*.**Report** = *report&*

## TotalMessageLength property

Read-only. Retrieves the length of the last message received by MQGET. If the message has not been truncated, this value is equal to the value of the MessageLength property.

**Defined in:** MQMessage class

**Data Type:** Long

**Syntax:** To get: *totalmessagelength&* = *MQMessage*.**TotalMessageLength**

## UserId property

Read-write. The MQMD UserIdentifier - part of the message Identity Context.

Its initial value is all blanks.

**Defined in:** MQMessage class

**Data Type:** String of 12 characters

**Syntax:** To get: *userid$* = *MQMessage*.**UserId**

To set: *MQMessage*.**UserId** = *userid$*

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQMessage class and the MQSession class.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**ClearErrorCodes()**

## ClearMessage method

This method clears the data buffer portion of the MQMessage object. Any Message Data in the data buffer is lost, because MessageLength, DataLength, and DataOffset are all set to zero.

The Message Descriptor (MQMD) portion is unaffected; an application may need to modify some of the MQMD fields before reusing the MQMessage object. If you wish to set the MQMD fields back to initial values you should use New to replace the object with a new instance.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**ClearMessage()**

## Read method

Reads a sequence of bytes from the message buffer into a byte array. DataOffset is incremented and Data Length decremented by the number of bytes read.

**Defined in:**
MQMessage class

**Syntax:** Data = MQMessage.**Read**(len&)

**Parameters:**
*len&:* Long. Length of data in bytes to be read.

## ReadBoolean method

Reads a 1-byte Boolean value from the current position in the message buffer and returns a 2-byte Boolean TRUE(-1)/FALSE(0) value. DataOffset is incremented by one and Data Length is decremented by one.

**Defined in:**
      MQMessage class

**Syntax:** *value = MQMessage*.**ReadBoolean**

## ReadByte method

This method reads 1 byte from the Message Data buffer, starting with the character referred to by DataOffset and returns it as an Integer (signed 2-byte) integer value in the range -128 to 127.

The method fails if MQMessage.DataLength is less than 1 when it is issued.

DataOffset is incremented by 1 and DataLength is decremented by 1 if the method succeeds.

The byte of message data is assumed to be a signed binary integer.

**Defined in:** MQMessage class

**Syntax:** *integerv% = MQMessage*.**ReadByte**

## ReadDecimal2 method

Reads a 2-byte packed decimal number and returns it as a signed 2-byte integer value. DataOffset is incremented by two and Data Length is decremented by two.

**Defined in:**
      MQMessage class

**Syntax:** *value% = MQMessage*.**ReadDecimal2**

## ReadDecimal4 method

Reads a 4-byte packed decimal number and returns it as a signed 4-byte integer value. DataOffset is incremented by four and Data Length is decremented by four.

**Defined in:**
      MQMessage class

**Syntax:** Call *value& = MQMessage*.**ReadDecimal4**

## ReadDouble method

This method reads 8 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as a Double (signed 8-byte) floating point value.

The method fails if MQMessage.DataLength is less than 8 when it is issued.

DataOffset is incremented by 8 and DataLength is decremented by 8 if the method succeeds.

The 8 characters of message data are assumed to be a binary floating point number whose encoding is specified by the MQMessage.Encoding property. Note that conversion from System/360™ format is not supported.

**Defined in:** MQMessage class

**Syntax:** *doublev# = MQMessage*.**ReadDouble**

# ReadDouble4 method

The ReadDouble4 and WriteDouble4 methods are alternatives to ReadFloat and WriteFloat. This is because they support 4-byte System/390 floating point message values that are too large to convert to 4-byte IEEE floating point format.

This method reads 4 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as a Double (signed 8-byte) floating point value.

The method fails if MQMessage.DataLength is less than 4 when it is issued.

DataOffset is incremented by 4 and DataLength is decremented by 4 if the method succeeds.

The 4 characters of message data are assumed to be a binary floating point number whose encoding is specified by the MQMessage.Encoding property. Note that conversion from System/360 format is not supported.

**Defined in:** MQMessage class

**Syntax:** *doublev# = MQMessage*.**ReadDouble4**

# ReadFloat method

This method reads 4 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as a Single (signed 4-byte) floating point value.

The method fails if MQMessage.DataLength is less than 4 when it is issued.

DataOffset is incremented by 4 and DataLength is decremented by 4 if the method succeeds.

The 4 characters of message data are assumed to be a floating point number whose encoding is specified by the MQMessage.Encoding property. Note that conversion from System/360 format is not supported.

**Defined in:** MQMessage class

**Syntax:** *singlev! = MQMessage*.**ReadFloat**

## ReadInt2 method

The method is identical to the ReadShort method.

**Syntax:** *integerv% = MQMessage*.**ReadInt2**

## ReadInt4 method

This method is identical to the ReadLong method.

**Syntax:** *bigint& = MQMessage*.**ReadInt4**

## ReadLong method

This method reads 4 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as a Long (signed 4-byte) integer value.

The method fails if MQMessage.DataLength is less than 4 when it is issued.

DataOffset is incremented by 4 and DataLength is decremented by 4 if the method succeeds.

The 4 characters of message data are assumed to be a binary integer whose encoding is specified by the MQMessage.Encoding property.

**Defined in:** MQMessage class

**Syntax:** *bigint& = MQMessage*.**ReadLong**

## ReadNullTerminatedString method

This is for use in place of ReadString if the string may contain embedded null characters.

This method reads the specified number of bytes from the message data buffer starting with the byte referred to by DataOffset and returns it as an ActiveX string. If the string contains an embedded null before the end then the length of the returned string is reduced to reflect only those characters before the null.

DataOffset is incremented and DataLength is decremented by the value specified regardless of whether or not the string contains embedded null characters.

The characters in the message data are assumed to be a string whose code page is specified by the MQMessage.CharacterSet property. Conversion to ActiveX representation is performed for the application.

**Defined in:**
    MQMessage class

**Syntax:** *string$ = MQMessage*.**ReadNullTerminatedString(length&)**

**Parameters:**
    length& *Long.* Length of string field in bytes.

# ReadShort method

This method reads 2 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as an Integer (signed 2-byte) value.

The method fails if MQMessage.DataLength is less than 2 when it is issued.

DataOffset is incremented by 2 and DataLength is decremented by 2 if the method succeeds.

The 2 characters of message data are assumed to be a binary integer whose encoding is specified by the MQMessage.Encoding property.

**Defined in:** MQMessage class

**Syntax:** *integerv% = MQMessage*.**ReadShort**

# ReadString method

This method reads n bytes from the Message Data buffer starting with the byte referred to by DataOffset and returns it as an ActiveX string.

The method fails if MQMessage.DataLength is less than n when it is issued.

DataOffset is incremented by n and DataLength is decremented by n if the method succeeds.

The n characters of message data are assumed to be a string whose code page is specified by the MQMessage.CharacterSet property. Conversion to ActiveX representation is performed for the application.

**Defined in:** MQMessage class

**Syntax:** *stringv$ = MQMessage*.**ReadString**(*length&*)

**Parameter**
> *length&* Long. Length of string field in bytes.

# ReadUInt2 method

This method reads 2 bytes from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as a Long (signed 4-byte) integer value.

The method fails if MQMessage.DataLength is less than 2 when it is issued.

DataOffset is incremented by 2 and DataLength is decremented by 2 if the method succeeds.

The 2 bytes of message data are assumed to be an unsigned binary integer whose encoding is specified by the MQMessage.Encoding property.

**Defined in:** MQMessage class

**Syntax:** *bigint& = MQMessage*.**ReadUInt2**

## ReadUnsignedByte method

This method reads 1 byte from the Message Data buffer, starting with the byte referred to by DataOffset and returns it as an Integer (signed 2-byte) integer value in the range 0 to 255.

The method fails if MQMessage.DataLength is less than 1 when it is issued.

DataOffset is incremented by 1 and DataLength is decremented by 1 if the method succeeds.

The 1 character of message data is assumed to be an unsigned binary integer.

**Defined in:** MQMessage class

**Syntax:** *integerv% = MQMessage*.**ReadUnsignedByte**

## ReadUTF method

This method reads a UTF format string from the message starting with the byte referred to by DataOffset and returns it as an ActiveX string. The string in the message consists of a 2-byte length followed by the character data.

The method fails if MQMessage.DataLength is less than the string length when it is issued.

DataOffset is incremented by the string length and DataLength is decremented by the string length if the method succeeds.

**Defined in:**
MQMessage class

**Syntax:** *value$ = MQMessage*.**ReadUTF**

## ResizeBuffer method

This method alters the amount of storage currently allocated internally to hold the Message Data buffer. It gives the application some control over the automatic buffer management, in that if the application knows that it is going to deal with a large message, it can ensure that a sufficiently large buffer is allocated. The application does not need to use this call - if it does not, the automatic buffer management code will grow the buffer size to fit.

If you resize the buffer to be smaller that the current MessageLength, you risk losing data. If you do lose data, the method returns a CompletionCode of MQCC_WARNING and a ReasonCode of MQRC_DATA_TRUNCATED.

If you resize the buffer to be smaller than the value of the **DataOffset** property the:
- **DataOffset** property is changed to point to the end of the new buffer
- **DataLength** property is set to zero
- **MessageLength** property is changed to the new buffer size

**Defined in:**
MQMessage class

**Syntax:** *MQMessage*.**ResizeBuffer**(*Length&*)

**Parameter:**
>Length& Long. Size required in characters.

# Write method

Writes a sequence of bytes to the message buffer from a byte array at the position referred to by Data Offset. If necessary the length of the buffer (MQMessage.MQMessageLength) is extended to accommodate the full length of the byte array. DataOffset is incremented by the number of bytes written if the method succeeds.

**Defined in:**
>MQMessage class

**Syntax:** Call *MQMessage*.**Write**(value)

**Parameters:**
>*data:* a byte array or a variant reference to a byte array

# WriteBoolean method

Writes a 1-byte Boolean value at the current position in the message buffer from a 2-byte Boolean value. DataOffset is incremented by one.

**Defined in:**
>MQMessage class

**Syntax:** Call *MQMessage*.**WriteBoolean**(value)

**Parameter:**
>*value:* Boolean (2-bytes). Value to be written.

# WriteByte method

This method takes a signed 2-byte integer value and writes it into the Message Data buffer as a 1-byte binary number at the position referred to by DataOffset. It replaces any data already at the position in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by one if the method succeeds.

The value specified should be in the range -128 to 127. If it is not, the method returns with CompletionCode MQCC_FAILED and ReasonCode MQRC_WRITE_VALUE_ERROR.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteByte**(*value%* )

**Parameter:** *value%* Integer. Value to be written.

# WriteDecimal2 method

Writes a signed 2-byte integer as a 2-byte packed decimal number. DataOffset is incremented by two.

**Defined in:**
    MQMessage class

**Syntax:** Call *MQMessage*.**WriteDecimal2**(value%)

**Parameter:**
    *value% Integer*. Value to be written.

# WriteDecimal4 method

Writes a signed 4-byte integer as a 4-byte packed decimal number. DataOffset is incremented by four.

**Defined in:**
    MQMessage class

**Syntax:** Call *MQMessage*.**WritedDecimal4**(value&)

**Parameter:**
    *value& Long.* Value to be written.

# WriteDouble method

This method takes a signed 8-byte floating point value and writes it into the Message Data buffer as an 8-byte floating point number starting at the position referred to by DataOffset. It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 8 if the method succeeds.

The method converts to the floating point representation specified by the MQMessage.Encoding property. *Conversion to System/360 format is not supported.*

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteDouble**(*value*# )

**Parameter:**
    value# Double. Value to be written.

# WriteDouble4 method

See "ReadDouble4 method" on page 65 for a description of when ReadDouble4 and WriteDouble4 should be used in place of ReadFloat and WriteFloat.

This method takes a signed 8-byte floating point value and writes it into the Message Data buffer as a 4-byte floating number starting at the position referred to by DataOffset.

DataOffset is incremented by 4 if the method succeeds.

It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

The method converts to the floating point representation specified by the MQMessage.Encoding property. *Conversion to System/360 format is not supported.*

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteDouble4(value#)**

**Parameter:** *value#* Double. Value to be written.

# WriteFloat method

This method takes a signed 4-byte floating point value and writes it into the Message Data buffer as a 4-byte floating point number starting at the character referred to by DataOffset. It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 4 if the method succeeds.

The method converts to the binary representation specified by the MQMessage.Encoding property. *Conversion to System/360 format is not supported.*

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteFloat**(*value!* )

**Parameter** *value!* Float. Value to be written.

# WriteInt2 method

This method is identical to the WriteShort method.

**Syntax:** Call *MQMessage*.**WriteInt2**(*value%* )

**Parameter** *value%* Integer. Value to be written.

# WriteInt4 method

This method is identical to the WriteLong method.

**Syntax:** Call *MQMessage*.**WriteInt4**(*value&* )

**Parameter** *value&* Long. Value to be written.

# WriteLong method

This method takes a signed 4-byte integer value and writes it into the Message Data buffer as a 4-byte binary number starting at the byte referred to by DataOffset. It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 4 if the method succeeds.

The method converts to the binary representation specified by the MQMessage.Encoding property.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteLong**(*value&*)

**Parameter** *value&* Long. Value to be written.

## WriteNullTerminatedString method

This method performs a normal WriteString and pads any remaining bytes up to the specified length with null. If the number of bytes written by the initial write string is equal to the specified length then no nulls are written. If the number of bytes exceeds the specified length then an error (reason code MQRC_WRITE_VALUE_ERROR) is set.

DataOffset is incremented by the specified length if the method succeeds.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteNullTerminatedString**(*value$, length&*)

**Parameters:**
    value$ String. Value to be written.

length& Long. Length of string field in bytes.

## WriteShort method

This method takes a signed 2-byte integer value and writes it into the Message Data buffer as a 2-byte binary number starting at the byte referred to by DataOffset. It replaces any data already at these positions in the buffer, and will extend the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 2 if the method succeeds.

The method converts to the binary representation specified by the MQMessage.Encoding property.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteShort**(*value%*)

**Parameter** *value%* Integer. Value to be written.

## WriteString method

This method takes an ActiveX string and writes it into the Message Data buffer starting at the byte referred to by DataOffset. It replaces any data already at these positions in the buffer, and will extend the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by the length of the string in bytes if the method succeeds.

The method converts characters into the code page specified by the MQMessage.CharacterSet property.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteString**(*value$*)

**Parameter** *value$* String. Value to be written.

# WriteUInt2 method

This method takes a signed 4-byte integer value and writes it into the Message Data buffer as a 2-byte unsigned binary number starting at the byte referred to by DataOffset. It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 2 if the method succeeds.

The method converts to the binary representation specified by the MQMessage.Encoding property. The value specified should be in the range 0 to 2**16-1. If it is not the method returns with CompletionCode MQCC_FAILED and ReasonCode MQRC_WRITE_VALUE_ERROR.

**Defined in:** MQMessage class

**Syntax:** Call *MQMessage*.**WriteUInt2**(*value&* )

**Parameter** *value&* Long. Value to be written.

# WriteUnsignedByte method

This method takes a signed 2-byte integer value and writes it into the Message Data buffer as a 1-byte unsigned binary number starting at the character referred to by DataOffset. It replaces any data already at these positions in the buffer, and extends the length of the buffer (MQMessage.MessageLength) if necessary.

DataOffset is incremented by 1 if the method succeeds.

The value specified should be in the range 0 to 255. If it is not the method returns with CompletionCode MQCC_FAILED and ReasonCode MQRC_WRITE_VALUE_ERROR.

**Defined in:**
    MQMessage class

**Syntax:** Call *MQMessage*.**WriteUnsignedByte**(*value%*)

**Parameter** *value%* Integer. Value to be written.

# WriteUTF method

This method takes an ActiveX string and writes it into the message data buffer at the current position in UTF format. The data written consists of a 2-byte length followed by the character data. DataOffset is incremented by the length of the string if the method succeeds.

**Defined in:**
    MQMessage class

**Syntax: Call** *MQMessage*.**WriteUTF**(value$)

**Parameter:**
> *value$ String.* Value to be written.

---

# MQPutMessageOptions class

This class encapsulates the various options that control the action of putting a message onto a WebSphere MQ Queue.

## Containment

Contained by the MQSession class.

### Creation

**New** creates a new MQPutMessageOptions object and sets all its properties to initial values.

Alternatively, use the AccessPutMessageOptions method of the MQSession class.

### Syntax:

**Dim** *pmo* **As New MQPutMessageOptions** or

**Set** *pmo* = **New MQPutMessageOptions**

### Properties
- CompletionCode property
- Options property
- ReasonCode property
- ReasonName property
- RecordFields property
- ResolvedQueueManagerName property
- ResolvedQueueName property

### Methods
- ClearErrorCodes method

# CompletionCode property

Read-only. Returns the completion code set by the last method or property access issued against the object.

**Defined in:** MQPutMessageOptions class

**Data Type:** Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode&* = *PutOpts*.**CompletionCode**

# Options property

Read-write. The MQPMO Options field. See the WebSphere MQ Application Programming Reference for details. Initial value is MQPMO_NONE.

**Defined in:** MQPutMessageOptions Class.

**Data Type:** Long

**Syntax:** To get: *options& = PutOpts*.**Options**

To set: *PutOpts*.**Options** = *options&*

The MQPMO_PASS_IDENTITY_CONTEXT and MQPMO_PASS_ALL_CONTEXT options are not supported.

# ReasonCode property

Read-only. Returns the reason code set by the last method or property access issued against the object.

**Defined in:** MQPutMessageOptions class

**Data Type:** Long

**Values:**
• See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = PutOpts*.**ReasonCode**

# ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example, "MQRC_QMGR_NOT_AVAILABLE".

**Defined in:** MQPutMessageOptions class

**Data Type:** String

**Values:**
• See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = PutOpts*.**ReasonName**

# RecordFields property

Read-write. Flags indicating which fields are to be customized on a per-queue basis when putting a message to a distribution list. The initial value is zero.

This property corresponds to the PutMsgRecFields flags in the MQI MQPMO structure. In the MQI, these flags control which fields (in the MQPMR structure) are present and used by the MQPUT. In an MQPutMessageOptions object these fields are always present and the flags therefore only affect which fields are used by the Put. See the *WebSphere MQ Application Programming Reference* for further details.

**Defined in:**
> MQPutMessageOptions class

**Data Type:**
> Long

**Syntax:** To get: *recordfields& = PutOpts*.**RecordFields**

To set: *PutOpts*.**RecordFields** = *recordfields&*

## ResolvedQueueManagerName property

Read-only. The MQPMO ResolvedQMgrName field. See the WebSphere MQ
Application Programming Reference for details. The initial value is all blanks.

**Defined in:** MQPutMessageOptions class

**Data Type:** String of 48 characters

**Syntax:** To get: *qmgr$ = PutOpts*.**ResolvedQueueManagerName**

## ResolvedQueueName property

Read-only. The MQPMO ResolvedQName field. See the WebSphere MQ
Application Programming Reference for details. The initial value is all blanks.

**Defined in:** MQPutMessageOptions class

**Data Type:** String of 48 characters

**Syntax:** To get: *qname$ = PutOpts*.**ResolvedQueueName**

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE
for both the MQPutMessageOptions class and the MQSession class.

**Defined in:** MQPutMessageOptions class

**Syntax: Call** *PutOpts*.**ClearErrorCodes()**

# MQGetMessageOptions class

This class encapsulates the various options that control the action of getting a
message from a WebSphere MQ queue.

# Containment

Contained by the MQSession class.

### Properties
- CompletionCode property
- MatchOptions property
- Options property

- ReasonCode property
- ReasonName property
- ResolvedQueueName property
- WaitInterval property

### Methods
- ClearErrorCodes method

### Creation

**New** creates a new MQGetMessageOptions object and sets all its properties to initial values.

Alternatively, use the AccessGetMessageOptions method of the MQSession class.

### Syntax:

**Dim** *gmo* **As New MQGetMessageOptions** or

**Set** *gmo* = **New MQGetMessageOptions**

## CompletionCode property

Read-only. Returns the completion code set by the last method or property access issued against the object.

**Defined in:** MQGetMessageOptions Class.

**Data Type:** Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode&* = *GetOpts*.**CompletionCode**

## MatchOptions property

Read-write. Options controlling selection criteria used for MQGET. The initial value is MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID.

**Defined in:**
　　　MQGetMessageOptions class

**Data Type:**
　　　Long

**Values:**
　　　See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *matchoptions&* = *GetOpts*.**MatchOptions**

To set: *GetOpts*.**MatchOptions** =*matchoptions&*

## Options property

Read-write. The MQGMO Options field. See the WebSphere MQ Application
Programming Reference for details. Initial value is MQGMO_NO_WAIT.

**Defined in:** MQGetMessageOptions Class.

**Data Type:** Long

**Syntax:** To get: *options& = GetOpts*.**Options** To set: *GetOpts*.**Options** = *options&*

## ReasonCode property

Read-only. Returns the reason code set by the last method or property access
issued against the object.

**Defined in:** MQGetMessageOptions class

**Data Type:** Long

**Values:**
* See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = GetOpts*.**ReasonCode**

## ReasonName property

Read-only. Returns the symbolic name of the latest reason code. For example,
"MQRC_QMGR_NOT_AVAILABLE". **Defined in:** MQGetMessageOptions class

**Data Type:** String

**Values:**
* See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQGetMessageOptions*.**ReasonName**

## ResolvedQueueName property

Read-only. The MQGMO ResolvedQName field. See the WebSphere MQ
Application Programming Reference for details. The initial value is all blanks.

**Defined in:** MQGetMessageOptions class

**Data Type:** String of 48 characters

**Syntax:** To get: *qname$ = GetOpts*.**ResolvedQueueName**

## WaitInterval property

Read-write. The MQGMO WaitInterval field. The maximum time, in milliseconds,
that the Get will wait for a suitable message to arrive - if wait action has been
requested by the Options property. See the WebSphere MQ Application
Programming Reference for details. Initial value is 0.

**Defined in:** MQGetMessageOptions class

**Data Type:** Long

**Syntax:** To get: *wait&* = *GetOpts*.**WaitInterval**

To set: *GetOpts*.**WaitInterval** = wait&

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQGetMessageOptions class and the MQSession class.

**Defined in:** MQGetMessageOptions class

**Syntax: Call** *GetOpts*.**ClearErrorCodes()**

## MQDistributionList class

This class encapsulates a collection of queues - local, remote, or alias for output.

### Properties:
- AlternateUserId property
- CloseOptions property
- CompletionCode property
- ConnectionReference property
- FirstDistributionListItem property
- IsOpen property
- OpenOptions property
- ReasonCode property
- ReasonName property

### Methods:
- AddDistributionListItem method
- ClearErrorCodes method
- Close method
- Open method
- Put method

### Creation:

**new** creates a new MQDistributionList object.

Alternatively, use the AddDistributionList method of the MQQueueManager class.

### Syntax:

**Dim** *distlist*.**As New MQDistributionList** or **Set** *distlist* = **New MQDistributionList**

## AlternateUserId property

Read-write. The alternate user ID used to validate access to the list of queues when they are opened.

**Defined in:**
> MQDistributionList class

**Data Type:**
> String of 12 characters

**Syntax:** To get: *altuser$ = MQDistributionList*.**AlternateUserId**

To set: *MQDistributionList*.**AlternateUserId** = *altuser$*

## CloseOptions property

Read-write. Options used to control what happens when the distribution list is closed. The initial value is MQCO_NONE.

**Defined in:**
> MQDistributionList class

**Data Type:**
> Long

**Values:**
> - MQCO_NONE
> - MQCO_DELETE
> - MQCO_DELETE_PURGE

**Syntax:** To get: *closeopt& = MQDistributionList*.**CloseOptions**

To set: *MQDistributionList*.**CloseOptions** = *closeopt&*

## CompletionCode property

Read-only. The completion code set by the last method or property access issued against the object.

**Defined in:**
> MQDistributionList class

**Data Type:**
> Long

**Values:**
> - MQCC_OK
> - MQCC_WARNING
> - MQCC_FAILED

**Syntax:** To get: *completioncode& = MQDistributionList*.**CompletionCode**

## ConnectionReference property

Read-write. The queue manager to which the distribution list belongs.

**Defined in:**
> MQDistributionList class

**Data Type:**
> MQQueueManager

**Syntax:** To get: *set queuemanager = MQDistributionList*.**ConnectionReference**

To set: *set MQDistributionList*. **ConnectionReference** = *queuemanager*

## FirstDistributionListItem property

Read-only. The first distribution list item object associated with the distribution list.

**Defined in:**
> MQDistributionList class

**Data Type:**
> MQDistributionListItem

**Values:**

**Syntax:** To get: *set distributionlistitem =
MQDistributionList*.**FirstDistributionListItem**

## IsOpen property

Read-only. A value that indicates whether or not the distribution list is currently open.

**Defined in:**
> MQDistributionList class

**Data Type:**
> Boolean

**Values:**
> - TRUE (-1)
> - FALSE (0)

**Syntax:** To get: *IsOpen = MQDistributionList*.**IsOpen**

## OpenOptions property

Read-write. Options to be used when the distribution list is opened.

**Defined in:**
> MQDistributionList class

**Data Type:**
> Long

**Values:**
> See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *openopt& = MQDistributionList*.**OpenOptions**

To set: *MQDistributionList*.**OpenOptions** = openopt&

# ReasonCode property

Read-only. The reason code set by the last method or property access issued against the object.

**Defined in:**
> MQDistributionList class

**Data Type:**
> Long

**Values:**
> See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasoncode& = MQDistributionList*.**ReasonCode**

# ReasonName property

Read-only. The symbolic name for the ReasonCode. For example "MQRC_QMGR_NOT_AVAILABLE".

**Defined in:**
> MQDistributionList class

**Data Type:**
> String

**Values:**
> See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQDistributionList*.**ReasonName**

# AddDistributionListItem method

Creates a new MQDistributionListItem object and associates it with the distribution list object. The queue name parameter is mandatory.

The DistributionList property of the distribution list item is set to the owning distribution list and the FirstDistributionListItem property of the distribution list is set to reference this new distribution list item.

For the new distribution list item, the PreviousDistributionListItem property is set to nothing and the NextDistributionListItem property is set to reference any distribution list item that was previously first, or nothing if there was none previously (that is, the new one is inserted in front of those that exist already).

This will return an error if the distribution list is open.

**Defined in:**
> MQDistributionList class

**Syntax:** set distributionlistitem = *MQDistributionList*.**AddDistributionListItem** (QName$, QMgrName$)

**Parameters:**

> *QName$* String. Name of the WebSphere MQ queue.

> *QMgrName$* String. Name of the WebSphere MQ queue manager.

# ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQDistributionList class and the MQSession class.

**Defined in:**
MQDistributionList class

**Syntax:** Call *MQDistributionList*.**ClearErrorCodes**()

# Close method

Closes a distribution list using the current value of Close options.

**Defined in:**
MQDistributionList class

**Syntax:** Call *MQDistributionList*.**Close**()

# Open method

Opens each of the queues specified by the QueueName and (where appropriate) QueueManagerName properties of the distribution list items associated with the current object using the current value of AlternateUserId.

**Defined in:**
MQDistributionList class

**Syntax:** Call *MQDistributionList*.**Open**()

# Put method

Places a message on each of the queues identified by the distribution list items associated with the distribution list.

This method takes an MQMessage object as a parameter. The following distribution list item properties may be altered as a result of this method:
- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Feedback
- AccountingToken
- AccountingTokenHex

## Defined in:

MQDistributionList class

**Syntax:**

Call *MQDistributionList*.**Put**(Message, PutMsgOptions&)

**Parameters:**

*Message* MQMessage object representing the message to be put.

*PutMsgOptions* MQPutMessageOptions object containing options to control the put operation. If not specified, default PutMessageOptions are used.

# MQDistributionListItem class

This class encapsulates the MQOR, MQRR, and MQPMR structures and associates them with an owning distribution list.

## Properties:

- AccountingToken property
- AccountingTokenHex property
- CompletionCode property
- CorrelationId property
- CorrelationIdHex property
- DistributionList property
- Feedback property
- GroupId property
- GroupIdHex property
- MessageId property
- MessageIdHex property
- NextDistributionListItem property
- PreviousDistributionListItem property
- QueueManagerName property
- QueueName property
- ReasonCode property
- ReasonName property

### Methods:

- ClearErrorCodes method

### Creation:

Use the AddDistributionListItem Method of the MQDistributionList class

## AccountingToken property

Read-write. The AccountingToken to be included in the MQPMR of a message when put on a queue. Its initial value is all nulls.

**Defined in:**
  MQDistributionListItem class

**Data Type:**
  String of 32 characters

**Syntax:** To get: *accountingtoken$ = MQDistributionListItem*.**AccountingToken**

To set: *MQDistributionListItem*.**AccountingToken** = *accountingtoken$*

## AccountingTokenHex property

Read-write. The AccountingToken to be included in the MQPMR of a message when put on a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B" and so on.

You must supply 64 valid hexadecimal characters.

Its initial value is "0...0".

**Defined in:**
        MQDistributionListItem class

**Data Type:**
        String of 64 hexadecimal characters reqpresenting 32 ASCII characters.

**Syntax:** To get: *accountingtokenh$ = MQDistributionListItem*.**AccountingTokenHex**

To set: *MQDistributionListItem*.**AccountingTokenHex** = *accountingtokenh$*

## CompletionCode property

Read-only. The completion code set by the last open or put request issued against the owning distribution list object.

**Defined in:**
        MQDistributionListItem class

**Data Type:**
        Long

**Values:**
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *completioncode$ = MQDistributionListItem*.**CompletionCode**

## CorrelationId property

Read-write. The CorrelId to be included in the MQPMR of a message when put on a queue. Its initial value is all nulls.

**Defined in:**
        MQDistributionListItem class

**Data Type:**
        String of 24 characters

**Syntax:** To get: *correlid$ = MQDistributionListItem*.**CorrelationId**

To set: *MQDistributionListItem*.**CorrelationId** = *correlid$*

# CorrelationIdHex property

Read-write. The CorrelId to be included in the MQPMR of a message when put on a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters "6" and "1" represent the single character "A", the pair of characters "6" and "2" represent the single character "B" and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is "0..0".

**Defined in:**
MQDistributionListItem class

**Data Type:**
String of 48 hexadecimal characters representing 24 ASCII characters.

**Syntax:** To get: *correlidh$ = MQDistributionListItem*.**CorrelationIdHex**

To set: *MQDistributionListItem*.**CorrelationIdHex** = *correlidh$*

# DistributionList property

Read-only. The distribution list with which this distribution list item is associated.

**Defined in:**
MQDistributionListItem class

**Data Type:**
MQDistributionList

**Syntax:** To get: *set distributionlist = MQDistributionListItem*.**DistributionList**

# Feedback property

Read-write. The Feedback value to be included in the MQPMR of a message when put on a queue.

**Defined in:**
MQDistributionListItem class

**Data Type:**
Long

**Values:**
See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *feedback& = MQDistributionListItem*.**Feedback**

To set: *MQDistributionListItem*.**Feedback** = *feedback&*

## GroupId property

Read-write. The GroupId to be included in the MQPMR of a message when put on a queue. Its initial value is all nulls.

**Defined in:**
    MQDistributionListItem class

**Data Type:**
    String of 24 characters

**Syntax:** To get: *groupid$ = MQDistributionListItem*.**GroupId**

To set: *MQDistributionListItem*.**GroupId** = *groupid$*

## GroupIdHex property

Read-write. The GroupId to be included in the MQPMR of a message when put on a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters ″6″ and ″1″ represent the single character ″A″, the pair of characters ″6″ and ″2″ represent the single character ″B″ and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is ″0..0″.

**Defined in:**
    MQDistributionListItem class

**Data Type:**
    String of 48 hexadecimal characters reqpresenting 24 ASCII characters.

**Syntax:** To get: *groupidh$ = MQDistributionListItem*.**GroupIdHex**

To set: *MQDistributionListItem*.**GroupIdHex** = *groupidh$*

## MessageId property

Read-write. The MessageId to be included in the MQPMR of a message when put on a queue. Its initial value is all nulls.

**Defined in:**
    MQDistributionListItem class

**Data Type:**
    String of 24 characters

**Syntax:** To get: *messageid$ = MQDistributionListItem*.**MessageId**

To set: *MQDistributionListItem*.**MessageId** = *messageid$*

## MessageIdHex property

Read-write. The MessageId to be included in the MQPMR of a message when put on a queue.

Every two characters of the string represent the hexadecimal equivalent of a single ASCII character. For example, the pair of characters ″6″ and ″1″ represent the single character ″A″, the pair of characters ″6″ and ″2″ represent the single character ″B″ and so on.

You must supply 48 valid hexadecimal characters.

Its initial value is ″0..0″.

**Defined in:**
MQDistributionListItem class

**Data Type:**
String of 48 hexadecimal characters representing 24 ASCII characters.

**Syntax:** To get: *messageidh$ = MQDistributionListItem*.**MessageIdHex**

To set: *MQDistributionListItem*.**MessageIdHex** = *messageidh$*

## NextDistributionListItem property

Read-only. The next distribution list item object associated with the same distribution list.

**Defined in:**
MQDistributionListItem class

**Data Type:**
MQDistributionListItem

**Syntax:** To get: *set distributionlistitem = MQDistributionListItem*.**NextDistributionListItem**

## PreviousDistributionListItem property

Read-only. The previous distribution list item object associated with the same distribution list.

**Defined in:**
MQDistributionListItem class

**Data Type:**
MQDistributionListItem

**Syntax:** To get: *set distributionlistitem = MQDistributionListItem*.**PreviousDistributionListItem**

## QueueManagerName property

Read-write. The WebSphere MQ queue manager name.

**Defined in:**
MQDistributionListItem class

**Data Type:**
String of 48 characters.

**Syntax:** To get: *qmname$ = MQDistributionListItem*.**QueueManagerName**

To set: *MQDistributionListItem*.**QueueManagerName** = *qmname$*

## QueueName property

Read-write. The WebSphere MQ queue name.

**Defined in:**
MQDistributionListItem class

**Data Type:**
String of 48 characters.

**Syntax:** To get: *qname$ = MQDistributionListItem*.**QueueName**

To set: *MQDistributionListItem*.**QueueName** = *qname$*

## ReasonCode property

Read-only. The completion code set by the last open or put issued to the owning distribution list object.

**Defined in:**
MQDistributionListItem class

**Data Type:**
Long

**Values:**
See the WebSphere MQ Application Programming Reference
- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

**Syntax:** To get: *reasoncode& = MQDistributionListItem*.**ReasonCode**

## ReasonName property

Read-only. The symbolic name for the ReasonCode. For example
"MQRC_QMGR_NOT_AVAILABLE".

**Defined in:**
MQDistributionListItem class

**Data Type:**
String

**Values:**
See the WebSphere MQ Application Programming Reference

**Syntax:** To get: *reasonname$ = MQDistributionListItem*.**ReasonName**

## ClearErrorCodes method

Resets the CompletionCode to MQCC_OK and the ReasonCode to MQRC_NONE for both the MQDistributionListItem class and the MQSession class.

**Defined in:**
MQDistributionListItem class

**Syntax:** Call *MQDistributionListItem*.**ClearErrorCodes**

# Chapter 4. Troubleshooting

This chapter explains the trace facility provided and details common pitfalls, with help to avoid them, in the following sections:

- "Using trace"
- "When your WebSphere MQ Automation Classes for ActiveX script fails" on page 97
- "Reason codes" on page 98

## Code level tool

You may be asked by the IBM® Service Team which level of code you have installed.

To find this out, run the 'MQAXLEV' utility program.

From the command prompt, change to the directory containing the MQAX200.dll or add the full path length and enter:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

where MQAXLEV.OUT is the name of the output file.

If you do not specify an output file, the detail is displayed on the screen.

### Example output file from code level tool

```
5639-B43 (C) Copyright IBM Corp. 1996, 1998. ALL RIGHTS RESERVED.
    ***** Code Level is 5.1 ***** lib/mqole/mqole.cpp, mqole, p000, p000 L981119 1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216       1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212       1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212        1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212      1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212      1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212      1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219       1.11 99/02/18 12:12:59
```

## Using trace

MQAX includes a trace facility to help the service organization identify what is happening when you have a problem. It shows the paths taken when you run your MQAX script. Unless you have a problem, you are recommended to run with tracing set off to avoid any unnecessary overheads on your system resources.

There are three environment variables that you set to control trace:

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

You should be aware that specifying *any* value for OMQ_TRACE switches the trace facilty on. Even if you set OMQ_TRACE to OFF, trace is still active.

To switch trace off, do not specify a value for OMQ_TRACE.

1. Click the Start button
2. Choose Settings
3. Choose Control Panel
4. Open the System object
5. Click on the Environment tab
6. In the section titled "User Variables for (username)" enter the variable name and a valid value in the correct boxes
7. Click the Set button
8. Close the System object
9. Close the Control Panel window

When deciding where you want the trace files written, ensure that you have sufficient authority to write to, not just read from, the disk.

With tracing switched on, it slows down the running of the MQAX, but it does not affect the performance of your ActiveX or WebSphere MQ environments. When you no longer need a trace file, you can delete it.

You must stop MQAX running to change the status of the OMQ_TRACE variable.

## Trace filename and directory

The trace file name takes the form OMQnnnnn.trc, where nnnnn is the id of the ActiveX process running at the time.

| Command | Effect |
|---|---|
| SET OMQ_TRACE_PATH = drive:\directory | Sets the trace directory where the trace file will be written. |
| SET OMQ_TRACE_PATH = | Removes the OMQ_PATH environment variable the current working directory (when ActiveX is started) is used. |
| ECHO %OMQ_TRACE_PATH% | Displays the current setting of the trace directory on Windows. |
| SET OMQ_TRACE = xxxxxxxx | This sets tracing ON. You switch tracing on by putting one or more characters after the '=' sign. For example: SET OMQ_TRACE=yes SET OMQ_TRACE=no. In both of these examples, tracing will be set ON. This is only effective for a single window/session |
| SET OMQ_TRACE= | Sets tracing OFF |
| ECHO %OMQ_TRACE% | Displays the contents of the environment variable on Windows. |
| SET | Displays the contents of all the environment variables on Windows. |
| SET OMQ_TRACE_LEVEL = 9 | Sets the trace level to 9. Values greater than 9 do not produce any additional information in the trace file. |

# Example trace file

This is a sample trace file from the WebSphere MQ Automation Classes
for ActiveX product. It has been edited and annotated for clarity.
Comments have been added to illustrate its contents and are enclosed
between bands of dashes.
```
******************************************************************

Trace for program C:.\PROGRAM FILES\MICROSOFT VISUAL BASIC\VB32.EXE
                   ---- WebSphere MQ ActiveX ----
started at Wed Aug 06 08:59: 30 1997
@(!)     ***** Code Level is 2.0.0 *****
  ! BuildDate Aug  5 1997
  ! Trace Level is 2


-------------------------------------------------------------------
The head of the trace gives details of when the trace was
started, and the build level of the code. These details may
be required by IBM Service in order to help problem resolution.
The trace level is also shown (2 in this case) and may be
controlled by use of the OMQ_TRACE_LEVEL environment variable.
-------------------------------------------------------------------


(97161)@08:59:30.720
  -->xxxInitialize


-------------------------------------------------------------------
Every 40 entries (excluding text or data entries) there will be a
timestamp preceded in brackets by a 5 digit number representing
the current thread within the process.
Entries beginning --> show entry into a section of internal code
the number of dashes indicates the depth within the code. Entries
beginning <-- show return from a section of code.
-------------------------------------------------------------------


---->ObtainSystemCP
! Code page is 850

<----ObtainSystemCP (rc= OK)
! Attempting to find xlat path via Registry
! XLAT_PATH (stored in xihConvPath) is h:\convert

! XLAT_PATH is h:\convert
! Successfully opened CCSID.TBL under path - h:\convert\CCSID.TBL -


-------------------------------------------------------------------
Entries beginning ! are text entries placed in the code by the
programmer as an aid to problem determination.
-------------------------------------------------------------------


<--xxxInitialize (rc= OK)

-->MQSession:initialize

<--MQSession:initialize (rc= OK)

-->MQSession:initialize

<--MQSession:initialize (rc= OK)

-->MQSession:ccessQueueManager
!  ++ object e11bb0

---->GlobalLock

<----GlobalLock (rc= OK)
```

```
---->MQQueueManager::initialize

<----MQQueueManager::initialize (rc= OK)

---->ImqQueueManager::connect

<----ImqQueueManager::connbsp;  ---->ImqObject::open

<----ImqObject&:open (rc= OK)

---->GlobalUnlock

<----GlobalUnlock (rc= OK)

<--MQSession::AccessQueueManager (rc= OK)
!  -- object e11bb0

-->MQQueueManager::AccessQueue
!  ++ object e11d28

---->GlobalLock

<----GlobalLock (rc= OK)
! parm# 1 is char* 'SYSTEM.DEFAULT.LOCAL.QUEUE'
! parm# 2 is 17.000000
! parm# 3 is variant type 10
! parm# 4 is variant type 10
! parm# 5 is variant type 10

(97161)@08::59::47.750
---->ImqObject::open

<----ImqObject::open (rc= OK)

---->GlobalUnlock

<----GlobalUnlock (rc= OK)
!  -- object e11d28

<--MQQueueManager::AccessQueue (rc= OK)

-->MQSession::AccessMessage
!  ++ object e11bb0

---->MQMessage::MQMessage() <<< Constructor

<----MQMessage::MQMessage() (rc= OK)

<--MQSession::AccessMessage (rc= OK)
!  -- object e11bb0

-->MQMessage::SetCharacterSet
!  long value to set is 850
!  ++ object e11f60
!  -- object e11f60

<--MQMessage::SetCharacterSet (rc= OK)

-->MQMessage::SetMessageData
!  long value to set is 5724228
!  ++ object e11f60

---->MQMessage::WriteString
!  ++ object e11f60
! WriteString malloc, outstring = 0xe12104, outstrlen = 380
!  -- object e11f60
```

```
<----MQMessage::WriteString (rc= OK)
! -- object e11f60

<--MQMessage::SetMessageData (rc= OK)

-->MQMessage::SetMessageId
! BSTR value to set is as BSTR
!  ++ object e11f60
!  -- object e11f60

<--MQMessage::SetMessageId (rc= OK)

-->MQSession::AccessPutMessageOptions
!  ++ object e11bb0

<--MQSession::AccessPutMessageOptions (rc= OK)
!  -- object e11bb0

-->MQueue::Put
!  ++ object e11e48

---->GetObjectFromVariantOptional
! optVar type is 9
! optVar type is VT_DISPATCH

<----GetObjectFromVariantOptional (rc= 66)
!  ++ object e11f60
!  ++ object e12104

---->ImqQueue::put

<----ImqQueue::put (rc= OK)
!  -- object e11f60
!  -- object e12104
!  -- object e11e48

<--MQueue::Put (rc= OK)

-->MQSession::AccessMessage
!  ++ object e11bb0

---->MQMessage::MQMessage() <<< Constructor

<----MQMessage::MQMessage() (rc= OK)

<--MQSession::AccessMessage (rc= OK)
!  -- object e11bb0

-->MQMessage::SetMessageId
! BSTR value to set is as BSTR
!  ++ object e12300
!  -- object e12300

<--MQMessage::SetMessageId (rc= OK)

-->MQSession::AccessGetMessageOptions
!  ++ object e11bb0

<--MQSession::AccessGetMessageOptions (rc= OK)
! -- object e11bb0

-->MQueue::Get
!  ++ object e11e48
! parm# 2 is variant type 9
! parm# 3 is variant type 10

---->GetObjectFromVariantOptional
```

```
! optVar type is 9
!  optVar type is VT_DISPATCH

<----GetObjectFromVariantOptional (rc= 66)

---->GetObjectFromVariantOptional
! optVar type is a
! optional optVar not specified

<----GetObjectFromVariantOptional (rc= 67)
!  ++ object e12300
!  ++ object e121d0

---->ImqQueue::get

<----ImqQueue::get (rc= OK)
!  -- object e12300
!  -- object e121d0
!  -- object e11e48

<--MQueue::Get (rc= OK)

(97161)@08::59::47.970
-->MQMessage::ReadString
!  ++ object e12300

---->ConvertStrToDefault
! Readstring instrlen = 95
! string before conv::
  0000 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 : !"#¢%&'()*+,-./0
  0010 31 32 00 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 : 12.456789:;<=>?@
  0020 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 : ABCDEFGHIJKLMNOP
  0030 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 : QRSTUVWXYZœ\⅛_`
  0040 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 : abcdefghijklmnop
  0050 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 1C    : qrstuvwxyz{}}'.
-------------------------------------------------------------------
The entry above shows an example of a data area dump within the
trace where the contents of a piece of memory is displayed
in hex and ascii.
-------------------------------------------------------------------
! Character set conversion from 850 to 1200, rc = 0
! string after conv:
  0000 21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00 : !.".#.¢.%.&;'.(.
  0010 29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 : ).*.+.,.-../.0.
  0020 31 00 32 00 00 00 34 00 35 00 36 00 37 00 38 00 : 1.2..4.5.6.7.8.
  0030 39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 : 9.:..;.<=>.?.@.
  0040 41 00 42 00 43 00 44 00 45 00 46 00 47 00 48 00 : A.B.C.D.E.F.G.H.
  0050 49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00 50 00 : I.J.K.L.M.N.O.P.
  0060 51 00 52 00 53 00 54 00 55 00 56 00 57 00 58 00 : Q.R.S.T.U.V.W.X.
  0070 59 00 5A 00 5B 00 5C 00 5D 00 5E 00 5F 00 60 00 : Y.Z.œ.\.Ù.⅛._.`.
  0080 61 00 62 00 63 00 64 00 65 00 66 00 67 00 68 00 : a.b.c.d.e.f.g.h.
  0090 69 00 6A 00 6B 00 6C 00 6D 00 6E 00 6F 00 70 00 : i.j.k.l.m.n.o.p.
  00A0 71 00 72 00 73 00 74 00 75 00 76 00 77 00 78 00 : q.r.s.t.u.v.w.x.
  00B0 79 00 7A 00 7B 00 7C 00 7D 00 7E 00 7F 00       : y.z.{.³.}.'..

<----ConvertStrToDefault (rc= OK)
! Input length was 95, output length was 190
!  -- object e12300

<--MQMessage::ReadString (rc= OK)

-->MQSession::destruct

<--MQSession::destruct (rc= OK)

-->MQQueueManager::destruct
```

```
                    <--MQQueueManager::destruct (rc= OK)

                    -->ImqQueueManager::disconnect

                    ---->ImqObject::close

                    <----ImqObject::close (rc= OK)

                    ---->ImqObject::close

                    <----ImqObject::close (rc= OK)

                    ---->ImqQueueManager:backout

                    <----ImqQueueManager:backout (rc= OK)

                    ---->gmqdyn0a:MQDISC
                    ! >>>HConn..
                        0000 01 00 00 00
                         : ....

                    -->MQMessage::OnFinalRelease()

                    ---->MQMessage::'MQMessage() <<< Destructor

                    <----MQMessage::'MQMessage() (rc= OK)

                    <--MQMessage::OnFinalRelease() (rc= OK)

                    -->MQMessage::OnFinalRelease()

                    ---->MQMessage::'MQMessage() <<< Destructor

                    <----MQMessage::'MQMessage() (rc= OK)

                    <--MQMessage::OnFinalRelease() (rc= OK)
```

# When your WebSphere MQ Automation Classes for ActiveX script fails

If your WebSphere MQ Automation Classes for ActiveX script fails, there are a
number of sources of information.

## First failure symptom report

Independently of the trace facility, for unexpected and internal errors, a First
failure symptom report may be produced.

This report is found in a file named OMQnnnnn.fdc, where nnnnn is the number
of the ActiveX process that is running at the time. You find this file in the working
directory from which you started ActiveX or in the path specified in the
OMQ_PATH environment variable.

## Other sources of information

WebSphere MQ provides various error logs and trace information, depending on
the platform involved. See your Winodws NT application event log.

# Reason codes

The following reason codes can occur in addition to those documented for the WebSphere MQ MQI. For other codes, refer to your WebSphere MQ application event log.

| Reason code | Explanation |
| --- | --- |
| MQRC_LIBRARY_LOAD_ERROR (6000) | One or more of the WebSphere MQ libraries could not be loaded. Check that all WebSphere MQ libraries are in the correct search path on the system you are using. For example, make sure that the directories containing the WebSphere MQ libraries are in PATH. |
| MQRC_CLASS_LIBRARY_ERROR (6001) | One of the WebSphere MQ classlibrary calls returned an unexpected ReasonCode/CompletionCode. Check the First Failure Symptom Report for details. Take note of the last method/property and class being used and inform IBM Support of the problem. |
| MQRC_STRING_LENGTH_TOO_BIG (6002) | An attempt has been made to write a UTF format string with a length greater than 65,535 bytes to the message buffer. |
| MQRC_WRITE_VALUE_ERROR (6003) | A value is used that is out of range; for example msg.WriteByte (240). |
| MQRC_PACKED_DECIMAL_ERROR (6004) | An attempt has been made to read a packed decimal number from the message buffer but the data at the data pointer is not in a valid packed data format. |
| MQRC_FLOAT_CONVERSION_ERROR (6005) | An attempt has been made to read a single or double floating point number from the message buffer but the data at the data pointer is not in an appropriate floating point format. |
| MQRC_REOPEN_EXCL_INPUT_ERROR (6100) | An open object does not have the correct **OpenOptions** and requires one or more additional options. An implicit reopen is required but closure has been prevented. Set the **OpenOptions** explicitly to cover all eventualities so that implicit reopening is not required. Closure has been prevented because the queue is open for exclusive input and closure would present a window of opportunity for others potentially to gain access to the queue. |
| MQRC_REOPEN_INQUIRE_ERROR (6101) | An open object does not have the correct OpenOptions and requires one or more additional options. An implicit reopen is required but closure has been prevented. Set the OpenOptions explicitly to include MQOO_INQUIRE. Closure has been prevented because one or more characteristics of the object need to be checked dynamically prior to closure, and the OpenOptions do not already include MQOO_INQUIRE. |

| Reason code | Explanation |
|---|---|
| MQRC_REOPEN_SAVED_CONTEXT_ERR (6102) | An open object does not have the correct OpenOptions and requires one or more additional options. An implicit reopen is required but closure has been prevented. Set the OpenOptions explicitly to cover all eventualities so that implicit reopening is not required. Closure has been prevented because the queue is open with MQOO_SAVE_ALL_CONTEXT, and a destructive Get has been performed previously. This has caused retained state information to be associated with the open queue and this information would be destroyed by closure. |
| MQRC_REOPEN_TEMPORARY_Q_ERROR (6103) | An open object does not have the correct **OpenOptions** and requires one or more additional options. An implicit reopen is required, but closure has been prevented. Set the **OpenOptions**explicitly to cover all eventualities so that implicit reopening is not required. Closure has been prevented because the queue is a local queue of the definition type MQQDT_TEMPORARY_DYNAMIC, which would be destroyed by closure. |
| MQRC_ATTRIBUTE_LOCKED (6104) | An attempt has been made to change the value or attribute of an object while that object is open. Certain attributes, such as **AlternateUserId**, cannot be changed while an object is open. |
| MQRC_CURSOR_NOT_VALID (6105) | The browse cursor for an open queue has been invalidated since it was last used by an implicit reopen. Set the OpenOptions explicitly to cover all eventualities so that implicit reopening is not required. |
| MQRC_ENCODING_ERROR (6106) | The encoding of the next message item needs to be MQENC_NATIVE for reading. |
| MQRC_STRUCID_ERROR (6107) | The structure of the id for the next message item, which is derived from the 4 characters beginning at the data pointer, is either missing or is inconsistent with the type of variable into which the item is being read. |
| MQRC_NULL_POINTER (6108) | A null pointer has been supplied where a non-null pointer is either required or implied. This may be caused by using explicit declarations for WebSphere MQ objects used from VBA as parameters to calls (for example dim msg as Object is ok, dim msg as MqMessage can cause problems). For example, in Excel, with q defined and set dim msg as MqMessageq.put msg gives reasonCode MQRC_NULL_POINTER. It operates correctly from VisualBasic. |
| MQRC_NO_CONNECTION_REFERENCE (6109) | The **MQQueue** object has lost its connection to the **MQQueueManager**. This will occur if the **MQQueueManager**is disconnected. Delete the **MQQueue** object. |

| Reason code | Explanation |
|---|---|
| MQRC_NO_BUFFER (6110) | No buffer is available. For an **MQMessage** object, one cannot be allocated, denoting an internal inconsistency in the object state that should not occur. |
| MQRC_BINARY_DATA_LENGTH_ERROR (6111) | The length of the binary data is inconsistent with the length of the target attribute. Zero is a correct length for all attributes. 24 is a correct length for a **CorrelationId** and for a **MessageId** 32 is a correct length for an **AccountingToken** |
| MQRC_BUFFER_NOT_AUTOMATIC (6112) | A user-defined and managed buffer cannot be resized. Because message buffers are system managed, this indicates an internal inconsistency. |
| MQRC_INSUFFICIENT_BUFFER (6113) | There is insufficient buffer space available after the data pointer to accommodate the request. This could be because the buffer cannot be resized. |
| MQRC_INSUFFICIENT_DATA (6114) | There is insufficient data after the data pointer to accommodate the read request. Reduce the buffer to the correct size and read the data again. |
| MQRC_DATA_TRUNCATED (6115) | Data has been truncated when copying from one buffer to another. This could be because the target buffer cannot be resized, or because there is a problem addressing one or other buffer, or because a buffer is being downsized with a smaller replacement. |
| MQRC_ZERO_LENGTH (6116) | A zero length has been supplied where a positive length is either required or implied. |
| MQRC_NEGATIVE_LENGTH (6117) | A negative length has been supplied where a zero or positive length is required. |
| MQRC_NEGATIVE_OFFSET (6118) | A negative offset has been supplied where a zero or positive offset is required. |
| MQRC_INCONSISTENT_FORMAT (6119) | The format of the next message item is inconsistent with the type of variable into which the item is being read. |
| MQRC_INCONSISTENT_OBJECT_STATE (6120) | There is an inconsistency between this object, which is open, and the referenced MQQueueManager object, which is not connected. |
| MQRC_CONTEXT_OBJECT_NOT_VALID (6121) | The MQPutMessageOptions context reference does not reference a valid MQQueue object. The object has been previoulsy destroyed. |
| MQRC_CONTEXT_OPEN_ERROR (6122) | The MQPutMessageOptions context reference references an MQQueue object that could not be opened to establish a context. This may be because the MQQueue object has inappropriate open options. Inspect the referenced object reason code to establish the cause. |
| MQRC_STRUC_LENGTH_ERROR (6123) | The length of an internal data structure is inconsistent with its content. For an MQRMH, the length is insufficient to contain the fixed fields and all offset data. |

| Reason code | Explanation |
|---|---|
| MQRC_NOT_CONNECTED (6124) | A method failed because a required connection to a queue manager was not available, and a connection cannot be established implicitly. |
| MQRC_NOT_OPEN (6125) | A method failed because a WebSphere MQ object was not open, and opening cannot be accomplished implicitly. |
| MQRC_DISTRIBUTION_LIST_EMPTY (6126) | An MQDistributionList failed to open because there are no MQDistributionList Item objects in the distribution list.<br><br>Corrective action: Add at least one MQDistributionListItem object to the distribution list. |
| MQRC_INCONSISTENT_OPEN_OPTIONS (6127) | A method failed because the object is open, and the open options are inconsistent with the required operation.<br><br>Corrective action: Open the object with appropriate open options and retry. |
| MQRC_WRONG_VERSION (6128) | A method failed because a version number specified or encountered is either incorrect or not supported. |

# Chapter 5. ActiveX interface to the MQAI

For a brief overview of COM interfaces and their use in the MQAI, see "COM and ActiveX scripting" on page 1.

The MQAI enables applications to build and send Programmable Command Format (PCF) commands without directly obtaining and formatting the variable length buffers required for PCF. For more information about the MQAI, see the WebSphere MQ Programmable Command Formats and Administration Interface. The MQAI ActiveX MQBag class encapsulates the data bags supported by the MQAI in a way that is possible to use in any language that supports the creation of COM objects; for example, Visual Basic, C++, Java, and other ActiveX scripting clients.

The MQAI ActiveX interface is for use with the MQAX classes that provide a COM interface to the MQI. For more information about the MQAX classes, see "Using WebSphere MQ Automation Classes for ActiveX" on page 3.

The ActiveX interface provides a single class called MQBag. This class is used to create MQAI data bags and its properties and methods are used to create and work with data items within each bag. The MQBag Execute method sends the bag data to a WebSphere MQ queue manager as a PCF message and collects the replies.

The PCF message is sent to the queue manager object specified, optionally using specified request and reply queues. Replies are returned in a new MQBag object. The full set of commands and replies is described in WebSphere MQ Programmable Command Formats and Administration Interface. Commands can be sent to any queue manager in the WebSphere MQ network by selecting the appropriate request and reply queues.

This chapter discusses the following:
- "The MQBag class"
- "MQBag properties"
- "MQBag methods" on page 106
- "Error handling" on page 111

## The MQBag class

The class, MQBag, is used to create MQBag objects as required. When instantiated, the MQBag class returns a new MQBag object reference.

Create a new MQBag object in Visual Basic as follows:
```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

## MQBag properties

The properties of MQBag objects are explained over the following pages.

# Item property

## Purpose

The Item property represents an item in a bag. It is used to set or inquire about the value of an item. Use of this property corresponds to the following MQAI calls:

- "mqSetString"
- "mqSetInteger"
- "mqInquireInteger"
- "mqInquireString"
- "mqInquireBag"

in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Item (***Selector***,** *ItemIndex***,** *Value***)**

## Parameters

*Selector* **(VARIANT) – input**

Selector of the item to be set or inquired.

When inquiring about an item, MQSEL_ANY_USER_SELECTOR is the default. When setting an item, MQIA_LIST or MQCA_LIST is the default.

If the `Selector` is not of type long, MQRC_SELECTOR_TYPE_ERROR results.

This parameter is optional.

*ItemIndex* **(LONG) – input**

This value identifies the occurrence of the item of the specified selector that is to be set or inquired on. MQIND_NONE is the default.

This parameter is optional.

*Value* **(VARIANT) – input/output**

The value returned or the value to be set. When inquiring about an item, the return value can be of type long, string, or MQBag. However, when setting an item, the value must be of type long or string; if not, MQRC_ITEM_VALUE_ERROR results.

**Note:** You must enter a value for either the `Selector` or `ItemIndex` parameter; if one is not present, MQRC_PARAMETER_MISSING results. Item Property is the default property for the MQBag class, so does not need to be explicitly coded.

## Visual Basic Language Invocation

When inquiring about a value of an item within a bag:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

For MQBag references:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

To set the value of an item in a bag:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

# Count property

## Purpose

The Count property represents the number of data items within a bag. This property corresponds to the MQAI call, "mqCountItems," in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Count (***Selector***,** *Value***)**

## Parameters

*Selector* **(VARIANT) – input**
Selector of the data items to be included in the count.

MQSEL_ALL_USER_SELECTORS is the default.

If the `Selector` is not of type long, MQRC_SELECTOR_TYPE_ERROR is returned.

*Value* **(LONG) – output**
The number of items in the bag included by the *Selector*.

## Visual Basic Language Invocation

To return the number of items in a bag:

```
ItemCount = mqbag.Count([Selector])
```

# Options property

## Purpose

The Options property sets options for the use of a bag. This property corresponds to the `Options` parameter of the MQAI call, "mqCreateBag," in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Options (***Options***)**

## Parameters

*Options* **(LONG) – input/output**
The bag options.

**Note:** The bag options must be set *before* data items are added to or set within the bag. If the options are changed when the bag is not empty, MQRC_OPTIONS_ERROR results. This applies even if the bag is subsequently cleared.

### Visual Basic Language Invocation

When inquiring about the options of an item within a bag:

```
Options = mqbag.Options
```

To set an option of an item in a bag:

```
mqbag.Options = Options
```

## MQBag methods

The methods of the MQBag objects are explained over the following pages.

## Add method

### Purpose

The Add method adds a data item to a bag. This method corresponds to the MQAI calls, "mqAddInteger"and "mqAddString," in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**Add (***Value*, *Selector*)

### Parameters

*Value* **(VARIANT) – input**
> Integer or string value of the data item.

*Selector* **(VARIANT) – input**
> Selector identifying the item to be added.
>
> Depending on the type of `Value`, MQIA_LIST or MQCA_LIST is the default. If the `Selector` parameter is not of type long, MQRC_SELECTOR_TYPE_ERROR results.

### Visual Basic Language Invocation

To add an item to a bag:

```
mqbag.Add(Value,[Selector])
```

## AddInquiry method

### Purpose

The AddInquiry method adds a selector specifying the attribute to be returned when an administration bag is sent to execute an INQUIRE command. This method corresponds to the MQAI call, "mqAddInquiry," in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**AddInquiry** *(Inquiry)*

### Parameters

*Inquiry* **(LONG) – input**
> Selector of the WebSphere MQ attribute to be returned by the INQUIRE administration command.

### Visual Basic Language Invocation

To use the AddInquiry method:

```
mqbag.AddInquiry(Inquiry)
```

## Clear method

### Purpose

The Clear method deletes all data items from a bag. This method corresponds to the MQAI call, ″mqClearBag,″, in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**Clear**

### Visual Basic Language Invocation

To delete all data itmes from a bag:

```
mqbag.Clear
```

## Execute method

### Purpose

The Execute method sends an administration command message to the command server and waits for any reply messages. This method corresponds to the MQAI call, ″mqExecute,″, in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**Execute** *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag)*

### Parameter

*QueueManager* **(MQQueueManager) – input**
> The queue manager to which the application is connected.

*Command* **(LONG) – input**
> The command to be executed.

*OptionsBag* **(MQBag) – input**
> The bag containing options that affect the processing of the call.

*RequestQ* **(MQQueue) – input**
> The queue on which the administration command message will be placed.

*ReplyQ* **(MQQueue) – input**
> The queue on which any reply messages are received.

*ReplyBag* (**MQBag**) – **output**
>    A bag reference containing data from reply messages.

### Visual Basic Language Invocation

To send an administration command message and wait for any reply messages:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

## FromMessage method

### Purpose

The FromMessage method loads data from a message into a bag. This method corresponds to the MQAI call, "mqBufferToBag," in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**FromMessage** *(Message, OptionsBag)*

### Parameters

*Message* (**MQMessage**) – **input**
>    The message containing the data to be converted.

*OptionsBag* (**MQBag**) – **input**
>    Options to control the processing of the call.

### Visual Basic Language Invocation

To load data from a message into a bag:

```
mqbag.FromMessage(Message,[OptionsBag])
```

## ItemType method

### Purpose

The ItemType method returns the type of the value in a specified item in a bag. This method corresponds to the MQAI call, "mqInquireItemInfo," in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**ItemType** *(Selector, ItemIndex, ItemType)*

### Parameters

*Selector* (**VARIANT**) – **input**
>    Selector identifying the item to be inquired.
>
>    MQSEL_ANY_USER_SELECTOR is the default. If the `Selector` parameter is not of type long, MQRC_SELECTOR_TYPE_ERROR results.

*ItemIndex* (**LONG**) – **input**
>    Index of items to be inquired.
>
>    MQIND_NONE is the default.

*ItemType* **(LONG) – output**
> Datatype of the specified item.
>
> **Note:** Either the `Selector` parameter, `ItemIndex` parameter, or both must be specified. If neither parameter is present, MQRC_PARAMETER_MISSING results.

### Visual Basic Language Invocation

To return the type of a value:
```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

# Remove method

## Purpose

The Remove method deletes an item from a bag. This method corresponds to the MQAI call, "mqDeleteItem," in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Remove** *(Selector, ItemIndex)*

## Parameters

*Selector* **(VARIANT) – input**
> Selector identifying the item to be deleted.
>
> MQSEL_ANY_USER_SELECTOR is the default. If the `Selector` parameter is not of type long, MQRC_SELECTOR_TYPE_ERROR results.

*ItemIndex* **(LONG) – input**
> Index of the item to be deleted.
>
> MQIND_NONE is the default.
>
> **Note:** Either the `Selector` parameter, `ItemIndex` parameter, or both must be specified. If neither parameter is present, MQRC_PARAMETER_MISSING results.

## Visual Basic Language Invocation

To delete an item from a bag:
```
mqbag.Remove([Selector],[ItemIndex])
```

# Selector method

## Purpose

The Selector method returns the selector of a specified item within a bag. This method corresponds to the MQAI call, "mqInquireItemInfo," in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Selector** *(Selector, ItemIndex, OutSelector)*

### Parameters

*Selector* **(VARIANT) – input**
> Selector identifying the item to be inquired.
>
> MQSEL_ANY_USER_SELECTOR is the default. If the Selector parameter is not of type long, MQRC_SELECTOR_TYPE_ERROR results.

*ItemIndex* **(LONG) – input**
> Index of the item to be inquired.
>
> MQIND_NONE is the default.

*OutSelector* **(VARIANT) – output**
> Selector of the specified item.
>
> **Note:** Either the Selector parameter, ItemIndex parameter, or both must be specified. If neither parameter is present, MQRC_PARAMETER_MISSING results.

### Visual Basic Language Invocation

To return the selector of an item:
```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

## ToMessage method

### Purpose

The ToMessage method returns a reference to an MQMessage object. The reference contains data from a bag. This method corresponds to the MQAI call, "mqBagToBuffer," in the WebSphere MQ Programmable Command Formats and Administration Interface.

### Format

**ToMessage** *(OptionsBag, Message)*

### Parameters

*OptionsBag* **(MQBag) – input**
> A bag containing options that control the processing of the method.

*Message* **(MQMessage) – output**
> An MQMessage object reference containing data from the bag.

### Visual Basic Language Invocation

To use the ToMessage Method:
```
Set Message = mqbag.ToMessage([OptionsBag])
```

# Truncate method

## Purpose

The Truncate method reduces the number of user items in a bag. This method corresponds to the MQAI call, "mqTruncateBag," in the WebSphere MQ Programmable Command Formats and Administration Interface.

## Format

**Truncate** *(ItemCount)*

## Parameters

*ItemCount* **(LONG) – input**
  The number of user items to remain in the bag after truncation has occurred.

## Visual Basic Language Invocation

To reduce the number of user items in a bag:

```
mqbag.Truncate(ItemCount)
```

# Error handling

If an error is detected during an operation on an MQBag object, including those errors returned to the bag by an underlying MQAX or MQAI object, an error exception is raised. The MQBag class supports the COM ISupportErrorInfo interface so the following information is available to your error handling routine:

- Error number: this is composed of the WebSphere MQ reason code for the error detected and a COM facility code. The facility field, as standard for COM, indicates the area of responsibility for the error. For errors detected by WebSphere MQ it is always FACILITY_ITF.

- Error source: this identifies the type and version of the object that detected the error. For errors detected during MQBag operations this is always MQBag.MQBag1.

- Error description: this is the string giving the symbolic name for the WebSphere MQ reason code.

How you access the error information depends on your scripting language; for example, in Visual Basic the information is returned in the Err object and the WebSphere MQ reason code is obtained by subtracting the constant vbObjectError from Err.Number.

**ReasonCode = Err.Number - vbObjectError**

If the MQBag Execute message sends a PCF message and a reply is received, the operation is considered successful although the command sent may have failed. In this case, the reply bag itself contains the completion and error reason codes as described in WebSphere MQ Programmable Command Formats and Administration Interface.

# Chapter 6. Using the Active Directory Service Interfaces (ADSI)

The Active Directory Service Interfaces (ADSI) provide the means for client applications to use a common set of interfaces to communicate with and control any server that implements them. This allows a single client application to configure a number of different servers because it is shielded from API details specific to each server.

WebSphere MQ provides such an implementation for accessing and managing WebSphere MQ resources: the WebSphere MQ namespace. WebSphere MQ administrators and developers can use the ADSI to enumerate and manage the resources within this namespace to develop their own, or use third-party, system-management tools.

Any programming language that supports the COM interface can be used. To control a queue manager it must be configured to accept remote administration, be started, and have an associated listener and command server. See *Understanding ActiveX and OLE* by Microsoft Press for more information about COM interfaces. See also Chapter 5, "ActiveX interface to the MQAI," on page 103.
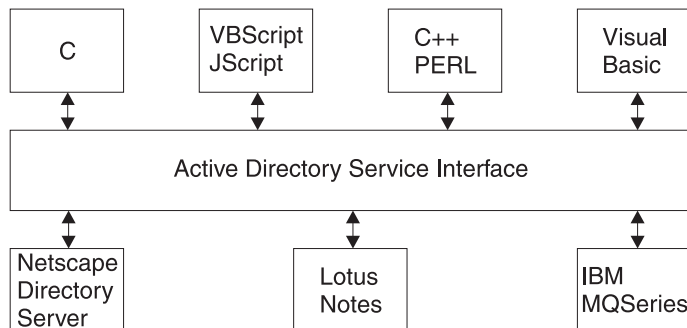


*Figure 2. ADSI and interoperability*

## Prerequisites

To make use of the WebSphere MQ Active Directory Service Interface, you must be using Windows 2000, Windows NT® 5.1 or later versions. Alternatively, use Windows NT Version 4 with Service Pack 3 and the Active Directory Service Interfaces (ADSI) Version 2 applied.

To understand this information, you must understand how to use the Active Directory Service Interfaces because this chapter does not provide such details. This chapter describes the naming scheme used within the WebSphere MQ implementation of these services. A full description of the ADSI can be found in the appropriate Microsoft documentation; for example, the Active Directory Service Interfaces Version 2 documentation available at the

# Troubleshooting

The WebSphere MQ ADSI implementation supports the standard WebSphere MQ diagnostic aids of trace and error logs. See the appropriate documentation for details on starting, stopping, and interpreting trace.

# The WebSphere MQ namespace and object hierarchy

The WebSphere MQ namespace lets you uniquely identify, access, and configure individual objects such as queues and channels. These objects can be represented as an *object hierarchy* within the namespace.

**Note:** In Figure 3 on page 115, object instances are represented within brackets (< ... >) while object identifiers are not.

The object hierarchy is used by the client to reference all objects uniquely. The WebSphere MQ namespace is rooted into the Active Directory namespace (ADS://), and below this point the hierarchy expands. You can access elements within the namespace structure by:

- Using COM or URL addresses. See"Accessing WebSphere MQ objects using COM or URL addresses" on page 115
- Using enumeration. See "Accessing WebSphere MQ objects using enumeration" on page 118.

ADS

IBMMQSeries

MQHost

<HOST>

MQQueueManager

<QUEUE MANAGER>

Schema

Class    Syntax

Property

MQCIntConnChannel — <CHANNEL>

MQClusterReceiverChannel — <CHANNEL>

MQClusterSenderChannel — <CHANNEL>

MQReceiverChannel — <CHANNEL>

MQRequesterChannel — <CHANNEL>

MQSenderChannel — <CHANNEL>

MQServerChannel — <CHANNEL>

MQSvrConnChannel — <CHANNEL>

MQAliasQueue — <QUEUE>

MQClusterQueue — <QUEUE>

MQLocalQueue — <QUEUE>

MQModelQueue — <QUEUE>

MQRemoteQueue — <QUEUE>

MQProcess — <PROCESS>
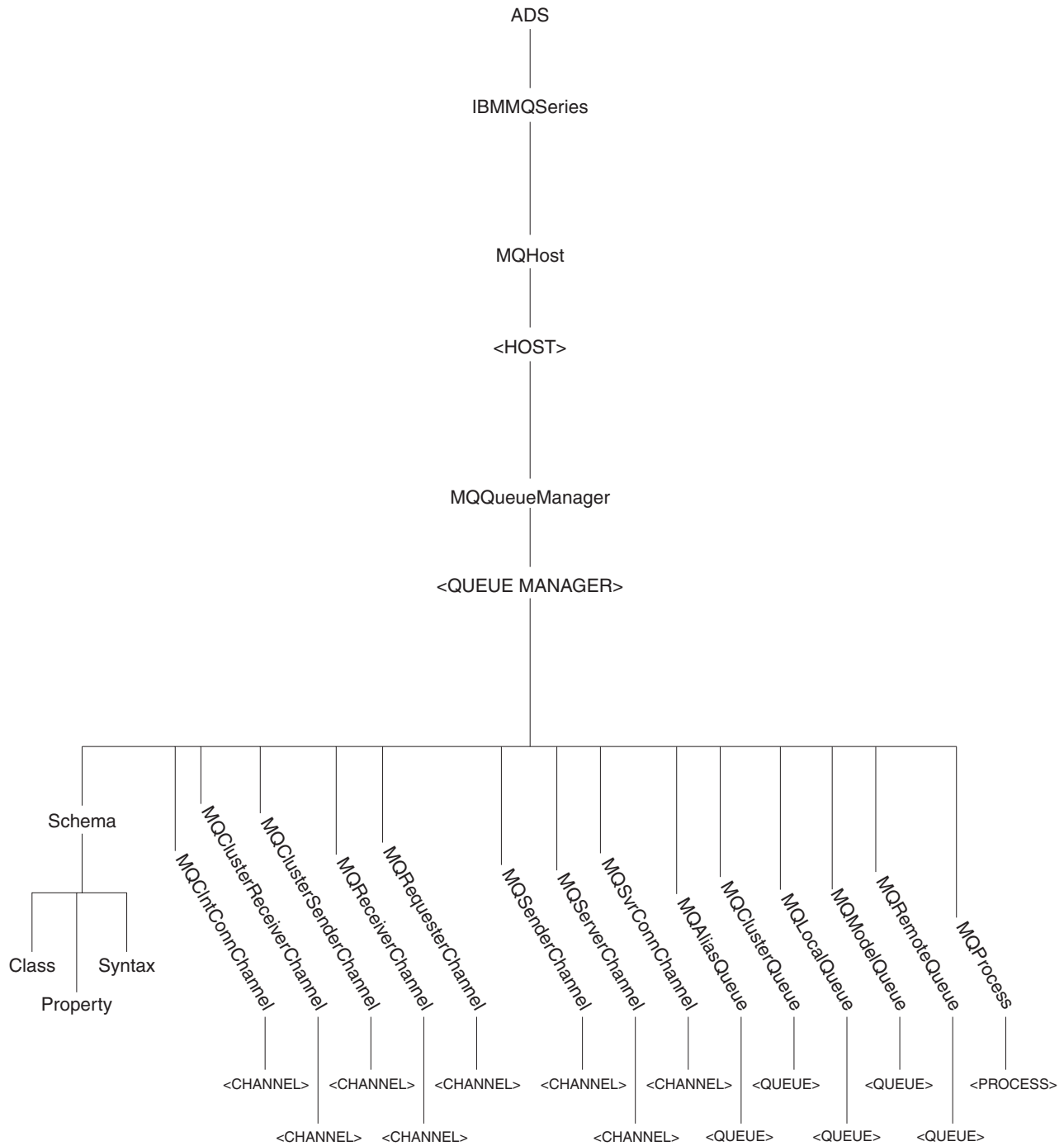
*Figure 3. WebSphere MQ object hierarchy*

## Accessing WebSphere MQ objects using COM or URL addresses

You can gain access to objects within an application namespace by:

1. Calling the ADsGetObject() and ADsOpenObject() functions provided by Microsoft to access the configuration interfaces supported by the specified object.

2. Using the IParseDisplayName interface implemented by the WebSphere MQ Provider Object.

COM or URL format strings are used to identify objects within a namespace. COM strings take the following syntax:

```
@<NamespaceID>!//<ProviderSpecificPath>
```

URL strings take the following syntax:

```
<NamespaceID>://<ProviderSpecificPath>
```

The URL format is recommended.

ADSI stipulates the syntax up to `<NamespaceID>` only; in this case, WebSphere MQ. This part of the object identifier is treated in the same way for both COM and URL format strings. Individual applications are responsible for the `<ProviderSpecificPath>`. For more information about the provider specific path, see "Structuring WebSphere MQ COM and URL addresses" on page 117.

The WebSphere MQ namespace consists of object identifiers and instances, as shown in Figure 3 on page 115. For example, to identify a particular channel the following string is required:

```
WebSphere MQ://MQHost/heron/MQQueueManager/queue.manager.1/MQChannel/
                                              SYSTEM.DEFAULT.SENDER
```

It is possible to shorten identifier strings by following these guidelines:
- When dealing with queue managers on the local machine, it is not necessary to specify `MQHost`.
- When referring to objects residing on the default queue managers of a given machine, it is not necessary to specify `MQQueueManager`.

**Note:** Using shortened identifier strings results in a performance impact, so only use them when strictly necessary.

Thus, it is possible to shorten the example URL. For example, if the `MQHost` instance is not specified in the example above, the default taken is the local host machine. Therefore, to access a channel object for a local queue manager, the following URL address is used:

```
WebSphere MQ://MQQueueManager/queue.manager.1/MQChannel/
                                 SYSTEM.DEFAULT.SENDER
```

In the case where the queue manager is the default, the URL address can be shortened further:

```
WebSphere MQ://MQChannel/SYSTEM.DEFAULT.SENDER
```

## C++ Language invocation

To access the IADs interface on the channel SYSTEM.DEFAULT.SENDER defined on queue manager `queue.manager.1` residing on machine `heron`, use the following:

```
IADs *pMyObject;

ADsGetObject(
TEXT("WebSphere MQ://MQHost/heron/MQQueueManager/queue.manager.1/MQChannel/
                                    SYSTEM.DEFAULT.SENDER")
    , IID_IADs
    , (void**)&pMyObject);
```

# Structuring WebSphere MQ COM and URL addresses

As discussed in "Accessing WebSphere MQ objects using COM or URL addresses" on page 115, ADSI identifier strings consist of a namespace identifier and a provider specific path. TheWebSphere MQ namespace identifier is *WebSphere MQ*. The provider specific path is described below.

It is possible to generate a provider specific path that uniquely identifies any object within the WebSphere MQ namespace using Figure 3 on page 115 as a guide.

WebSphere MQ passes messages between queue managers residing on one or more machines. Queue managers of the same name may reside across an organization but not on the same host machine. Consequently, the first element in the tree, `MQHost`, permits different machines to be identified and hence differentiates between queue managers of the same name. Therefore, the first part of the provider specific path is:

```
MQHost/<Host>
```

Having identified a particular host machine, it is necessary to differentiate the many queue managers that may potentially reside there. This is achieved through the `MQQueueManager` identifier producing a provider specific path of:

```
MQHost/<Host>/MQQueueManager/<QueueManager>
```

Each queue manager supports the following objects:
- "MQClntConnChannel" on page 123
- "MQClusterReceiverChannel" on page 124
- "MQClusterSenderChannel" on page 125
- "MQReceiverChannel" on page 125
- "MQRequesterChannel" on page 126
- "MQSenderChannel" on page 126
- "MQServerChannel" on page 127
- "MQSvrConnChannel" on page 128
- "MQProcess" on page 128
- "MQAliasQueue" on page 129
- "MQClusterQueue" on page 130
- "MQLocalQueue" on page 130
- "MQModelQueue" on page 131
- "MQRemoteQueue" on page 131

These are represented under a given `MQQueueManager` by the following paths:

```
MQHost/Host>/MQQueueManager/<QueueManager>/MQClntConnChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQClusterReceiverChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQClusterSenderChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQReceiverChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQRequesterChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQSenderChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQServerChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQSvrConnChannel/<Channel>
MQHost/Host>/MQQueueManager/<QueueManager>/MQProcess/<Process>
MQHost/Host>/MQQueueManager/<QueueManager>/MQAliasQueue/<Queue>
MQHost/Host>/MQQueueManager/<QueueManager>/MQClusterQueue/<Queue>
MQHost/Host>/MQQueueManager/<QueueManager>/MQLocalQueue/<Queue>
MQHost/Host>/MQQueueManager/<QueueManager>/MQModelQueue/<Queue>
MQHost/Host>/MQQueueManager/<QueueManager>/MQRemoteQueue/<Queue>
```

# Accessing WebSphere MQ objects using enumeration

Objects within the Active Directory contain an *enumeration function* on the IADsContainer interface. This function lists objects contained within the current object. Enumeration can be used to reach any object in the *WebSphere MQ* namespace by traversing down the object hierarchy from a known point. For example, starting from the ADSI root of ADS:

1. From the ADS container, an enumeration request returns all available namespace containers; for example, LDAP, WinNT, and WebSphere MQ.

2. From the WebSphere MQ namespace object, an enumeration request returns MQHost containers, each representing machines that host queue managers.

3. From an MQHost/<Host> instance, an enumeration request returns MQQueueManager containers for each queue manager hosted by that machine.

4. From an MQHost/<Host>/MQQueueManager/<QueueManager> instance, an enumeration request returns MQAliasQueue, MQClntConnChannel, MQClusterQueue, MQClusterReceiverChannel, MQClusterSenderChannel, MQLocalQueue, MQModelQueue, MQProcess, MQReceiverChannel, MQRemoteQueue, MQRequesterChannel, MQSenderChannel, MQServerChannel, MQSvrConnChannel.

   MQChannel, MQProcess and MQQueue objects are represented as containers. Enumerating these objects results in an empty return set, as they contain no child objects.

**Note:** Enumeration is an alternative to using a URL or COM address of the object to be interrogated. However, the process implies the enumeration of several intermediate objects.

## C++ Language invocation

The following code fragment connects to the queue manager called queue.manager.1 on machine heron and obtains an enumeration of its children. Using the enumeration object obtained, the code then displays information about each child before issuing a count of the number of child objects processed:

```
//
// Define and initialize variables.
//

ULONG cElementFetched = 0L;
IEnumVARIANT *pEnumVariant = NULL;
VARIANT VariantArray[MAX_ADS_ENUM];
IADsContainer *pADsContainer = NULL;
DWORD dwObjects = 0, i = 0;
BOOL fContinue = TRUE;

//
// Ensure VARIANT array used to store results is empty
//

for (i = 0; i <MAX_ADS_ENUM; i++)
{
    VariantInit(&VariantArray*lbrk.i]);
}

//
// Attach to the IADs Container interface for the queue manager
// queue.manager.1 residing on machine heron.
//

ADsGetObject( _TEXT("WebSphere MQ://MQHost/heron/MQQueueManager/
```

```
                         queue.manager.1")
              , IID_IADsContainer
              , (void **)&pADsContainer);

//
// Build an enumerator object for the specified Active Directory container
//

ADsBuildEnumerator(pADsContainer, &pEnumVariant);

while (fContinue)
{
    BSTR bstrClass = NULL;
    BSTR bstrName = NULL;
    IADs *pObject;

    //
    // Populate VARIANT array with elements fetched from
    // the enumerator object
    //

    fContinue = ADsEnumerateNext( pEnumVariant
                                , MAX_ADS_ENUM
                                , VariantArray
                                , &pElementFetched);

    //
    // Step through the VARIANT obtaining a pointer to the IADs interface
    // on each object. Using this interface, extract the name and class
    // of the object, printing this information onto the screen.
    //

    for (i= 0; i < cElementFetched; i++ )
    {
        IDispatch *pDispatch = NULL;
        pDispatch=VariantArray[i].pdispVal;
        pDispatch->QueryInterface( IID_IADs
                                 , (VOID **)&pObject);
        pObject->Get_Name (&bstrName);
        pObject->get_Class(&bstrClass)

        printf("  %S(%S)\n", bstrName, bstrClass) ;

        pObject->Release();
        pDispatch->Release();
    }

    memset( VariantArray
          , 0
          , sizeof(VARIANT)*MAX_ADS_ENUM);

    dwObjects += cElementFetched;
}

printf("Total Number of Objects enumerated is %d\n", dwObjects);

if (pEnumVariant)
{
    pEnumVariant->Release();
}

if (pADsContainer)
{
    pADsContainer->Release();
}
```

# Configuring WebSphere MQ Active Directory objects

After a WebSphere MQ object is accessed, its configuration can be modified using the Get and Put functions of the IADs interface. The ADSI objects communicate with underlying queue managers by MQAI COM objects in response to Get and Put requests. For more information about the MQAI see the appropriate help documentation.

Within the IADs interfaces Get and Put functions, it is necessary to refer to properties by names. A list of property names supported by a given class is available through the IADsClass interface on the schema object for that class. To obtain the schema object for a class, call the get_Schema function on its IADs interface.

**Note:** The property names used within the *WebSphere MQ* ADSI implementation are similar to those used within the MQAI COM. For example, to determine the name of a queue manager, call the Get function on its IAD interface passing in a property name of "MQCA_Q_MGR_NAME". Under MQAI you would use the defined MQCA_Q_MGR_NAME.

Here is an example showing the use of the IADs interface to extract the name and description of a queue manager object and printing this information to the screen:

```
//
// Define and initialize variables.
//

VARIANT vDesc;
VARIANT vName;
IADs *pObject = NULL;

//
// Initialize Variants
//
VariantInit(&vDesc);
VariantInit(&vName);
//
// Attach to the IADs interface for the queue manager queue.manager.1
// residing on machine heron.
//

ADsGetObject( _TEXT("WebSphere MQ://MQHost/heron/MQQueueManager/
                     queue.manager.1")
            , IID_IADs
            , (void **)&pObject);

//
// Using the IADs interface extract the name and description of
// the queue manager printing this information to the screen.
//

pObject->Get(_TEXT("MQCA_Q_MGR_NAME"),&vName);
pObject->Get(_TEXT("MQCA_Q_MGR_DESC"),&vDesc);

printf("  %S,%S",vName.bstrVal, vDesc.bstrVal);

pObject->Release();
```

# Object descriptions

This section describes the objects contained within the WebSphere MQ object hierarchy:

- "WebSphere MQ"
- "MQHost" on page 122
- "MQQueueManager" on page 122
- "Schema" on page 123
- "MQClntConnChannel" on page 123
- "MQClusterReceiverChannel" on page 124
- "MQClusterSenderChannel" on page 125
- "MQReceiverChannel" on page 125
- "MQRequesterChannel" on page 126
- "MQSenderChannel" on page 126
- "MQServerChannel" on page 127
- "MQSvrConnChannel" on page 128
- "MQProcess" on page 128
- "MQAliasQueue" on page 129
- "MQClusterQueue" on page 130
- "MQLocalQueue" on page 130
- "MQModelQueue" on page 131
- "MQRemoteQueue" on page 131

# WebSphere MQ

The WebSphere MQ container represents the namespace presented by WebSphere MQ. All other object types are contained within WebSphere MQ. Enumerating the default container (ADS) provides access to all installed namespaces; for example, WinNT, LDAP and WebSphere MQ.

Alternatively, the container object (*WebSphere MQ* Namespace Object) may be instantiated directly by the CoCreateInstance call.

Enumerating *WebSphere MQ* provides a list of MQHost containers which are described in the next section.

## Type

ADSI namespace container object.

## Interfaces

The following interfaces are available:

- IADs
- IADsContainer
- IADsOpenDSObject
- IDispatch
- IUnknown

For more information about these interfaces, see the

### Properties

None.

## MQHost

The MQHost object represents a machine that hosts WebSphere MQ queue managers. Enumerating this object provides a list of queue managers supported by the machine, each represented by an MQQueueManager object. For more information about the MQQueueManager object, see "MQQueueManager."

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the

### Properties

None.

## MQQueueManager

The MQQueueManager object represents a queue manager hosted by a particular machine. Enumerating the MQQueueManager object provides access to the items it contains (MQClntConnChannel, MQClusterReceiverChannel, MQClusterSenderChannel, MQReceiverChannel, MQRequesterChannel, MQSenderChannel, MQServerChannel, MQSvrConnChannel, MQProcess, MQAliasQueue, MQClusterQueue, MQLocalQueue, MQModelQueue, and MQRemoteQueue).

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQQueueManager class, see the
WebSphere MQ Programmable Command Formats and Administration Interface.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to
represent the property names used. The names found in the WebSphere MQ
Programmable Command Formats and Administration Interface should be
provided as a string between quotes (' ').

## Schema

The schema object contains all the ADSI Schema class objects, one for every type of
object that can be created in the *WebSphere MQ* directory. The object also provides
access to the ADSI property objects and the ADSI syntax objects.

### Type

ADSI schema container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsClass
- IDispatch
- IUnknown

Several schema objects exist within the WebSphere MQ namespace. You will be
directed to the correct one for a particular object by calling the:
`get_Schema()`

function on its IADs interface.

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

None.

## MQClntConnChannel

The MQClntConnChannel class represents an individual client connection channel
on a particular queue manager. Enumerating an object of this class results in an
empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
* IADs
* IADsContainer
* IDirectoryObject
* IDispatch
* IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQClntConnChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface guide should be provided as a string between quotes (' ').

## MQClusterReceiverChannel

The MQClusterReceiverChannel class represents an individual cluster receiver channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
* IADs
* IADsContainer
* IDirectoryObject
* IDispatch
* IUnknown

### Properties

For details about the properties supported by the MQClusterReceiverChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQClusterSenderChannel

The MQClusterSenderChannel class represents an individual cluster sender channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

#### Interfaces

- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at

`http://www.microsoft.com/`

#### Properties

For details about the properties supported by the MQClusterSenderChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface.

**Note:** The properties are not provided in the header files; they must be entered as a string between quotes (' ').

## MQReceiverChannel

The MQReceiverChannel class represents an individual receiver channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

#### Interfaces

The following interfaces are available:

- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at

`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQReceiverChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQRequesterChannel

The MQRequesterChannel class represents an individual requester channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:

- IADs
- IADsContainer
- IDIrectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at

http://www.microsoft.com/

### Properties

For details about the properties supported by the MQRequesterChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQSenderChannel

The MQSenderChannel class represents an individual sender channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQSenderChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQServerChannel

The MQServerChannel class represents an individual server channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQServerChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQSvrConnChannel

The MQSvrConnChannel class represents an individual server connection channel on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQSvrConnChannel class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQProcess

The MQProcess class of objects represents an individual process definition defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject

- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQProcess class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQAliasQueue

The MQAliasQueue class represents an individual alias queue defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQAliasQueue class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQClusterQueue

The MQClusterQueue class represents an individual cluster queue defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at:

`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQClusterQueue class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQLocalQueue

The MQLocalQueue class represents an individual local queue defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

### Interfaces

The following interfaces are available:

- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at

`http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQLocalQueue class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQModelQueue

The MQModelQueue class represents an individual model queue defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI Container Object.

### Interfaces

The following interfaces are available:
- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at `http://www.microsoft.com/`

### Properties

For details about the properties supported by the MQModelQueue class, see the WebSphere MQ Programmable Command Formats and Administration Interfacebook.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

## MQRemoteQueue

The MQRemoteQueue class represents an individual remote queue defined on a particular queue manager. Enumerating an object of this class results in an empty list.

### Type

ADSI container object.

## Interfaces

The following interfaces are available:

- IADs
- IADsContainer
- IDirectoryObject
- IDispatch
- IUnknown

For more information about these interfaces, see the Microsoft MSDN Web site at
`http://www.microsoft.com/`

## Properties

For details about the properties supported by the MQRemoteQueue class, see the WebSphere MQ Programmable Command Formats and Administration Interface book.

**Note:** Unlike other WebSphere MQ administration interfaces, ADSI uses strings to represent the property names used. The names found in the WebSphere MQ Programmable Command Formats and Administration Interface book should be provided as a string between quotes (' ').

# Chapter 7. About the WebSphere MQ Automation Classes for ActiveX Starter samples

This appendix describes the WebSphere MQ Automation Classes for ActiveX Starter samples, and explains how to use them.

WebSphere MQ for Windows provides the following Visual Basic sample programs:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

These samples run on Visual Basic 4 or Visual Basic 5. You will find them in the directory ...\tools\mqax\samples\vb.

In the same directory you will also find samples for Microsoft Excel and html. These are:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

**Note:** If using Visual Basic 5, you **must** select and install Visual Basic component grid32.ocx.

## What is demonstrated in the samples

The samples demonstrate how to use WebSphere MQ Automation Classes for ActiveX to:

- Connect to a queue manager
- Access a queue
- Put a message on a queue
- Get a message from a queue

The central part of the Visual Basic sample is shown on the following pages.

### Preparing to run the samples

To run any of the samples you need one of the following depending on which of the samples you intend to run.

- Microsoft Visual Basic Version 4 (or later)
- Microsoft Excel 95 (or later)
- A Web browser

You will also need:

- A WebSphere MQ queue manager running.
- A WebSphere MQ queue already defined.

## Error handling in the samples

Most of the samples provided in the WebSphere MQ Automation Classes for ActiveX package exhibit little or no error handling. For more information about error handling, see "Error handling" on page 8.

## Running the ActiveX Starter samples

Before you run the WebSphere MQ Automation Classes for ActiveX Starter samples check that you have a default queue manager running and that you have created the required queue definitions. For details of creating and running a queue manager and creating a queue, refer to WebSphere MQ System Administration Guide. The sample uses the queue SYSTEM.DEFAULT.LOCAL.QUEUE which should be defined on any normally set up WebSphere MQ server.

## MQAX Starter samples for Microsoft Visual Basic Version 4 or later

This section explains how to run the MQAX starter samples for Microsoft Basic Version 4 or later.

### Running the MQAXTRIV sample
1. Start the queue manager.
2. In Windows Explorer or File Manager, select the icon for the sample, MQAXTRIV.VBP (Visual Basic Project file) and open the file.

   The Visual Basic program starts and opens the file, MQAXTRIV.VBP.
3. In Visual Basic, press function key 5 (F5) to run the sample.
4. Click anywhere in the window form, "MQAX trivial tester".

If everything is working correctly, the window background should change to green. If there is a problem with your setup, the window background should change to red and error information will be displayed.

The central part of the Visual Basic sample is shown below.

```
Option Explicit

Private Sub Form_Click()

'*******************************************************************************
'* This simple example illustrates how to put and get a WebSphere MQ message to
'* and from a WebSphere MQ message queue. The data from the message returned by the
'*get is read and compared with that from the original message.
'*******************************************************************************
Dim MQSess As MQSession                   '* session object
Dim QMgr As MQQueueManager                '* queue manager object
Dim Queue As MQQueue                      '* queue object
Dim PutMsg As MQMessage                   '* message object for put
Dim GetMsg As MQMessage                   '* message object for get
Dim PutOptions As MQPutMessageOptions     '* get message option

Dim GetOptions As MQGetMessageOptions     '* put message options
Dim PutMsgStr As String                   '* put message data string
Dim GetMsgStr As String                   '* get message data string
'*******************************************************************
'* Handle errors
'*******************************************************************
On Error GoTo HandleError
```

```
'*******************************************************************************
'* Initialize the current position for the form
'*******************************************************************************
CurrentX = 0
CurrentY = 0


'*******************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*******************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                         MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)


'*******************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*******************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions


'*******************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*******************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*******************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*******************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127)  '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:       """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*******************************************************************************
'* Handle errors
'*******************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer
```

```
                Cls
                BackColor = RGB(255, 0, 0)  '* set to red for error
                Print "An error occurred as follows:"
                Print ""
                If MQSess.CompletionCode <> MQCC_OK Then
                    ErrMsg = Err.Description
                    StrPos = InStr(ErrMsg, " ")          '* search for first blank
                    If StrPos > 0 Then
                        Print Left(ErrMsg, StrPos)       '* print offending MQAX object name
                    Else
                        Print Error(Err)                 '* print complete error object
                    End If
                    Print ""
                    Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
                    Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
                    Print "(" & MQSess.ReasonName & ")"
                Else
                    Print "Visual Basic error:  " & Err
                    Print Error(Err)
                End If

                Exit Sub

                End Sub
```

## Starting the MQAXCLSS sample

This sample allows you to browse properties and methods of queue managers and queue objects.

1. Start the queue manager.
2. Open the file, MQAXCLSS.VBP, by double clicking on the document icon in Windows Explorer or by choosing File - Open from the file menu in Visual Basic.
3. Start the sample.
4. Enter the appropriate queue manager and queue names then click the corresponding buttons.

## The MQAXDLST sample

The Visual Basic MQAXDLST sample demonstrates the use of a distribution list to send the same message to two queues with one put. To run the sample, do the same as for the MQAXCLSS sample above.

## MQAX Starter sample for Microsoft Excel 95 or later

This section explains how to run the MQAX starter sample for Microsoft Excel 95 or later, MQAXTRIV.XLS.

### Running the MQAXTRIV.XLS sample

1. Start the queue manager.
2. In Explorer or File Manager, select the icon for the MQAX sample MQAXTRIV.XLS.
3. Click on the button.
4. The screen will be updated with a success (or failure) message.

### Running the Bank demonstration with MQAX.XLS

1. Start the queue manager.
2. Run the WebSphere MQ MQSC command file, BANK.TST. This sets up the necessary WebSphere MQ queue definitions.

   To find out how to use an MQSC command file, refer to the WebSphere MQ System Administration Guide book.
3. Run MQAXBSRV.VBP. This is the server, simulating a back-end application. This has to run in conjunction with Microsoft Excel.
4. Run MQAX.XLS. This is the client WebSphere MQ demonstration.
5. Select a customer from the drop-down list box.
6. Click on the Submit button.

After a short time, (3 seconds or so) the fields should become populated with values and you should see a bar chart appear.

## Starter sample using an ActiveX compatible WWW browser

**Note:** To run this sample, you must be running an ActiveX compatible Web browser. Microsoft Internet Explorer (but not Netscape Navigator) is a compatible Web browser.

### Running the HTML sample

This sample demonstrates how you can invoke MQAX from both VBScript and JavaScript™.

1. Start the queue manager.
2. Open the file, "MQAXTRIV.HTM", in your ActiveX compatible Web browser.

   You can do this either by double-clicking the file icon in Windows Explorer or you can choose File - Open from the File menu of your ActiveX compatible Web browser.
3. Follow the instructions on the screen.

# Notices

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

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

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

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

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

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

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

IBM
System/360
System/390
WebSphere

Intel, Intel logo, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

# Index

## A

About WebSphere MQ automation classes
  for ActiveX classes   11
AccessGetMessageOptions method   15
Accessing WebSphere MQ objects using
  COM or URL addresses   115
  C++ Language invocation   116
Accessing WebSphere MQ objects using
  enumeration   118
  C++ Language invocation   118
AccessPutMessageOptions method   15
AccessQueue method   27
AccessQueueManager method   15
AccountingToken property   52, 84
AccountingTokenHex property   53, 85
Add method   106
  Selector parameter   106
  Value parameter   106
AddDistributionList method   28
AddDistributionListItem method   82
Addinquiry method   106
AddInquiry method
  Inquiry parameter   107
ADSI (Active Directory Service
  Interface   113
ADSI Troubleshooting   114
AlternateUserId property   18, 32, 80
ApplicationIdData property   53
ApplicationOriginData property   53
applications that access non-ActiveX
  applications   3
    using WebSphere MQ automation
      classes for ActiveX   3
AuthorityEvent property   18

## B

Backout method   28
BackoutCount property   54
BackoutRequeueName property   32
BackoutThreshold property   32
BaseQueueName property   33
Begin method   28
BeginOptions property   19

## C

ChannelAutoDefinition property   19
ChannelAutoDefinitionEvent
  property   19
ChannelAutoDefinitionExit property   20
character set conversion   7
CharacterSet property   20, 54
Clear method   107
ClearErrorCodes method
  MQDistributionList class   83
  MQDistributionListItem class   89
  MQGetMessageOptions class   79
  MQMessage class   63
  MQPutMessageOptions class   76

ClearErrorCodes method *(continued)*
  MQQueue class   45
  MQQueueManager class   28
  MQSession class   16
ClearMessage method   63
Close method   46, 83
CloseOptions property   20, 33, 80
code level tool   91
COM and ActiveX scripting   1
CommandInputQueueName
  property   20
CommandLevel property   20
commands using Windows   92
Commit method   29
CompletionCode property
  MQDistributionList class   80
  MQDistributionListItem class   85
  MQGetMessageOptions class   77
  MQMessage class   50
  MQPutMessageOptions class   74
  MQQueue class   33
  MQQueueManager class   21
  MQSession class   13
Configuring WebSphere MQ Active
  Directory objects   120
Connect method   29
ConnectionHandle property   21
ConnectionReference property   34, 80
ConnectionStatus property   21
ConnectOptions property   21
CorrelationId property   54, 85
CorrelationIdHex property   55, 86
Count property   105
CreationDateTime property   34
CurrentDepth property   34

## D

data conversion   5
DataLength property   51
DataOffset property   51
DeadLetterQueueName property   22
DefaultInputOpenOption property   34
DefaultPersistence property   35
DefaultPriority property   35
DefaultTransmissionQueueName
  property   22
DefinitionType property   35
DepthHighEvent property   35
DepthHighLimit property   35
DepthLowEvent property   36
DepthLowLimit property   36
DepthMaximumEvent property   36
Description property   22, 36
design and programming using
  WebSphere MQ automation classes for
  ActiveX   3
designing applications that access
  non-ActiveX applications   3
Disconnect method   29
DistributionList property   86

DistributionLists property   22
DynamicQueueName property   37

## E

Encoding property   55
environment support   1
error handling   8
Error handling   111
error handling in the samples   134
example output file from code level
  tool   91
example trace file   93
execute method   107
Execute method
  Command parameter   107
  OptionsBag parameter   107
  QueueManager parameter   107
  ReplyBag parameter   108
  ReplyQ parameter   107
  RequestQ parameter   107
Expiry property   56

## F

failure of the WebSphere MQ automation
  classes for ActiveX script   97
Feedback property   56, 86
first failure symptom report   97
FirstDistributionListItem property   81
Format property   57
Frommessage method   108
FromMessage method
  Message parameter   108

## G

Get method   46
GroupId property   57, 87
GroupIdHex property   57, 87

## H

HardenGetBackout property   37
how error handling works   8

## I

InhibitEvent property   23
InhibitGet property   37
InhibitPut property   38
InitiationQueueName property   38
introduction to ActiveX   1
IsConnected property   23
IsOpen property   23, 38, 81
item property   104
ItemIndex parameter
  Item property   104
itemtype method   108

# 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.

IBM.®

Spine information:

IBM

WebSphere MQ

Using the Component Object Model Interface

Version 7.0