

WebSphere MQ

Administrator's Guide

Version 7.0

WebSphere MQ

Administering a Queue Manager

Version 7.0

Note

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

Second edition (January 2009)

This edition of the book applies to the following products:

-

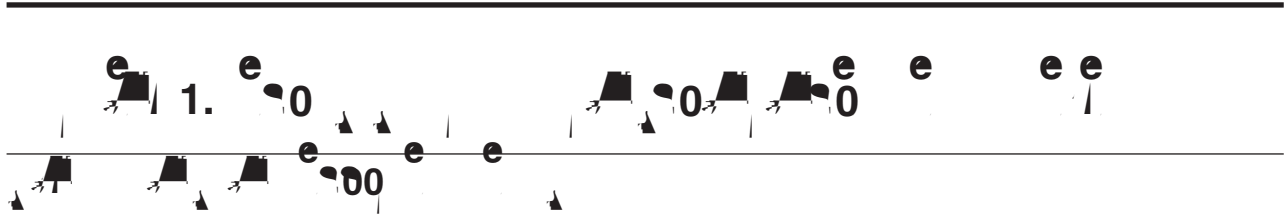
e 0 **A**

e **21**

90



- 1. Boolean operator outcome when logic is A
AND B 41
- 2. Boolean operator outcome when logic is A OR
B 41



The WebSphere® MQ products enable programs to communicate with one another

Queues reside in, and are managed by, a queue manager (see “What is a queue manager?”). The physical nature of a queue depends on the operating system on which the queue manager is running. A queue can either be a volatile buffer area in the memory of a computer, or a data set on a permanent storage device (such as

As soon as you establish even the smallest cluster you will benefit from simplified system administration. Queue managers that are part of a cluster need fewer definitions and so the risk of making an error in your definitions is reduced.

No direct connections between programs



Some of the benefits of message queuing are:

- You can design applications using small programs that you can share between many applications.
- You can quickly build new applications by reusing these building blocks.
- Applications written to use message queuing techniques are not affected by changes in the way that queue managers work.
- You do not need to use any communication protocols. The queue manager deals with all aspects of communication for you.
- Programs that receive messages need not be running at the time that messages are sent to them. The messages are retained on queues.


- WebSphere MQ for z/OS provides support in the IMS environment for online message processing programs (MPPs), interactive fast path programs (IFPs), and batch message processing programs (BMPs). If you are writing batch DL/I



This chapter introduces the design of WebSphere MQ applications, under these headings:

- “Planning your design”
- “Using WebSphere MQ objects” on page 10
- “Designing your messages” on page 11
- “WebSphere MQ techniques” on page 12
- “Application programming” on page 14
- “Testing WebSphere MQ applications” on page 16

These subjects are discussed in greater detail in the remaining chapters of this book.



When you have decided how your applications can take advantage of the platforms and environments available to you, you need to decide how to use the features offered by WebSphere MQ.

Some of the key aspects are:

What types of queue should you use?

Do you want to create a queue each time that you need one, or do you want to use queues that have already been set up? Do you want to delete a queue when you have finished using it, or is it going to be used again? Do you want to use alias queues for application independence? To see what types of queues are supported, refer to “Queues” on page 52.

Should you use shared queues and queue-sharing groups, and should you use queue-sharing group clusters (WebSphere MQ for z/OS only)?

You might want to take advantage of the increased availability, scalability, and workload balancing that are possible when you use shared queues with queue-sharing groups. You might also want to estimate the average and peak message flows and consider using queue-sharing group clusters to spread the workload. See *WebSphere MQ for z/OS Concepts and Planning Guide* for a full discussion of these topics.

Should you use queue manager clusters?

You might want to take advantage of the simplified system administration, and increased availability, scalability, and workload balancing that are possible when you use clusters. See *WebSphere MQ Queue Manager Clusters* for a full discussion of this topic.

What types of message should you use?

You might want to use datagrams for simple messages, but request messages (for which you expect replies) for other situations. You might want to assign different priorities to some of your messages.

Should you use publish/subscribe or point-to-point messaging?

Using publish/subscribe messaging, a sending application sends the information that it wants to share in a WebSphere MQ message to a standard destination managed by WebSphere MQ publish/subscribe, and lets WebSphere MQ handle the distribution of that information. The target application does not have to know anything about the source of the

and receives that information when it is available. For more information about publish/subscribe messaging, see *WebSphere MQ Publish/Subscribe User's Guide*.

Using point-to-point messaging, a sending application sends a message to a specific queue, from where it knows a receiving application will retrieve

- Services
- Listeners
- Process definitions
- Channels
- Storage classes (WebSphere MQ for z/OS only)
-

What type of message should I use?

use. For a more advanced application, you might want to use some of the techniques introduced in the following sections.



A program that is serving a queue can await messages by:

- Waiting until either a message arrives, or a specified time interval expires (see “Waiting for messages” on page 151).
- Setting a signal so that the program is informed when a message arrives (WebSphere MQ for z/OS only). For information about this, see “Signaling” on page 152.
- Establish a call back exit to be driven when a message arrives; see “Asynchronous consumption of WebSphere MQ messages” on page 42.
- Making periodic calls on the queue to see whether a message has arrived (*polling*)

- Coordinate queue manager updates and updates made by other resource managers.

The MQI provides *structures* (groups of fields) with which you supply input to, and get output from, the calls. It also provides a large set of named constants to help you supply options in the parameters of the calls. The definitions of the calls, structures, and named constants are supplied in data definition files for each of the supported programming languages. Also, default values are set within the MQI calls.

You can *discard* a message after an exception has arisen. If you select the discard option, and have requested an exception report message, the report message goes to the `exceptionReportQueue` and `exceptionReportQueue`, and the original message is discarded.

Note: A benefit of this is that you can reduce the number of messages going to the dead-letter queue. However, it does mean that your application, unless it sends

- MQIIH
- MQH*

MQH* means any name that starts with the characters MQH.

The Format name occurs at specific positions for MQDLH and MQXQH, but for the

Set the CONVERT channel attribute to YES if you need the sending message channel agent (MCA) to convert the application data.

call, the Convert characters call, the MQGMO_CONVERT option, and the built-in formats, see the

In general, when you use `MQSETTMP` to set properties

MaxMsgLength attribute of channels going to a system prior to Version 7.0 as necessary, to compensate for the fact that more data might be sent for each message. Alternatively, you can lower the queue or queue manager *MaxMsgLength*, so that the overall level of data being sent around the system remains the same.

There is a length limit of 100 MB for message properties, excluding the message descriptor or extension for each message.

of the selection string is returned in the VSLength field of the MQCHARV. If the

A selector can be passed in on the call to MQSUB by using the `selector` field in the MQSD structure. The effect of passing in a selector on the MQSUB is that only those messages published to the topic being subscribed to, that match a supplied selection string, are made available on the destination queue.

Figure 4 on page 35 shows the process of selection using the MQOPEN call.

A selector can be passed in on the call to MQOPEN by using the `selector` field in the MQOD structure. The effect of passing in a selector on the MQOPEN call is that only those messages on the opened queue, that match a selector, are

an example, if a subscriber specifies a selection string of “a = 3” and a message is

zero-prefixed decimal number like this, for example, '09' returns a syntax error because 9 is not a valid octal digit. Examples of octal numbers are 0177, 0713.

- An exact numeric literal is a numeric value without a decimal point, such as 57, -957, and +62. An exact numeric literal can have a trailing upper or

- Specifying a number that is out of the defined range
- Specifying an arithmetic expression which would cause overflow or underflow

- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3
comparison operator:
 - Age BETWEEN 15 and 19 is equivalent to age >= 15 AND age <= 19.
 - Age NOT BETWEEN 15 and 19 is equivalent to age < 15 OR age > 19.
 - If any of the expressions of a BETWEEN operation are NULL, the value of the operation is false. If any of the expressions of a NOT BETWEEN operation are NULL, the value of the operation is true.
- identifier [NOT] IN (valuee.)-333(If)-333ng-lit333l1alue
NULL, t(comparison)-333(oNOT))-333(IN)-33has-333(BETWEEN)S333ngTJ0-1rthe ve
vT=-3'U33ngT3nF=v

the.–

- Equivalence is tested using a single equals character, for example, “a == b” is incorrect, whereas “a = b” is correct.
- An operator used by many programming languages to represent ‘not-equals’ is ‘!=’. This representation is not a valid synonym for ‘<>’, for example, “a != b” is not valid, whereas “a <> b” is valid.
- Care must be taken to ensure that the correct type of quotes are used to contain selectors. Single quotes are recognized only if the ‘ (U+0039) character is used, not, ` (U+0145), for example is not recognized. Similarly, double quotes are valid only when used to enclose a string, for example, “a” is valid, but “a” is not.

the application by invoking a 'unit of code', identified by the application passing

Figure 6 on page 45 This sample flow shows a single threaded application consuming messages from two queues. The example shows all of the messages being delivered to a single function.

The difference from the asynchronous case is that control does not return to the

e 00 e 1 0

Segmentation is not supported on WebSphere MQ for z/OS.

Logical message

Logical messages within a group are identified by the `id`, `seq` and `type`

a queue manager stops, whether the stoppage is as a result of an operator command or because of the failure of some part of your system. Nonpersistent messages for WebSphere MQ for z/OS stored in a coupling facility (CF) are an exception to this. They persist as long as the CF remains available.

When you create a message, if you initialize the message descriptor (MQMD) using the defaults, the persistence for the message is taken from the `mqmd.msgflags` attribute of the queue specified in the MQOPEN command.

When you use the MQPUT or MQPUT1 call to put a message on a queue, you can

- “Namelists” on page 61
-

- The maximum number of open handles for any one connection

To create a queue you can use WebSphere MQ commands (MQSC), PCF commands, or platform-specific interfaces such as the WebSphere MQ for z/OS operations and control panels.

You can create local queues for temporary jobs *dynamically* from your application. For example, you can create *reply-to* queues (which are not needed after an application ends). For more information, see “Dynamic queues” on page 57.

Before using a queue, you must open the queue, specifying what you want to do with it. For example, you can open a queue for:

-

This means that more than one program can work with the same queue, accessing it using different names. For more information about topic aliases, see *WebSphere MQ Publish/Subscribe User's Guide*.

Model and dynamic queues

A model queue is a template of a queue definition used only when you want to create a dynamic local queue.

You can create a local queue dynamically from a WebSphere MQ program, naming the model queue that you want to use as the template for the queue attributes. At that point you can change some attributes of the new queue. However, you cannot change the *DefinitionType*. If, for example, you require a permanent queue, select a model queue with the definition type set to permanent. Some conversational applications can use dynamic queues to hold replies to their queries because they probably do not need to maintain these queues after they have processed the replies.

Cluster queues

A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster.

The cluster queue manager makes a local queue definition for the queue specifying the name of the cluster that the queue is to be available in. This definition has the effect of advertising the queue to the other queue managers in the cluster. The other queue managers in the cluster can put messages to a cluster queue without needing a corresponding remote-queue definition. A cluster queue can be advertised in more than one cluster. See “What is a cluster?” on page 3 and

Event queues

Event queues hold event messages. These messages are reported by the queue manager or a channel.

These special queues are described in greater detail in the following sections.



Some of the attributes of a queue are specified when the queue is defined, and cannot be changed afterwards (for example, the type of the queue). Other

- In a client-server model, each client must create and use its own dynamic reply-to queue. If a dynamic reply-to queue is shared between more than one client, deleting the reply-to queue might be delayed because there is uncommitted activity outstanding against the queue, or because the queue is in use by another client. Additionally, the queue might be marked as being logically deleted, and inaccessible for subqueue musttionaPI17.9(eply-to)-33(sown)-333(3

of the original message, the reason that the queue manager put the message on the dead-letter queue, and the date and time that it did this.

Applications can also use the queue for messages that they cannot deliver. For more information, see “Using the dead-letter (undelivered message) queue” on page 68.

These queues receive the PCF, MQSC, and CL commands, as supported on your platform, in readiness for the queue manager to action them.

On WebSphere MQ for z/OS the queue is known as the **SYSTEM.COMMAND.INPUT.QUEUE**; on other platforms it is known as the **SYSTEM.ADMIN.COMMAND.QUEUE**. The commands accepted vary by platform. See *WebSphere MQ Programmable Command Formats and Administration*

- *MQI* channels, which are bidirectional, and transfer MQI calls from a WebSphere MQ client to a queue manager, and responses from a queue manager to a WebSphere MQ client.

To refer to a local queue, you can omit the name of the queue manager (by replacing it with blank characters or using a leading null character). However, all

- “Locally determined errors”
- “Using report messages for problem determination” on page 67
- “Remotely determined errors” on page 68

ee e e 11 10

The three most common causes of errors that the queue manager can report immediately are:

- Failure of an MQI call; for example, because a queue is full
-

logical unit of work. For information on RRS syncpoint support see “Transaction management and recoverable resource manager services” on page 219.

i5/OS You can make your MQPUT and MQGET calls within global units of work that are managed by i5/OS commitment control. You can declare syncpoints by using the native i5/OS COMMIT and ROLLBACK commands or the language-specific commands. Local units of work are managed by WebSphere MQ using the MQCMIT and MQBACK calls.

UNIX systems and Windows systems

In these environments, you can make your MQPUT and MQGET calls in the usual way, but you must declare syncpoints by using the MQCMIT and MQBACK calls (see “Committing and backing out units of work” on page 214). In the CICS environment, MQCMIT and MQBACK commands are disabled, because you can make your MQPUT and MQGET calls within units of work that are managed by CICS.

Use persistent messages for carrying all data that you cannot afford to lose. Persistent messages are reinstated on queues if the queue manager has to recover from a failure. With WebSphere MQ on UNIX systems and WebSphere MQ for

On WebSphere MQ for z/OS, to make the backout count survive restarts of the queue manager, set the `MQQA_BACKOUT_HARDENED` attribute to 1; otherwise, if the queue manager has to restart, it does not maintain an accurate backout count for each message. Setting the attribute this way adds the penalty of extra processing.

On WebSphere MQ for i5/OS, WebSphere MQ for Windows, and WebSphere MQ on UNIX systems, the backout count always survives restarts of the queue manager.

Also, on WebSphere MQ for z/OS, when you remove messages from a queue within a unit of work, you can mark one message so that it is *not* made available again if the unit of work is backed out *by the application*. The marked message is treated as if it has been retrieved under a new unit of work. You mark the message if it backout by using the `MQSET` command with the `MQO` option. The backout count is maintained for the message. The backout count is maintained for the message. The backout count is maintained for the message.

Your application must contain procedures that monitor your reply-to queue and process any messages that arrive on it. Remember that a report message can contain all the original message, the first 100 bytes of the original message, or none of the original message.

The queue manager sets the `error` field of the report message to indicate the reason for the error; for example, the target queue does not exist. Your programs should do the same. Your format report message is "R9(eport)-333(messas" on)-33tionRotelar

Your programs can use the dead-letter queue in the same way that the queue manager uses it. You can find the name of the dead-letter queue by opening the queue manager object (using the MQOPEN call) and inquiring about the `DLQ` attribute (using the MQINQ call).

9. 2.

- Calls through which WebSphere MQ for Windows and WebSphere MQ on UNIX systems programs can commit and back out changes.
- *Include files* that define the values of constants supplied on these platforms.
- *Library files* to link your applications.
-

AAprsrIansac8.2(111Tf1.0820TD[(A)manager3(pr)17.9utab(pr1TD35gh)-331TD35gCalls)-33601.42oMC

MQSTAT

Use this call to retrieve status information about previous asynchronous put operations.

The MQI calls are described fully in the *WebSphere MQ Application Programming Reference*



Syncpoint calls are available as follows:

WebSphere MQ for z/OS calls:

WebSphere MQ for z/OS provides the MQCMIT and MQBACK calls.

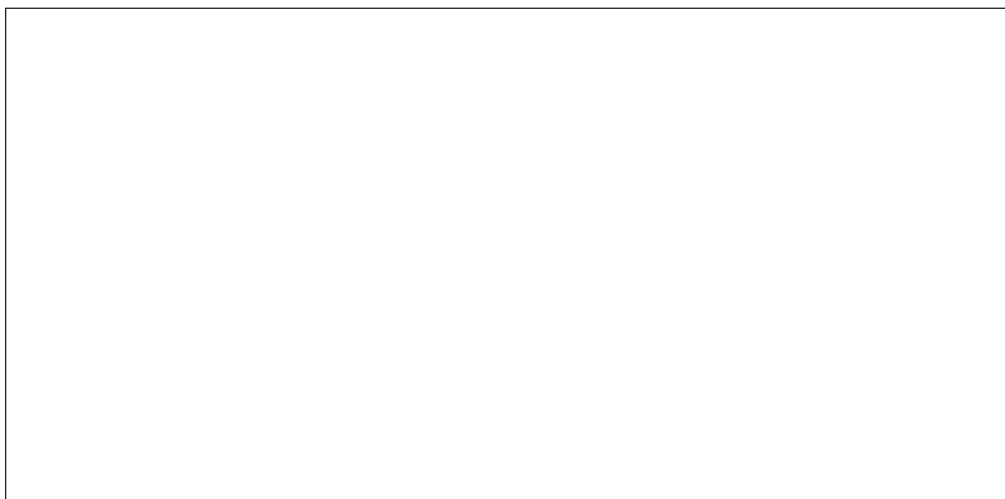
Use these calls in z/OS batch programs to tell the queue manager that all the MQGET and MQPUT operations since the last syncpoint are to be made permanent (committed) or are to be backed out0B-333(T)91.7(o)-333(commit)-333(and)-333(back)-3

The MQXCNCV (convert characters) call converts message character data from one

Before you can run a program written with WebSphere MQ for z/OS, you must

LIBMQMZF_R Installable exits for C

On WebSphere MQ for i5/OS you can write your applications in C++. To see how



WebSphere MQ for HP-UX:

On WebSphere MQ for HP-UX, you must link your program to the MQI library files supplied for the environment in which you are running your application, in addition to those provided by the operating system.

PA-RISC platform:

In a non-threaded application:

libmqcxa_r.so	Client XA interface for C
libmqcxa64_r.so	Client alternative XA interface for C
libimqi23ah_r.so	C++

WebSphere MQ for Linux:

On WebSphere MQ for Linux, you must link your program to the MQI library files supplied for the environment in which you are running your application, in addition to those provided by the operating system.

In a non-threaded application:

libmqm.so	Server for C
libmqic.so	Client for C
libmqmzf.so	Installable service exits for C
libmqmxa.so	Server XA interface for C
libmqmxa64.so	Server alternative XA interface for C
libmqcxa.so	Client XA interface for C
libmqcxa64.so	Client alternative XA interface for C
libimqc23gl.so	Client for C++
libimqs23gl.so	Server for C++

In a threaded application:

libmqm_r.so	Server for C
libmqic_r.so	Client for C
libmqmzf_r.so	Installable service exits for C
libmqmxa_r.so	Server XA interface for C
libmqmxa64_r.so	Server alternative XA interface for C
libmqcxa_r.so	Client XA interface for C
libmqcxa64_r.so	Client alternative XA interface for C
libimqc23gl_r.so	Client for C++
libimqs23gl_r.so	Server for C++

WebSphere MQ for Solaris:

On WebSphere MQ for Solaris, you must link your program to the MQI library files supplied for the environment in which you are running your application in addition to those provided by the operating system.

libmqm.so	Server for C
libmqmzse.so	For C
libmqic.so	Client for C
libmqmcs.so	Common services for C
libmqmzf.so	Installable service exits for C
libmqmxa.so	Server XA interface for C
libmqmxa64.so	Server alternative XA interface for C
libmqcxa.so	Client XA interface for C
libmqcxa64.so	Client alternative XA interface for C
libimqc23as.a	Client for C++
libimqs23as.a	Server for C++

There are two types of parameter common to all the calls: handles and return codes.

All MQI calls use one or more

e



e

to

C.8Mameters that areand of type MQHCONN, MQHOBJ, MQHMSG, orMQLONG are passed by va

- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

To include them, add the keyword EQUONLY=NO when you call the macro.

CMQA is protected against multiple declaration, so you can include it many times. However, the keyword EQUONLY takes effect only the first time that the macro is included.

Specifying the name of a structure:

To allow more than one instance of a structure to be declared, the macro that generates the structure prefixes the name of each field with a user-specifiable string and an underscore character (_).

Specify the string when you invoke the macro. If you do not specify a string, the macro uses the name of the structure to construct the prefix:

```
* Declare two object descriptors
           CMQODA                      Prefix used="MQOD_" (the default)
MY_MQOD CMQODA                      Prefix used="MY_MQOD_"
```

The structure declarations in the *WebSphere MQ Application Programming Reference* show the default prefix.

Specifying the form of a structure:

The macros can generate structure declarations in one of two forms, controlled by the DSECT parameter:

DSECT=YES An assembler-language DSECT instruction is used to start a new data

You can specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message descriptor structure with the `msgtype` field initialized with `MQMT_REQUEST`, and the `replytoq` field initialized with the string `MY_REPLY_TO_QUEUE`, use the following code:

```
MY_MQMD          CMQMDA          MSGTYPE=MQMT_REQUEST,          X
                  REPLYTOQ=MY_REPLY_TO_QUEUE
```

In this book, the parameters of calls, the names of data types, the fields of

UNIX safely allows the setting up of a signal handler for such signals for the



The MQCONN call is similar to the MQCONN call, but includes options to control the way that the call works.

As input to MQCONN, you can supply a queue manager name, or a

MQCNO_SHARED_BINDING is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

MQCNO_ISOLATED_BINDING

Specify this option to make the application and the local queue manager

non-MQI call requiring authentication (for example, opening a file), but you *must* change it back to mqm before making the next MQI call.

- On WebSphere MQ for i5/OS:
 1. Trusted applications must run under the QMQM user profile. It is not sufficient that the user profile be a member of the QMQM group or that the program adopt QMQM authority. It might not be possible for the QMQM user profile to be used to sign on to interactive jobs, or to be specified in the job description for jobs running trusted applications. In this case one approach is to use the i5/OS profile swapping API functions, QSYGETPH, QWTSETP, and QSYRLSPH to temporarily change the current user of the job

For example, the following sequence of activity is possible with a shared Hconn:

1. Thread 1 issues MQCONN and gets a shared Hconn *h1*
- 2.

The MQCLOSE and MQDISC calls usually perform no authority checking.

In the normal course of events a job that has the authority to open or connect to a WebSphere MQ object closes or disconnect from that object. Even if the authority of a job that has connected to or opened a WebSphere MQ object is revoked, the MQCLOSE and MQDISC calls are accepted.

To perform any of the following operations, you must first *open* the relevant WebSphere MQ object:

-

Table 5. Resolving queue names when using MQOPEN (continued)

Input to MQOD		Resolved names		



In the `options` parameter of the `MQOPEN` call, you must choose one or more options to control the access that you are given to the object that you are opening. With these options you can:

- Open a queue and specify that all messages put to that queue must be directed to the same instance of it
- Open a queue to allow you to put messages on it
- Open a queue to allow you to browse messages on it
- Open a queue to allow you to remove messages from it
- Open an object to allow you to inquire about and set its attributes (but you can set the attributes of queues only)
- Open a topic or topic string to publish messages to it
- Associate context information with a message
- Nominate an alternate user identifier to be used
- `S.5iMnEurity(must)-3eckssage`
-

To open an object so that you can inquire about its attributes, use the MQOO_INQUIRE option.

Note: You cannot specify this option when opening a distribution list.

MQOPEN options relating to message context:

If you want to be able to associate context information with a message when you put it on a queue, you must use one of the message context options when you open the queue.

MQOD structure; if the structure is less than Version 3,
MQOO_RESOLVE_LOCAL_Q is ignored without an error being returned.

- A reason code
- The object handle, reset to the value MQHO_UNUSABLE_HOBJ

Descriptions of the parameters of the MQCLOSE call are given in the *WebSphere MQ Application Programming Reference*.

- $e_1 \cdot e_2 = e$
- $e_1 \cdot e_2 = e$
- \dots, e_1, e_2, \dots
- $e_1, e_2, e_3, \dots, e_n - e_1, e_2, e_3, \dots$
- $\dots - e$
- $-e$
- \dots
- $\dots ee$

These fields are described below.

StrucId

This identifies the structure as a put-message options structure. This is a

The MQPMO can also accommodate fields required for distribution lists (see "Distribution lists" on page 121). If you want to use this facility, Version 2 of the MQPMO structure is used. This includes the following fields:

RecsPresent

This field contains the number of queues in the distribution list; that is, the number of Put Q.9fsge 1ecsoed (sQPMOR)-333(acn)-333(conr)17.9(e)spondng fR necsoed (sQPRR)-333(ap)17.9(e)snt

ResponseRecOffset and ResponseRecPtr

You also use pointers and offsets to address the Response Records (see “Using the MQRR structure” on page 124 for further information about Response Records).

Use the `ResponseRecPtr` field to specify a pointer to the first Response Record, or the `ResponseRecOffset` field to specify the offset of the first Response Record. This is the offset from the start of the MQPMO structure. Enter a nonnull value for either `ResponseRecPtr` or `ResponseRecOffset`.

Note: If you are using MQPUT1 to put messages to a distribution list, `ResponseRecPtr` must be null or zero and `ResponseRecOffset` must be zero.

Version 3 of the MQPMO structure additionally includes the following fields:

OriginalMsgHandle

The use you can make of this field depends on the value of the *Action*

The maximum size of the data is determined by:

- The `maxLength` attribute of the queue manager
- The `maxLength` attribute of the queue on which you are putting the message
- The size of any message header added by WebSphere MQ (including the

- Make sure that the `DeadLetterQueue` attribute of the dead-letter queue is set to the same as the `DeadLetterQueue` of the queue manager that owns the dead-letter queue.

The attributes for the queue manager and the message queuing constants are described in the *WebSphere MQ Application Programming Reference*.

For information on how undelivered messages are handled in a distributed queuing environment, see *WebSphere MQ Intercommunications*.

Two message handles are available in the MQPMO structure, *OriginalMsgHandle* and *NewMsgHandle*. The relationship between these message handles is defined by the value of the MQPMO *Action* field.

For full details see the description of the Action field in the *WebSphere MQ Application Programming Reference*. A message handle is not necessarily required in

If you have properties in an MQRFH2 header from a message you have previously

1

Use the `test_auth_uid` field if you want to nominate an alternate user identifier that is to be used to test authority to open the queue.

`mqmd_t`

This is a message descriptor structure (MQMD). As with the MQPUT call, use this structure to define the message that you are putting on the queue.

`mqpmo_t`

This is a put-message options structure (MQPMO). Use it as you would for the MQPUT call (see “Specifying options using the MQPMO structure” on page 112).

When the `test_auth_uid` field is set to zero, the queue manager uses your own user ID when it performs tests for authority to access the queue. Also, the

Figure 9 shows how distribution lists work.



Use the MQOPEN call to open a distribution list, and use the options of the call to specify what you want to do with the list.

As input to MQOPEN, you must supply:

-

Using the MQOR structure:

Provide an MQOR structure for each destination.

The structure contains the destination queue and queue manager names. The `\ .e. _e` and `\ .e. / _e` fields in the MQOD are not used for distribution lists. There must be one or more object records. If the `\ .e. / _e` is left blank, the local queue manager is used. See the *WebSphere MQ Application Programming Reference*

Whichever technique you choose, you must use one of `MQRC_OBJECT_ERROR` and `MQRC_OBJECT_RECORDS_ERROR`; the call fails with reason code `MQRC_OBJECT_RECORDS_ERROR` if both are zero, or both are nonzero.

Using the MQRR structure:

These structures are destination specific; each Response Record contains a `MQRR_DEST` and `MQRR_QNAME` field for each queue of a distribution list. You must use this structure to enable you to distinguish where any problems lie.

For example, if you receive a reason code of `MQRC_MULTIPLE_REASONS` and your distribution list contains five destination queues, you will not know which queues the problems apply to if you do not use this structure. However, if you have a completion code and reason code for each destination, you can locate the errors more easily.

See the *WebSphere MQ Application Programming Reference* for further information about the MQRR structure.

- A connection handle (see “Putting messages on a queue” on page 111 for a description).
-

Figure 13 shows how you can put a message to a distribution list in COBOL.

Using MQPUT1:

If you are using MQPUT1, consider the following:

1. The values of the `MQMD`, `MQMDL`, `MQMDL`, `MQMDL` and `MQMDL`.

Note: If you want to use MQGET more than once (for example, to step through the messages in the queue), you must set the `MSGTRAIL` and `MSGIDC` fields of this

- Except on WebSphere MQ for z/OS, whether complete, logical messages only are retrievable
- Whether messages in a group can be retrieved only when *all* messages in the group are available
- Except on WebSphere MQ for z/OS, whether segments in a logical message can be retrieved only when *all* segments in the logical message are available

If you leave the `MQGMO_NO_WAIT` field set to the default value (MQGMO_NO_WAIT), the MQGET call operates this way:

- If there is no message matching your selection criteria on the queue, the call does not wait for a message to arrive, but completes immediately. Also, in WebSphere MQ for z/OS, the call does not set a signal requesting notification when such a message arrives.
- The way that the call operates with syncpoints is determined by the platform:

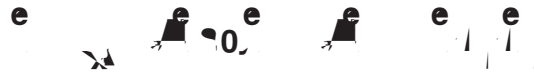
Platform	Under syncpoint control
i5WNo	
UNIX systems	No
zWYes	
Windows systems	No

- On WebSphere MQ for z/OS, the message .009482.933m411.5tha(sycse(s)]TJT*[(W)54.9Jag
- The sele(W)54.9J3(message)-335tha(syc3(.0094mo33m411.f(gr)17.9mon)-333(the)-333(quehe)

MQGMO_PROPERTIES_FORCE_MQRFH2 or MQPROP_FORCE_MQRFH2. You can also use the MQMHBUFF call to convert properties from a message handle to MQRFH2 format.

If you set properties using a message handle, an application connected to an earlier version of Websphere MQ can retrieve them using MQRFH2 headers.

Websphere MQ Version 7.0 clients connected to queue managers at an earlier version can retrieve properties using message handles, although those properties were set using MQRFH2 headers.



In the `mqget` parameter of the MQGET call, specify the size of the buffer area to hold the message data that you retrieve. You decide how big this should be in three ways:

1. You might already know what length of messages to expect from this program.

not know what length of message to expect, you can inquire about the `_msglen` attribute (using the MQINQ call), then specify a buffer of this size.

Try to make the buffer size as close as possible to the actual message size to avoid reduced performance.

For further information about the `_msglen` attribute, see “Increasing the maximum message length” on page 145.

(-)

(-)

These messages appear in the following physical order on the queue:

1. Message A (not in a group)
2. Logical message 1 of group Y
3. Logical message 2 of group Z
4. Logical message 2 of group Y
5. Segment 1 of (last) logical message 3 of group Y
6. (Last) segment 2 of (last) logical message 3 of group Y
7. Logical message 1 of group Z
8. Message B (not in a group)

Note: On WebSphere MQ for z/OS, the physical order of messages on the queue is not guaranteed if the queue is indexed by GROUPID.

When getting messages, you can specify `MQGMO_LOGICAL_ORDER` to retrieve messages in logical rather than physical order.

If you issue an `MQGET` call with `MQGMO_BROWSE_FIRST` and `MQGMO_LOGICAL_ORDER`, subsequent `MQGET` calls with `MQGMO_BROWSE_NEXT` must also specify this option. Conversely, if the `MQGET` with `MQGMO_BROWSE_FIRST` does not specify `MQGMO_LOGICAL_ORDER`, neither must the following `MQGETs` with `MQGMO_BROWSE_NEXT`.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that the queue manager retains for MQGET calls that remove

```
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
```

```
MQCMT
```

The getting application chooses not to start processing any group until all the messages within it have arrived. specify MQGMO_ALL_MSGS_AVAILABLE for the first message in the group; the option is ignored for subsequent messages within the group.

```

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMD.Options = MQGMD_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMD.Options = MQPMD_SYNCPOINT
MQCMT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMD.Options = MQGMD_SYNCPOINT
MQCMT

```



```

/* The next message on the group must be retrieved by matching
   the sequence number and group id with those retrieved from the
   status information. */
GMD.Options = MQGMD_COMPLETE_MSG | MQGMD_SYNCPOINT | MQGMD_WAIT
MQGET GMD.MatchOptions = MQMD_MATCH_GROUP_ID | MQMD_MATCH_MSG_SEQ_NUMBER,
      MQMD.GroupId      = value from Status message,
      MQMD.MsgSeqNumber = value from Status message plus 1
msgs = 1
/* ProR. 9, * [(/*)T1s message */
...

```

```

/* Now normal proR. 9,ing is resumed */
/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

```

```

/* ProR. 9, * [(/*up)-500(to)-500(4)-500(messages)-500(in)-500(the)-500(group)-500(*)]TJ 0 -

```

```

...

```


If you set the `mqmd` field of the MQMD structure to 2, you can use the `mqmd`, `mqmd`, `mqmd`, and `mqmd` fields. Table 8 shows which message is retrieved for the possible settings of these fields.

If you decide that because of these restrictions on MQGET, that a client application design is not suited to read ahead, specify the MQOPEN option MQOO_READ_AHEAD_NO. Alternatively set the default read ahead value of the queue being opened altered to either NO or DISABLED.

By default read ahead is disabled. You can enable read ahead at queue or application level.

To enable read ahead:

-

Value	Description
MSGID	An index of message identifiers is maintained. Use this when retrieving messages using the

This is the simplest scenario, in which one application puts a message to be retrieved by another. The message might be large: not too large for either the putting or the getting application to handle in a single buffer, but too large for the queue manager or a queue on which the message is to be put.

The only changes necessary for these applications are for the putting application to authorize the queue manager to perform segmentation if necessary:

```
PMD.Options = (existing options)
MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

and for the getting application to ask the queue manager to reassemble the message if it has been segmented:

```
GMD.Options = MQGMD_COMPLETE_MSG | (existing options)
MQGET
```

In this simplest scenario, the application must reset the GroupId field to MQGI_NONE before the MQPUT call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might subsequently lead to incorrect processing.

The application buffer must be large enough to contain the reassembled message (unless you include the MQGMO_ACCEPT_TRUNCATED_MSG option).

If data conversion is necessary, the getting application might have to do it by specifying MQGMO_CONVERT. This should be straightforward because the data

If you do not use MQPMO_LOGICAL_ORDER, the application must set the `MQPMO_SEGMENT_LENGTH` and the length of each segment. In this case, logical state is not maintained

cause one object to be transferred. Each message must identify the *logical* length and offset of the object that is to be appended to it. However, in cases where it is not possible to know the total size of the object or the maximum size allowed by the channel, design the sending message exit so that the putting application just puts a single message, and the exit itself puts the next message on the transmission queue when it has appended as much data as it can to the message it has been passed.

Before using this method of dealing with large messages, consider the following:

- The MCA and the message exit run under a WebSphere MQ user ID. The message exit (and therefore, the user ID) needs to access the object to either retrieve it at the sending end or create it at the receiving end; this might only be feasible in cases where the object is universally accessible. This raises a security issue.
- If the reference message with bulk data appended to it must travel through

Here is an example of the use of the `__len__()`, `e` and `__eq__` fields of the MQRMH:

A putting application might put a reference message with:

- No physical data
- `__len__` = 0 (this message represents the entire object)ata

than one program is waiting to browse a message, all the programs can be activated. For more information, see the description of the `wait_time` field of the `MQGMO` structure in the *WebSphere MQ Application Programming Reference*.

If the state of the queue or the queue manager changes before the wait interval expires, the following actions occur:

- If the queue manager enters the quiescing state, and you used the `MQGMO_FAIL_IF QUIESCING` option, the wait is canceled and the `MQGET` call completes with the `MQRC_Q_MGR QUIESCING` reason code. Without this option, the call remains waiting.
- On z/OS, if the connection (for a CICS or IMS application) enters the quiescing state, and you used the `MQGMO_FAIL_IF QUIESCING` option, the wait is canceled and the `MQGET` call completes with the `MQRC_CONN QUIESCING` reason code. Without this option, the call remains waiting.
- If the queue manager is forced to stop, or is canceled, the `MQGET` call completes with either the `MQRC_Q_MGR STOPPING` or the `MQRC_CONNECTION_BROKEN` reason code.
- If the attributes of the queue (or a queue to which the queue name resolves) are changed so that get requests are now inhibited, the wait is canceled and the `MQGET` call completes with the `MQRC_GET_INHIBITED` reason code.
- If the attributes of the queue (or a queue to which the queue name resolves) are changed in such a way that the `FORCE` option is required, the wait is canceled and the `MQGET` call completes with the `MQRC_OBJECT_CHANGED` reason code.

If you want your application to wait on more than one queue, use the signal facility of WebSphere MQ for z/OS (see “Signaling”). For more information about the circumstances in which these actions occur, see the *WebSphere MQ Application Programming Reference*.

3. Specify the address of the *Event Control Block* (ECB) in the , _ field. This

Note:

1. If more than one program has set a signal on the same shared queue to remove a message, only one program is activated by a message arriving. However, if more than one program is waiting to browse a message, all the programs can be activated. The rules that the queue manager follows when deciding which applications to activate are the same as those for waiting applications: for more information, see the description of the `wait` field of the MQGMO structure in the

MQGMO_MARK_SKIP_BACKOUT fail with reason code
MQRC_SECOND_MARK_NOT_ALLOWED.

Note:

1. The marked message skips backout only if the unit of work containing it is terminated by an application request to back it out. If the unit of work is

- The `msglen` specified on the MQGET call is not zero.
- The message data length is not zero.
- The queue manager supports conversion between the

2.

Queues in priority sequence:

The first message in a queue in this sequence is the message that has been on the queue the longest and that has the highest priority at the time that the MQOPEN call is issued.

Use MQGMO_BROWSE_NEXT to read the messages in the queue.

The browse cursor points to the next message, working from the priority of the first message to finish with the message at the lowest priority. It browses any messages put to the queue during this time as long as they are of priority equal to, or lower than, the message identified by the current browse cursor.

Any messages put to the queue of higher priority can be browsed only by:

- Opening the queue for browse again, at which point a new browse cursor is established
- Using the MQGMO_BROWSE_FIRST option

Uncommitted messages:

An uncommitted message is never visible to a browse; the browse cursor skips past it.

Messages within a unit-of-work cannot be browsed until the unit-of-work is committed. Messages do not change their position on the queue when committed, so skipped, uncommitted messages will not be seen, even when they *are* committed, unless you use the MQGMO_BROWSE_FIRST option and work through the queue again.

Change to queue sequence:

If the message delivery sequence is changed from priority to FIFO while there are messages on the queue, the order of the messages that are already queued is not changed. Messages added to the queue subsequently take the default priority of the queue.

Using the queue's index:**Supported only on WebSphere MQ for z/OS.**

When you browse an indexed queue that contains only messages of a single priority (either persistent or nonpersistent or both), the queue manager performs the browse by making use of the index, when any of the following forms of browse are used:

1. If the queue is indexed by MSGID, and the above condition is true, browse requests that pass a MSGID in the MQMD structure are processed using the index to find the target message.
2. If the queue is indexed by CORRELID, and the above condition is true, browse requests that pass a CORRELID in the MQMD structure are processed using the index to find the target message.
3. If the queue is indexed by GROUPID, and the above condition is true, browse requests that pass a GROUPID in the MQMD structure are processed using the index to find the target message.

If an application browses through the various messages of one group (using logical order), it is important that logical order should be followed to reach the start of the next group, because the last message of one group might occur physically *after* the first message of the next group. The MQGMO_LOGICAL_ORDER option ensures that logical order is followed when scanning a queue.

Use MQGMO_ALL_MSGS_AVAILABLE (or MQGMO_ALL_SEGMENTS_AVAILABLE) with care for browse operations. Consider the case of logical messages with MQGMO_ALL_MSGS_AVAILABLE. The effect of this is that a logical message is available only if all the remaining messages in the group are also present. If they are not, the message is passed over. This can mean that when the missing messages arrive subsequently, they are not noticed by a browse-next operation.


```
/* Another way, which works for both grouped and ungrouped messages, */  
/* would be to remember the MsgId of the first message when it was */
```




You might run multiple copies of a dispatcher application to browse messages on a

If get operations are inhibited for a queue from which you are attempting to get messages (or any queue to which the queue name resolves), the MQGET call fails and returns the MQRC_GET_INHIBITED reason code. This happens even if you are using the MQGET call for browsing. You might be able to get a message successfully if you attempt the MQGET call at a later time, if the design of the application is such that other programs change the attributes of queues regularly.

If a dynamic queue (either temporary or permanent) has been deleted, MQGET calls using a previously-acquired object handle fail and return the MQRC_Q_DELETED reason code.

Note:

In practice, publisher applications cover a spectrum from solely using fixed topics, as in this example, and variable topics, as in the next. “Example 2: Publisher to a variable topic” also demonstrates combining the use of topics and topic strings.

Related concepts

“Example 2: Publisher to a variable topic”

A Websphere MQ program to illustrate publishing to a programmatically defined topic.

See the output in Figure 20 on page 172.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mqc.h>
int main(int argc, char **argv)
{
    char    topicNameDefault[] = "STOCKS";
    char    topicStringDefault[] = "IBM/PRICE";
    char    publicationDefault[] = "130";
    MQCHAR48 qmName = "";
```

There are a few points to note about this example.

```
char topicNameDefault[] = "STOCKS";
```

The default topic name STOCKS defines part of the topic string. You can override this topic name by providing it as the first argument to the program, or eliminate the use of the topic name by supplying / as the first

Related concepts

You can also use a *durable* subscription with this pattern. Typically if a managed durable subscription is used it is done for reliability reasons, rather than to establish a subscription that, without any errors occurring, would outlive the

topic object named in `sd.Objectname`, thought they usually turn out to be one and the same. See the discussion in “Example 2: Publisher to a variable topic” on page 170.

By making the subscription durable in the example, publications continue to be sent to the subscription queue after the subscriber has closed the subscription with the `MQCO_KEEP_SUB` option. The queue continues to receive publications when the subscriber is not active. You can override this behavior by creating the subscription with the `MQSO_PUBLICATIONS_ON_REQUEST` option and using `MQSUBRQ` to request the retained publication.

The subscription can be resumed later by opening the subscription with the `MQCO_RESUME` option.

You can use the queue handle, `Hobj`, returned by `MQSUB` in a number of ways. The queue handle is used in the example to inquire on the name of the subscription queue. Managed queues are opened using the default model queues `SYSTEM NDURABLE.MODEL.QUEUE` or `SYSTEM DURABLE.MODEL.QUEUE`. You can override the defaults by providing your own durable and non-durable model queues on a topic by topic basis as properties of the topic object associated with the subscription.

Regardless of the attributes inherited from the model queues, you cannot reuse a managed queue handle to create an additional subscription. Nor can you obtain another handle for the managed queue by opening the managed queue a second.9(properties)pe os

MQCHAR48 qName;

Although the example doesn't require knowledge of the subscription queue, we do inquire the name of the subscription queue - the MQINQ binding is a little awkward in the C language, so you may find this part of the example useful to study.

```
do {
    MQCONN(qmName, &Hconn, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
    sd.ObjectString.VSPtr = topicString;
    sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
    sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF QUIESCING ;
    sd.ResObjectString.VSPtr = resTopicStr;
    sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
    MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
    gmo.WaitInterval = 10000;
    inquireQname(Hconn, Hobj, qName);
    printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\\\"\\n",
        gmo.WaitInterval/1000, resTopicStr, qName);
    do {
        memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
        memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
        md.Encoding = MQENC_NATIVE;
        md.CodedCharSetId = MQCCSI_Q_MGR;
        memset(publicationBuffer, 0, sizeof(publicationBuffer));
        MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1,
            publication, &messlen, &CompCode, &Reason);
        if (Reason == MQRC_NONE)
            printf("Received publication \"%s\\\"\\n", publication);
    }
    while (CompCode == MQCC_OK);
    if (CompCode != MQCC_OK && Reason != MQRC_NONE && Reason != MQRC_AVESC) break;
```


Related concepts

might be part of a publish/subscribe cluster and the consumer attached to a queue manager outside the queue manager cluster. The consumer receives publications through standard distributed queuing by defining the subscription queue as a remote queue definition.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
```


Note: Setting MQSO_ALTER or MQSO_RESUME without setting MQSO_DURABLE is an error, and sd.SubName must be set and refer to a subscription that can be resumed or altered.

```
*subscriptionQueue = '\0';  
sdOptions = sdOptions + MQSO_MANAGED;
```

There are some additional comments to make on the code in this example,
if (strlen(subscriptionQueue))

conversion on the MQGET, you must specify CONVERT(YES) on the sender channel that sends the data to its final destination. This ensures that WebSphere MQ converts the data during transmission. In this case, your data-conversion exit

MQFMT_DEAD_LETTER_HEADER format. The exit is invoked to convert only the user-defined format; the queue manager converts any built-in formats that precede the user-defined format.

A user-written exit can also be invoked to convert a built-in format, but this happens only if the built-in conversion routines cannot convert the built-in format successfully.

- Do not update any resources controlled by a resource manager other than

Table 13. Skeleton source files (continued)

Platform	File

- MQFLOAT32
- MQFLOAT64
- MQSHORT
- MQLONG
- MQINT8
- MQUINT8
- MQINT16
- MQUINT16
- MQINT32
- MQUINT32
- MQINT64
- MQUINT64

MQCHAR fields are code page converted, but MQBYTE, MQINT8 and MQUINT8 are left untouched. If the encoding is different, MQSHORT, MQLONG, MQINT16, MQUINT16, MQINT32, MQUINT32, MQINT64, MQUINT64, MQFLOAT32, MQFLOAT64 and MQBOOL are converted

Supported only on WebSphere MQ for z/OS.

TEST EQU *
SERIAL_NUMBER DS F
ID DS CL5
VERSION DS H
CODID-5500(DS)-1000XL4H

F

DS

SERIAL_NUMBER31), F
D5), F
VERSION15), F
CODID-5500CHAR(4), F /F
3)R31), F
24);y

The functions generated by the `crtmxcvx` command use macros that assume



You need to take into consideration if you are building 32-bit or 64-bit applications and whether or not you are in a threaded or non threaded environment.

Non-threaded environment:

The loadable object must have its name in upper case, for example MYFORMAT. Use the libmqm library to resolve the calls to MQXCNV.

Threaded environment:

In addition to creat-333(thr)15eversdition for environm,ons.

```
xl c_r -e MQStart -bE: MFORMAT.exp -bM SRE -o /var/mqm/exits/MFORMAT_r \
MFORMAT.c -I/usr/mqm/inc -L/usr/mqm/lib -lmqzfr
```


Threaded:

Compile the exit source code by issuing the following command:

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MFORMAT_r MFORMAT.c  
-I/opt/mqm/inc -L/opt/mqm/lib -Wl,-rpath=/opt/mqm/lib  
-Wl,-rpath=/usr/lib -lmqzsf_r
```

32 bit applications:

Non-threaded:

Compile the exit source code by issuing the following command:

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MFORMAT MFORMAT.c  
-I/opt/mqm/inc -L/opt/mqm/lib -Wl,-rpath=/opt/mqm/lib  
-Wl,-rpath=/usr/lib -lmqzsf
```

Threaded:

Compile the API exit source code by issuing the following command:

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MFORMAT_r MFORMAT.c  
-I/opt/mqm/inc -L/opt/mqm/lib -Wl,-rpath=/opt/mqm/lib  
-Wl,-rpath=/usr/lib -lmqzsf_r
```

64 bit applications:

Non-threaded:

Compile the exit source code by issuing the following command:

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MFORMAT MFORMAT.c  
-I/opt/mqm/inc -L/opt/mqm/lib64 -Wl,-rpath=/opt/mqm/lib64  
-Wl,-rpath=/usr/lib64 -lmqzsf
```

Threaded:

Compile the exit source code by issuing the following command:

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MFORMAT_r MFORMAT.c  
-I/opt/mqm/inc -L/opt/mqm/lib64 -Wl,-rpath=/opt/mqm/lib64  
-Wl,-rpath=/usr/lib64 -lmqzsf_r
```

On Solaris:

SPARC platform:

32 bit applications:

Compile the exit source code by issuing the following command:

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MFORMAT \  
MFORMAT.c -I/opt/mqm/inc -L/opt/mqm/lib -R/opt/mqm/lib \  
-R/usr/lib/32 -lmqm -lmqmcs -lmqmzse -lsocket -lnsl -ldl
```

64 bit applications:

Compile the API exit source code by issuing the following command:

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MFORMAT \  
MFORMAT.c -I/opt/mqm/inc -L/opt/mqm/lib64 -R/opt/mqm/lib64 \  
-R/usr/lib/64 -lmqm -lmqmcs -lmqmzse -lsocket -lnsl -ldl
```

x86-64 platform:

32 bit applications:

Compile the API exit source code by issuing the following command:

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MFORMAT \
```

Here, insert the code fragment generated in step 3 on page 209.

- b. Near the middle of the source file, a comment box starts with:

```
/* Insert calls to the code fragments to convert the format's */
```

This is followed by a commented-out call to the function `ConverttagSTRUCT`.

Change the name of the function to the name of the function that you added in step 5a on page 209 above. Remove the comment characters to activate the function. If there are several functions, create the following to

2. Data-conversion exit programs must be reentrant.
3. MQXCNVC is the *only* MQI call that can be issued from a data-conversion exit.



You can inquire about the current values of most attributes using the MQINQ call. The MQI also provides an MQSET call with which you can change some queue attributes. You cannot use the MQI calls to change the attributes of any other type of object; instead you must use:

For WebSphere MQ for z/OS

of work. When a program performs a backout, WebSphere MQ restores the

Two-phase commit is supported under:

- WebSphere MQ for AIX
- WebSphere MQ for i5/OS
- WebSphere MQ for HP-UX
- WebSphere MQ for Linux
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- CICS for MVS/ESA 4.1
- CICS Transaction Server for z/OS
- TXSeries
- IMS/ESA[®]
- z/OS batch with RRS
- Other external coordinators using the X/Open XA interface

Single-phase commit is supported under:

- WebSphere MQ for i5/OS
- WebSphere MQ on UNIX systems
- WebSphere MQ for Windows
- z/OS batch


Note: For further details on external interfaces see “Interfaces to external syncpoint

message that has been repeatedly backed out. This might indicate a problem with this message, which you should investigate. See `BackoutCount` for details of *BackoutCount*.

Except on z/OS batch with RRS, if a program issues the MQDISC call while there are uncommitted requests, an implicit syncpoint occurs. If the program ends abnormally, an implicit backout occurs. On z/OS, an implicit syncpoint occurs if the program ends normally without first calling MQDISC.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify WebSphere MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

 For batch applications, you can use the WebSphere MQ syncpoint management calls: MQCMIT and MQBACK. For backward compatibility, CSQBCMT and CSQBBAK are available as synonyms.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as WebSphere MQ and DB2, within a single unit of work you can use RRS. For further information see “Transaction management and recoverable resource manager services” on page 219.

Committing changes using the MQCMIT call:

There is a description of the MQBACK call in the *WebSphere MQ Application Programming Reference*.

commit the unit of work. If, however, the application terminates without disconnecting, the unit of work is rolled back as the application is deemed to have terminated abnormally.



Use global units of work when you also need to include updates to resources belonging to other resource managers. Here the coordination can be internal or external to the queue manager:

Internal syncpoint coordination:

Queue manager coordination of global units of work is no33(coal)-33u.coaldisconnectW333(8(eb

In these cases, the unit of work must include updates to only those resource managers that were available when the unit of work was started.

If one of the resource managers cannot commit its updates, all the resource managers are instructed to roll back their updates, and MQCMIT completes with a warning. In unusual circumstances (typically, operator intervention), an MQCMIT call might fail if some resource managers commit their updates but others roll them back; the work is deemed to have completed with a *mixed* outcome. Such

For more information about setting up transactional support, see the *WebSphere MQ System Administration Guide*, and also TXSeries CICS documentation, for example, *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

 WebSphere MQ on UNIX systems, WebSphere MQ for i5/OS, and WebSphere MQ

TPM allows you to specify to WebSphere MQ the transaction manager name, for

If you want to use WebSphere MQ for i5/OS with native i5/OS commitment control as an external syncpoint coordinator, ensure that any job with commitment

the commit contains message CPF835F indicating that a commit or rollback operation failed. The messages preceding this indicate the cause of the problem,

If triggering is enabled for a queue, you can create a *process-definition object* associated with it. This object can contain one or more other

object associated with the application queue. Trigger messages have a fixed format (see “Format of trigger messages” on page 245).

Initiation queue

An *initiation queue* is a local queue on which the queue manager puts trigger messages. A queue manager can own more than one initiation queue, and each one is associated with one or more application queues. A shared queue, a local queue accessible by queue managers in a queue-sharing group, can be an initiation queue on WebSphere MQ for z/OS.

Trigger monitor

A *trigger monitor* is a continuously-running program that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. The trigger monitor uses

In Figure 37, the sequence of events is:

- 1.

3. When using triggering for channels, use trigger type FIRST or DEPTH.

So far, the relationship between the queues within triggering has been only on a one to one basis. Consider Figure 38.

An application queue has a process definition object associated with it that holds details of the application that will process the message. The queue manager places the information in the trigger message, so only one initiation queue is necessary. The trigger monitor extracts this information from the trigger message and starts the relevant application to deal with the message on each application queue.

When the queue manager creates a trigger message, it copies information from the attributes of the process definition object into the trigger message.

3. The number of messages on the queue with priority greater than or equal to `mqttPriority` was previously, depending on `mqttTriggerType`:
- Zero (for trigger type MQTT_FIRST)
 - Any number (for trigger type MQTT EVERY)
 - `mqttDepth` minus 1 (for trigger type MQTT_DEPTH)

Note:

- a. For non-shared local queues, the queue manager counts both committed and uncommitted messages when it assesses whether the conditions for a trigger event exist. Consequently an application might be started when there are no messages for it to retrieve because the messages on the queue have not been committed. In this situation, consider using the wait option with a suitable `mqttWaitTime`, so that the application waits for its messages

- init_queue : blanks
- init_queue_len : blanks
- init_queue_size : blanks

7. An initiation queue has been created, and has been specified in the

$\text{init_queue_size} = e$

This is to allow for a queue server that issues an MQGET call, finds the queue empty, and so ends; however, in the interval between the MQGET and the MQCLOSE calls, one or more messages arrive.

Note:

- a. If the program serving the application queue does not retrieve all the messages, this can cause a closed loop. Each time that the program closes the queue, the queue manager creates another trigger message that causes the trigger monitor to start the server program again.
- b. If the program serving the application queue backs out its get request (or if the program abends) before it closes the queue, the same happens. However, if the program closes the queue before backing out the get

This allows applications to be triggered only when they can retrieve messages from the application queue.

- d. A trigger-monitor application issues an MQOPEN call for input from an

on this reply-to queue so that each time a reply arrives, the reply could trigger an instance of the server to process the reply.



Consider an organization with a number of branch offices that each transmit details of the day's business to the head office. They all do this at the same time, at

the trigger interval is very small, another trigger message is generated each time that a message is put onto the application queue, even though the trigger type is FIRST, not EVERY. (Trigger type FIRST with a trigger interval of zero is equivalent to trigger type EVERY.)

• •

•

••

must always be prepared to handle this situation. It is recommended that you use the wait option with the MQGET call, setting the `wait` to a suitable value.

2. For local shared queues (that is, shared queues in a queue-sharing group) the queue manager counts committed messages only.



When you design applications that use triggering, be aware that there might be a delay between a trigger monitor starting a program and other messages becoming available on the application queue. This can happen when the message that causes coming

It is supplied as a CICS program; define it with a 4-character transaction name. Enter the 4-character name to start the trigger monitor. It uses the default queue manager (as named in the qm.ini file or, on WebSphere MQ for Windows, the registry), and the SYSTEM.CICS.INITIATION.QUEUE.

If you want to use a different queue manager or queue, build the trigger monitor the MQTMC2 structure: this requires you to write a program using the EXEC CICS START call, because the structure is too long to add as a parameter. Then, pass the MQTMC2 structure as data to the START request for the trigger monitor.

When you use the MQTMC2 structure, you need to supply only the `MQTMC2`, `MQTMC2`, `MQTMC2`, and `MQTMC2` parameters to the trigger monitor as it does not reference any other fields.

Messages are read from the initiation queue and used to start CICS transactions, using EXEC CICS START, assuming the APPL_TYPE in the trigger message is MQAT_CICS. The reading of messages from the initiation queue is performed under CICS syncpoint control.

Messages are generated when the monitor has started and stopped as well as when an error occurs. These messages are seJ0-2.4TD[(Mhe)-333(-333c0Tc0T.contr)4uansactioo2rol

`pdobj.trigger_data` A character field containing user data for use by the trigger monitor. When the queue manager creates a trigger message, it fills this field using the `pdobj.trigger_data` attribute of the process definition object identified in `pdobj.trigger_data`.

that queue manager. It is more efficient to do this above the level of individual applications, and thus without having to change the code of each application affected.

Here are a few suggestions of areas in which API exits might be useful:

- For *security*, you can provide authentication, checking that applications are

For more information about using the sample exit that we supply, see “The API exit sample program” on page 485. For reference information on the API exit calls, external control blocks, and associated topics, see “Reference information” on page 254.

Using message handles in API exits:

You can control which message properties an API exit has access to. Properties are associated with an ExitMsgHandle. Properties set in a put exit are set on the message being put, but properties retrieved in a get exit are not returned to the application.

When you register an MQ_INIT_EXIT exit function using the MQXEP MQI call with **Function** set to MQXF_INIT and **ExitReason** set to MQXR_CONNECTION, you pass in an MQXEPO structure as the **ExitOpts** parameter. The MQXEPO structure contains the following fields:

When a new queue manager is created, the ApiExitTemplate definitions in mqs.ini are copied to the ApiExitLocal definitions in qm.ini for the new queue manager. When a queue manager is started, both the ApiExitCommon and ApiExitLocal definitions are used. The ApiExitLocal definitions replace the ApiExitCommon definitions if both identify a routine of the same name. The Sequence attribute, described in the *WebSphere MQ System Administration Guide* determines the order in which the routines defined in the stanzas run.



PA-RISC platform:

32 bit applications:

Non-threaded:

1. Compile the ApiExit source code


```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \  
-I/opt/mqm/inc -L/opt/mqm/lib -Wl,-rpath=/opt/mqm/lib \  
-Wl,-rpath=/usr/lib -lmqzfr
```

64 bit.503Tmit.it.503Tmit.it4ications:.9780mqm9.9780-2.3m/inceded:.97807m9.9780-2.4m/inAPI3Tm

5. “The exit entry point registration call (MQXEP)” on page 266
6. “Invoking exit functions” on page 269



Here we describe the structure of the external control blocks, MQAXP and MQAXC.

WebSpU2iRe-333(MQA-333(MAPI-333(exit)-333(fparamern-333(str)ctur)e-333((MQXXP)):TJ/F71

MQXR_CONNECTION

The exit is being invoked to initialize itself before an MQCONN or MQCONNX call, or to end itself after an MQDISC call.

MQXR_BEFORE

The exit is being invoked before executing an API call, or before converting data on an MQGET.

MQXR_AFTER

The exit is being invoked after executing an API call.

ExitResponse (MQLONG) - output

This value can be set only by an MQXR_BEFORE exit function. It

Bypass exit functions later in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain for this API call invocation. Invoke exit functions in the MQXR_AFTER chain that match exit functions earlier in the MQXR_BEFORE chain.

MQXR2_CONTINUE_CHAIN

Continue with the next exit in the chain.

For information on the interaction between ExitResponse and ExitResponse2, and its affect on exit processing, see “How queue managers process exit functions” on page 259.

Feedback (MQLONG) - input/output

Communicate feedback codes between exit function invocations. This is initialized to:

MQFB_NONE (0)

before invoking the first function of the first exit in a chain.

Exits can set this field to any value, including any valid MQFB_* or MQRC_* value. Exits can also set this field to a user-defined feedback value in the range MQFB_APPL_FIRST to MQFB_APPL_LAST.

APICallerType (MQLONG) - input

The API caller type, indicating whether the WebSphere MQ API caller is external or internal to the queue manager: MQXACT_EXTERNAL or MQXACT_INTERNAL.

ExitUserArea (MQBYTE16) - input/output

A user area, available to all the exits associated with a particular

ExitInfoObj(Ex33317.9(e)-333(M)0the)-333(t/outpu)-333(to:)]TJ/F1ACT_EXTERUA0((0)YTE16

ExitPDArea (MQBYTE48) - input/output

A problem determination area, initialized to MQXPDA_NONE (binary zeros for the length of the field) for each invocation of an exit function.

For an MQXR_AFTER exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored (the remaining exit functions in the MQXR_AFTER chain are always invoked)
- MQXCC_SUPPRESS_FUNCTION and MQXCC_SKIP_FUNCTION are not valid

For an MQXR_CONNECTION exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored
- MQXCC_SUPPRESS_FUNCTION, MQXCC_SKIP_FUNCTION, MQXCC_SUPPRESS_EXIT are not valid

In all cases, where an exit or the queue manager sets CompCode and Reason, the values set can be changed by an exit invoked later, or by the API call (if the API call is invoked later).

Table 14. MQXR_BEFORE exit processing

Value of ExitResponse	CompCode and Reason set by	Value of ExitResponse2 (default continuation) Chain	Value of ExitResponse2 (default continuation) API
MQXCC_OK	exit	Y	Y
MQXCC_SUPPRESS_EXIT	exit	Y	Y
MQXCC_SUPPRESS_FUNCTION	queue manager	N	N
MQXCC_SKIP FUNCTION	exit	N	N
MQXCC_FAILED	queue manager	N	N

WebSphere MQ API exit context structure (MQAXC):

The MQAXC structure is used as an input parameter to an API exit.

MQAXC has the following C declaration:

```
typedef struct tagMQAXC {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;           /* Structure version number */
    MQLONG    Environment;       /* Environment */
    MQCHAR12   UserId;           /* UserId associated with appl */
    MQBYTE40   SecurityId        /* Extension to UserId running appl */
    MQCHAR264  ConnectionName;   /* Connection name */
    MQLONG     LongMCAUserIdLength; /* long MCA user identifier length */
    MQLONG     LongRemoteUserIdLength; /* long remote user identifier length */
    MQPTR      LongMCAUserIdPtr;  /* long MCA user identifier address */
    MQPTR      LongRemoteUserIdPtr; /* long remote user identifier address */
}
```


programs, the constant MQAXC_STRUC_ID_ARRAY is also defined, with the same value as MQAXC_STRUC_ID, but as an array of characters instead of a string.

The exit handler sets this field on entry to each exit function.

Version (MQLONG) - input

The structure version number, with a value of:

MQAXC_VERSION_1

Version number for the exit context structure.

MQAXC_CURRENT_VERSION

Current version number for the exit context structure.

The exit handler sets this field on entry to each exit function.

Environment (MQLONG) - input

The environment from which a WebSphere MQ API call was issued that


```
typedef struct tagMQACH {  
    MQCHAR4   StrucId;           /* Structure identifier */  
    MQLONG    Version;          /* Structure version number */  
    MQLONG    StrucLength;      /* Length of the MQACH structure */  
    MQLONG    ChainAreaLength;  /* Exit chain area length */  
}
```

Exit functions must release the storage for any exit chain areas that they acquire,thae6Npointer(ar)1


```
typedef MQXP      MQPOINTER PMQXP;
typedef MQACH     MQPOINTER PMQACH;
typedef MQAXC     MQPOINTER PMQAXC;

typedef MQCHAR    MQCHAR16[16];
typedef MQCHAR16 MQPOINTER PMQCHAR16;

typedef MQLONG    MQPID;
typedef MQLONG    MQTID;
```

ee

01

- There is no other opportunity to drive the *after* MQ_TERM_EXIT exit function

MQRC_NONE

(0, x'000') No reason to report.

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation:

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;          /* Exit parameter structure */
MQAXC    ExitContext;        /* Exit context-3000(*)-500(Exit)-500(para7.9(eason)-33d(AILED)-33
```

Options (MQLONG)- input/output

The interface to this function is:

`MQ_CMT_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason)`

where the parameters are:

ExitParms (MQAXP) - input/output

MQCC_WARNING

Warning (partial completion)

MQCC_FAILED

Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

ControlOpts (MQCTLO) input/output

Options that control the action of MQCTL

CompCode (MQLONG) - input/output

Completion code, valid values for which are:

MQCC_OK

Successful completion.

MQCC_WARNING

Partial completion.

MQCC_FAILED

Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation - MQ_CTL_EXIT:

The queue manager logically defines the following variables:

ExitContext (MQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation:

C language invocation:

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;          /* Exit parameter structure */
MQAXC    ExitContext;        /* Exit context structure */
MQHCONN   Hconn;             /* Connection handle */
MQHOBJ    Hobj;              /* Object handle */
MQLONG    SelectorCount;     /* Count of selectors */
PMQLONG   pSelectors;        /* Ptr to array of attribute selectors */
```

Options (MQLONG) - input/output

Open options.

pHobj (PMQHOBj) - input

Pointer to object handle.

CompCode (MQLONG) - input/output

Completion code, valid values for which are:

MQCC_OK


```
MQH0BJ      Hobj;      /* Object handle */
```

CompCode (MQLONG) - input/output

Completion code, valid values for which are:

MQCC_OK

Successful completion.

MQCC_WARNING

Partial completion.

MQCC_FAILED

Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation:

The queue manager logically defines the following variables:

MQAXP	ExitParms;	/* Exit parameter structure */
MQAXC	ExitContext;	/* Exit context structure */

The interface to this function is:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
```

CompCode (MQLONG) - input/output

Completion code, valid values for which are:

MQCC_OK

Successful completion.

MQCC_WARNING

Partial completion.

MQCC_FAILED

Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language declaration:

Your exit must match the following C function prototype:

ExitContext (MQAXC) - input/output

Exit context structure.

CompCode (MQLONG) - input/output

Completion code, valid values for which are:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED, the exit function can set the reason code field to any valid MQRC_* value.

[J136.4485XIT.9500- covis,.9500-&(str)-333(-,.9500-&)eesfu(ea(eas(Reason)-64719.4104Tm000

Subscription request - MQ_SUBRQ_EXIT:

MQ_SUBRQ_EXIT provides a subscription request exit function to perform *before* and *after* subscription request processing. Use function identifier MQXF_SUBRQ with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* subscription request call exit functions.

The interface to this function is:

MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pHsub, &Action, &pSubRqOpts,
&CompCode, &Reason)

where the parameters are:

ExitParms (MQAXP) - input/output

The queue manager then logically calls the exit as follows:

What if the ExitResponse fields are incorrectly set:

If the ExitResponse field is set to a value other than one of the supported values, the following actions apply:

- For a *before* MQCONN or MQDISC API exit function:
 - The ExitResponse2 value is ignored.
 - No further *before* exit functions in the exit chain (if any) are invoked; the API

- A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.

You must write your own module to start applications in other environments.

-

- Provides z/OS-wide coordinated commitment control (using z/OS RRS) for recoverable resources accessed through z/OS RRS compliant recoverable managers for:
 - Applications that connect to WebSphere MQ using the RRS batch adapter.
 - DB2-stored procedures executing in a DB2-stored procedures address space that is managed by a workload manager (WLM) on z/OS.
- Supports the ability to switch a WebSphere MQ batch thread between TCBs.

WebSphere MQ for z/OS provides two RRS batch adapters:

CSQBRSTB

This adapter requires you to change any MQCMIT and MQBACK statements in your WebSphere MQ application to SRRCMIT and SRRBACK

- Facilities that allow the MQI calls to be executed under multiple z/OS TCBs. For more information, see the *WebSphere MQ for z/OS Concepts and Planning Guide*.

CICS adapter performance considerations:

This section describes how the CICS adapter optimizes the performance of a CICS

Two-phase commit

A two-phase commit consumes more resources than a single-phase commit, both in host processor cost and response time. This is because a two-phase commit involves one more flow to WebSphere MQ and more physical logging. If an application is restricted to recoverable updates in WebSphere MQ and no other resource managers, CICS invokes the adapter for a single-phase commit.

Syncpoint bypassing

The adapter does not use the read-only commit feature in CICS. When a transaction is restricted to non-recoverable or non-destructive work in WebSphere MQ, syncpointing is bypassed because it is not necessary. The clean-up process is performed when the task ends.

Statistics com(MQ,)-333(s.od4a)Tf-3.6-1.8TD(Statisticcpointi-phase)-3natisticcpoin

tro333(invol-333(adapeach1.7(ebSpI-phase)al7.9(e)-3330)TJT*{an3(it)-333b3(involswitch33(is)-3of{

owneicced to

The batch adapter supports queue manager connections from batch and TSO address spaces:

If we consider a Batch address space, the adapter supports connections from multiple TCBs within that address space as follows:

- Each TCB can connect to multiple queue managers using the MQCONN or MQCONNEX call (but a TCB can only have one instance of a connection to a particular queue manager at any one time).
- Multiple TCBs can connect to the same queue manager (but the queue manager handle returned on any MQCONN or MQCONNEX call is bound to the issuing TCB and cannot be used by any other TCB).

z/OS OpenEdition supports two types of pthread_create callre(r)17.9(etd33(consid0818-1.5TD(v)Tj/F

the parameters on the MQI call, suppress the call completely, or allow the call to be processed. If the call is processed, the exit is invoked again after the call has completed.

Note: The exit program is not recursive. Any MQI calls made inside the exit do not invoke the exit program for a second time.

Communicating with the exit program:

After it has been invoked, the exit program is passed a parameter list in the CICS communication area pointed to by a field called DFHEICAP.

The CICS Exec Interface Block field EIBCALEN shows the length of this area. The structure of this communication area is defined in the CMQXPA assembler-language macro that is supplied with WebSphere MQ for z/OS :

*

MQXP_COPYPLIST DSECT

the 16-byte `exit_e` field to store the address of any dynamic storage that the application obtains. This field is retained across invocations of the exit and has the same lifetime as a CICS task.

- Because the parameters in the communication area are addresses, the exit

the APPLICID attribute. Ensure that a WLM-managed queue uniquely references an associated process and that two processes do not specify the same APPLICID

- Shared queue peer recovery ensures that any incomplete updates from the primary application are completed or backed out.

3. If you want to send COMMAREA data, you must pad the program name with



For MQGETs from the reply queue, use the value of MQMD.MsgId that WebSphere MQ set in your message descriptor when you put your first message to the request queue.

MQMD.ReplyToQ

Set the value to the name of the queue where you want the bridge to send reply messages.

The CICS bridge reply messages have the same persistence as the request

[illegible]

- SEND TEXT

Optionally, and only for CICS TS2.2 and above, additional headers with formatMQH

- Outbound messages can contain several vectors, for example as a result of successive EXEC CICS SEND MAP commands being issued by a transaction.


```

brvh.brmq_vector_length = sizeof(brrcv) + strlen(CMDSTRING) ;
strncpy(brvh.brmq_vector_descriptor, RCV_VECTOR,  strlen(RCV_VECTOR)) ;
strncpy(brvh.brmq_vector_type, INBOUND,  strlen(INBOUND)) ;
strncpy(brvh.brmq_vector_version, VERSION,  strlen(VERSION)) ;
/* Initialize fields in the RECEIVE vector structure: */
strncpy(brrcv.brmq_re_transmit_send_areas, YES,  strlen(YES)) ;
strncpy(brrcv.brmq_re_buffer_indicator, NO,  strlen(NO)) ;
strncpy(brrcv.brmq_re_aid, AID,  sizeof(brrcv.brmq_re_aid)) ;
brrcv.brmq_re_data_len =strlen(CMDSTRING) ;

```

Building the message

```

/* Copy the MQCIH to the start of the message buffer */

```

structure of the ADS requested by the CICS application. The WebSphere MQ application can then build a RECEIVE MAP vector with this ADS, and send it as a new request.

As an application programmer, you can choose whether you want to interpret vector data in messages using the ADS, the ADSD, or the ADSDL (long form of the application data structure descriptor). In order to interpret the ADS directly, include the copybook from map assembly in your WebSphere MQ bridge application. If you want to do this, but you do not have access to the copybook or

Include logic in your bridge application to interpret outbound SEND MAP and RECEIVE MAP request vectors, and to build and send an inbound RECEIVE MAP vector in response to the corresponding outbound RECEIVE MAP request.

SEND MAP vectors:

Include logic in your bridge application to interpret outbound SEND MAP request vectors.

An outbound SEND MAP vector can contain an application data structure (ADS) and an application data structure descriptor in short form (ADSD) or long form (ADSDL).

To interpret a SEND MAP vector, do the following (assuming that the message contains both an ADS and an ADSD or ADSDL):

- 1.

show hexadecimal values as you would see them in an ISPF editor with *hex on*. This is equivalent to the hexadecimal value X'A1B2C3D4'.

Fields `offset` and `length` give the offset and length of the ADS, and fields `offset` and `length` give the offset and length of the ADSD or ADSDL.

Fields `offset` and `length` give the offset and length of the ADSD or ADSDL.

assumed that \mathbf{e}_i

structure. This example shows part of such a union (only the first field definition is shown for each structure, and the comments have been added following map assembly).

```
union
{
struct {
    char    dfhms1[12];    /* 12 reserved bytes    */
    int     dfhms2;        /* Offset to next field  */
    int     tranidl;        /* Data length of this field */
    int     tranidf;        /* Flag or attribute value */
}
```

```

pADSO += pADSDL_FD->adsdl_field_offset ;

/* Enter a loop where we process all fields in the ADS */
/* sequentially. It is assumed that the value of pADSDL_FD */
/* is being augmented to the next field descriptor in the */
/* ADSDL with every loop. A model for this is shown in a code */
/* fragment above. Note that adsdl_field_offset contains */

```

```
pVector = MsgBuffer + ((MQCIH *) MsgBuffer)->StrucLength ;  
:
```

For full details of all the fields, see the references already cited. Values in the diagrams shown like this:

ABCD
1234

show hexadecimal values as you would see them in an ISPF editor with *hex on*. This is equivalent to the hexadecimal value X'A1B2C3D4'.

This diagram shows the start of the ADSDL (even though the eyecatcher shows ADSL):

... ½ADSL.....±....CH0 L&\$...TRANID

adsdl_field_data_len

The data length of the named field, in this case four bytes.

adsdl_next_field

The start of the next field description.

The next diagram shows the start of the ADS, which is in long form. The values here relate directly to the sample ADSDL shown above and are for the field named as TRANID in `_. . . e . _ e`.

```
.....BAAA.....  
00000000000000002000000000000000000000000000CCCC000200000000...  
0000000000000000C00000000000000000000000000000002111000C00000000...
```

Offset to next field	Start of next field
12 bytes reserved	Value of field

For MQGETs from the reply queue, use the value of MQMD.MsgId that

2. If the WebSphere MQ application is sending a message in response to a request vector, the value in the MQCIH is ignored.
3. In the case of pseudoconversational transactions, enter the same value in the MQCIH and the first vector.

The first byte of this field is set to the value in the CICS copybook DFHAID.

MQCIH.LinkType

This section contains the following information:

- “Setting the open options and put message options for the bridge request queue” on page 350
- “Error handling by the CICS bridge” on page 350
- “Debugging CICS bridge applications” on page 352
-

In this figure:

- The client application sends a request message to run a CICS program named P1. The queue manager used by the client receives the message.

The monitor task browses the request queue awaiting the arrival of a message.
When a message arrives, it:

- Gets the request message with browse
- Checks for any problems with the request message
- Starts a CICS bridge task
- Continues browsing the request queue

The CICS bridge task

- Gets the request message from the request queue under syncpoint control
-

2. Check the message that is sent to the reply queue by the bridge monitor. If an

Some errors can cause the bridge monitor transaction, CKBR, to terminate unexpectedly. If you are using triggered queues to start the monitor, and there are still messages on the bridge request queue, the CKTI transaction might attempt to restart CKBR. If the original error persists, this can lead to a loop of CKBR failures. To halt the loop, set off the `error_clear` attribute of the request queue while you diagnose and fix the underlying problem.

data structure descriptor, long form, is ADSDL. However, if you look at the eye-catcher in the ADSDL you will see that it is ADSL, which leads to the ambiguity. The correct use of ADSL is to describe the application data structure, not its descriptor, in long form.

This chapter helps you to write IMS applications using WebSphere MQ.

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using WebSphere MQ.”
- To write applications that exploit the WebSphere MQ-IMS Bridge, see “Writing WebSphere MQ-IMS bridge applications” on page 359.

This section discusses the following subjects that you should consider when using WebSphere MQ in IMS applications:

- “Syncpoints in IMS applications”
- “MQI calls in IMS applications” on page 356

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint). To back out all changes since

This section covers the use of MQI calls in the following types of IMS applications:

- “Server applications”
- “Enquiry applications” on page 358

Server applications:

Here is an outline of the MQI server application model:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from WebSphere MQ queue
.
Do while Get does not fail
.
    If expected message received
        Process the message
```



```

        Do nothing
    Else if no IOPBC
        Set MessageOriented to false
        Initialize error message
        Build 'Started as batch oriented BMP' message
        Call ReportCallError to output the message
    End-if
    Else if response is not 'no message available'
        Initialize error message
        Build 'GU failed' message
        Call ReportCallError to output the message
        Set return code to error
    End-if
End-if
End-while
Else
    Initialize error message
    Build 'INIT failed' message
    Call ReportCallError to output the message
    Set return code to error
End-if

Return to calling function

```

The WebSphere MQ iTD[(Inie2ror)]T[(Set(outpead)-5e/F71Tf9.9-if)Tj33(W)91.7(ebS9)-333(M)0pk9a

Only certain types of application (for example, a simple database update or

2.

TranState
C

See “Writing data-conversion exits” on page 194 for detailed information about data conversion in general.

Sending messages to the WebSphere MQ-IMS bridge:

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIHH structure, the `MQMD` must be set to the built-in format `MQFMT_IMS`, and the `MQIHH` must be set to the name of the format that describes your message data. If there is no `MQIHH`, set the `MQMD` to your format name.

If your data (other than the LLZZs) is all character data (`MQCHAR`), use as your format name (in the `MQIHH` or `MQMD`, as appropriate) the built-in format `MQFMT_IMS_VAR_STRING`. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any `MQIHH` at the start of the message).

If your application uses `MQMD`, you are recommended to use messages with

Triggering:

The WebSphere MQ-IMS bridge does not support trigger messages.

If you define an initiation queue that uses a storage class with XCF parameters, messages put to that queue are rejected when they get to the bridge.

object's properties (the equivalent of issuing an MQINQ or MQSET call), and by

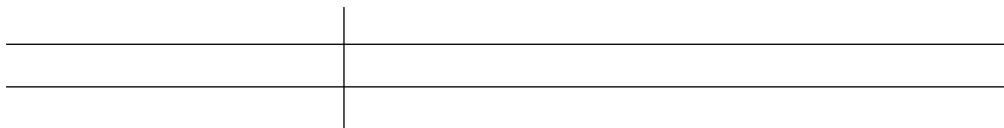
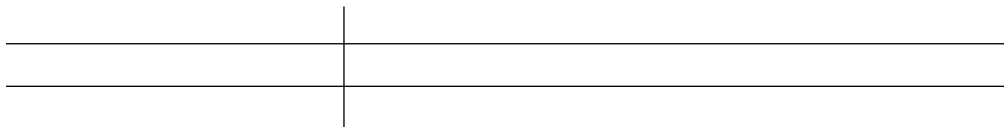
MQDistributionList

(Not WebSphere MQ for z/OS.) An object of the MQDistributionList class represents a distribution list (used to send multiple messages with a single MQPUT). It contains a list of MQDistributionListItem objects.

MQDistributionListItem

(Not WebSphere MQ for z/OS.) An object of the MQDistributionListItem class represents a single distribution list destination. It encapsulates the MQOR, MQRR, and MQPMR structures, and has properties corresponding to the fields of these structures.

In a WebSphere MQ f.



64 bit COBOL copy books are installed in the following directory:

`/usr/mqm/inc/cobcpy64`

3. In the following examples set COBCPY to:

```
$ cob64 -xvP amqspu.t.cbl -L /usr/mqm/lib64 -lmqmc.b Server for COBOL
$ cob64 -xvP amqspu.t.cbl -L /usr/mqm/lib64 -lmqic.b Client for COBOL
$ cob64 -xtvP amqspu.t.cbl -L /usr/mqm/lib64 -lmqmc.b_r Threaded Server for COBOL
```



```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvcexit.o srvcexit.c -I/opt/mqm/inc
ld +b: -b srvcexit.o +ee MQStart -o /var/mqm/exits/srvcexit_32_r -L/opt/mqm/lib \
-L/usr/lib -lmqic_r -lpthread
```

The following example builds a server exit srvcexit in a non-threaded 64-bit environment:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvcexit.o srvcexit.c -I/opt/mqm/inc
ld -b +noenvvar srvcexit.o +ee MQStart -o /var/mqm/exits64/srvcexit_64_r \
-L/opt/mqm/lib64 -L/usr/lib/pa20_64 -lmqic
```

The following example builds a server exit srvcexit in a threaded 64-bit environment:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvcexit.o srvcexit.c -I/opt/mqm/inc
ld -b +noenvvar srvcexit.o +ee MQStart -o /var/mqm/exits64/srvcexit_64_r \
-L/opt/mqm/lib64 -L/usr/lib/pa20_64 -lmqic_r -lpthread
```

4 ()

Build examples of amqsput0, cliexit and srvcexit on IA64(IPF) platform.

The following example builds the sample program amqsput0 as a client application in a non-threaded 32-bit environment:

```
c89 -Wl, +b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -I/opt/mqm/inc
-L/opt/mqm/lib -L/usr/lib/hpux32 -lmqic
```

The following example builds the sample program amqsput0 as a client application in a threaded 32-bit environment:

```
c89 -mt -Wl, +b,: +e -D_HPUX_SOURCE -o amqsputc_32_r f 8.9802 0 0 8.9802 161.6428 415.8312 Tm [(c89)-
```

```
c89 -mt +DD64 +e -Wl,+noenvvar -D_HPUX_SOURCE -o amqspu64_r amqspu0.c -I/opt/mqm/inc
-L/opt/mqm/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

The following example builds a client exit cliexit in a non-threaded 32-bit environment:

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I/opt/mqm/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -L/opt/mqm/lib \
-L/usr/lib/hpux32 -lmqic
```

The following example builds a client exit cliexit in a threaded 32-bit environment:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I/opt/mqm/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -L/opt/mqm/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

The following example builds a client exit cliexit in a non-threaded 64-bit environment:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I/opt/mqm/inc
ld -b +noenvvar cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-L/opt/mqm/lib64 -L/usr/lib/hpux64 -lmqic
```

The following example builds a client exit cliexit in a threaded 64-bit environment:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I/opt/mqm/inc
ld -b +noenvvar cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64_r \
-L/opt/mqm/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

The following example builds a server exit srvexit in a non-threaded 32-bit environment:

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I/opt/mqm/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -L/opt/mqm/lib \
-L/usr/lib/hpux32 -lmqm
```

The following example builds a server exit srvexit in a threaded 32-bit environment:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I/opt/mqm/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -L/opt/mqm/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

The following example builds a server exit srvexit in a non-threaded 64-bit environment:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I/opt/mqm/inc
ld -b +noenvvar srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-L/opt/mqm/lib64 -L/usr/lib/hpux64 -lmqm
```

The following example builds a server exit srvexit in a threaded 64-bit environment:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I/opt/mqm/inc
ld -b +noenvvar srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64_r \
-L/opt/mqm/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
environment:
```

The following table shows which library to use in different environments

[illegible]


```
$ cob32 -xv amqsput.cbl -L /opt/mqm/lib -lmqmb Server for COBOL
$ cob32 -xv amqsput.cbl -L /opt/mqm/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L /opt/mqm/lib -lmqmb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L /opt/mqm/lib -lmqicb_r Threaded Client for COBOL
```

Compiling 64 bit programs:

```
$ cob64 -xv amqsput.cbl -L /opt/mqm/lib64 -lmqmb Server for COBOL
$ cob64 -xv amqsput.cbl -L /opt/mqm/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L /opt/mqm/lib64 -lmqmb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L /opt/mqm/lib64 -lmqicb_r Threaded Client for COBOL
```

where amqsput is a sample program

Ensure that you have specified adequate run-time stack sizes; 16 KB is the recommended minimum.

You need to link your programs with the appropriate library provided by WebSphere MQ. The following table shows which library to use in different environments

Hardware platform	Program/exit type	Library file
PA-RISC	Server for COBOL	libmqmb.sl
PA-RISC	Client for COBOL	libmqicb.sl
PA-RISC	Threaded applications	amqmb_r.sl
IA64 (IPF)	Server for COBOL	libmqmb.so
IA64 (IPF)	Client for COBOL	libmqicb.so
IA64 (IPF)	Threaded applications	amqmb_r.so



In a threaded environment, link to one of the following libraries:

```
export COBCPY=<COBCPY>  
export LIB=/opt/mqm/lib:SLIB
```


An example of the command for a nonthreaded environment is:

Table 22. Example of CRTPGM in the nonthreaded environment

- Use either the CRTBNDCBL command or the two separate commands

COBOL, C, and C++ programming languages are supported. For information about preparing your C++ programs, see [WebSphere MQ Using C++](#).

In addition to coding the MQI calls in your source code, you must add the

C++ client application, 32-bit

11/11/20

Client for COBOL

- libmqmcb_r.sl

- If you are producing an application for external coordination by an XA-compliant transaction manager such as IBM TXSeries Encina, or BEA Tuxedo, you need to link to the mqmxa.lib or mqmxa.lib library.
- If you are writing a CICS exit, link to the mqmcics4.lib library.
- You must link WebSphere MQ libraries before any other product libraries.
- The DLLs must be in the path (PATH) that you have specified.
- If you use lowercase characters whenever possible, you can move from WebSphere MQ for Windows to WebSphere MQ on UNIX systems, where use of lowercase is necessary.



Sample C source for a CICS WebSphere MQ transaction is provided by AMQSCIC0.CCS. You build it using the standard CICS facilities. For example, for TXSeries for Windows 2000:

1. 500pr[(500)Files\ -33\W91.7(ebe[(500)MQ\TooTD(17nl33clude;0:)]1f0-1111.5TD%3_-33C_FLAG


```
cobol amq0put0 LITLINK
```

3. Link the object file to the runtime system.

- Set the LIB environment variable to point to the compiler COBOL libraries.
- Link the object file for use on the WebSphere MQ server:

```
cbllink amq0put0.obj mqmcb.lib
```

- Or link the object file for use on the WebSphere MQ client:

```
cbllink amq0put0.obj mqiccb.lib
```



To compile and link a TXSeries for Windows NT®, V5.1 program using IBM VisualAge COBOL:

1. Set the environment variable (enter the following on one line):

CMQPSB.BAS

Publish/subscribe

See “Coding in Visual Basic” on page 88 for information about using the MQCONNAny call from within Visual Basic.

Call the procedure MQ_SETDEFAULTS before making any MQI calls in the project code. This procedure sing arupdefaultingsucte sise pate pr(calls)-333(in)-333() -333(siqui)-333(si-2.4TL

Table 26. Context initiators and their associated context acceptors

Context Initiator	Context Acceptor
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

The security exit program has two entry points:

- **SCY_NTLM**

The access control that WebSphere MQ provides is based on the user and group. The authentication that Windows provides is based on principals, such as user and servicePrincipalName (SPN). In the case of servicePrincipalName, there might be many of these associated with a single user.

The SSPI security exit uses the relevant Windows principals for authentication. If Windows authentication is successful, the exit passes the user ID that is associated

e

After you have written the program for your WebSphere MQ application, to create an executable application you have to compile or assemble it, then link-edit the resulting object code with the stub program that WebSphere MQ for z/OS supplies for each environment that it supports.

To run a batch or RRS program, you must include the libraries
thlqual

Note: If you are using COBOL, select the NODYNAM compiler option to enable the linkage editor to resolve references to CSQQSTUB unless you intend to use dynamic linking as described in “Dynamically calling the 4.e:

[illegible]

...

EXEC peo0 0 8.I'CSQCOPEN')0 8.IR15)0]TJ -.5 -1.1111 TD [...ECALL0 8.I15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,F

Another example of the use of a directory is for a company that has many small depots or offices, WebSphere MQ clients can be used to send messages to

```
host: LondonHost
...
host: SydneyHost
...
host: WashingtonHost
```

You must change LondonHost, SydneyHost, and WashingtonHost to the names of

description,
l,
ou,
seeAlso

The LDAP calls used in the first part of the program differ slightly depending on

from the object when we have found it. The next attribute says whether we want just the attributes or their values as well; setting this to FALSE means that we want the attribute values. The final parameter is used to return the result.

The result could contain many directory entries, each with the specified attributes and their values. We have to extract the values that we want from the result. In this sample program we only expect one entry to be found, so we only look at the first entry in the result:

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

This call returns a handle that represents the first entry, and we set up a for loop to extract all the attributes from the entry:

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);  
     attribute != NULL;  
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))  
{
```

For each of these attributes, we extract the values associated with it. Again we only expect one value per attribute, so we only use the first value; we determine which attribute we have and store the value in the appropriate program variable:

```
values = ldap_get_values(ld, ldapEntry, attribute);  
if (values != NULL && values[0] != NULL)  
{  
    if (strcmp(attribute, MQ_HOST_ATTR) == 0)  
    {  
        mqHost = strdup(values[0]);  
        ...  
    }  
}
```

Finally we tidy up by freeing memory (`ldap_value_free`, `ldap_memfree`, `ldap_msgfree`) and close the session by *unbinding* from the server:

```
ldap_unbind(ld);
```

We check that we have found all the WebSphere MQ values that we need from the directory, and if so we call `sendMessages()` to connect to the WebSphere MQ server and send the WebSphere MQ messages.

The second part of the sample program is the `sendMessages()` routine that contains all the WebSphere MQ calls. This is modelled on the `amqsput0` sample program, the differences being that the parameters to the program have been extended and `MQCONN` is used instead of the `MQCONN` call.

e 4.

e e

e e

1 1 90

- “The Asynchronous Put sample program” on page 463
- “Running the samples using remote queues” on page 464
- “Database coordination samples” on page 464
- “The CICS transaction sample” on page 471
- “TUXEDO samples” on page 471
- “Encina sample program” on page 483
- “Dead-letter queue handler sample” on page 483
- “The Connect sample program” on page 484
- “The API exit sample program” on page 485
- “Using the SSPI security exit on Windows systems” on page 486



The following tables show the techniques demonstrated by the WebSphere MQ sample programs on all systems except z/OS (see “Sample programs for WebSphere MQ for z/OS” on page 486). All the samples open and close queues using the MQOPEN and MQCLOSE calls, so these techniques are not listed separately in the tables. See the heading that includes the platform that you are interested in.



Table 29 shows the techniques demonstrated by the sample programs for WebSphere MQ on UNIX systems.

Table 29. WebSphere MQ on UNIX sample programs demonstrating use of the MQI (C and COBOL) (continued)

Technique	C (source) (1)	COBOL (source) (2)	C (executable)	Client (executable) (3)
Putting messages asynchronously and getting status using the MQSTAT call	amqsapt0.c	no sample	amqsapt	amqsaptc
Notes:				

[illegible]

To run the samples use either the C executable versions, supplied in the library



Before you can run any of the sample programs, create a queue manager and set

For the Inquire, Set, and Echo examples, the sample definitions trigger the C versions of these samples.

If you want the COBOL versions you must change the process definitions:

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

Type `amqssbx -d mysub -t mytopic -k` to invoke the subscriber using durable subscriptions that are retained after the subscriber has terminated.

Test the subscription by publishing another item using the publisher. Wait for 30 seconds for the subscriber to terminate. Publish some more items under the same topic. Restart the subscriber. The last item published whilst the subscriber was not running is displayed by the subscriber immediately it is restarted.

Run the sample from the command line using the Java command, if you have a Java environment configured. You can also run the sample from the WebSphere MQ Explorer Eclipse workspace that has a Java programming workbench already set up.

The instructions that follow show how to run the sample from the Eclipse workspace.

e

- l. In the right pane, select MQPubSubApiSample.java. The **Into folder** field should contain MQ Java Samples/src. Click **Finish**.

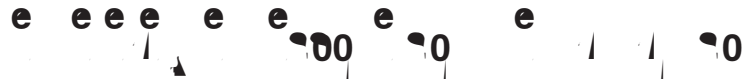
2. Run the publish/subscribe sample program. There are two ways to run the program, depending on whether you need to change the default parameters.
 - The first choice is to run the program with out making any changes.
 - In the workspace main menu, expand the src folder. Right-click **MQPubSubApiSample.javaRun-as**

To set System properties, code a default value in the accessor, for example,

[illegible]

Figure 1 is a schematic representation of the experimental design. It shows two groups of subjects: 'Control' and 'Patients'. Each group is divided into 'Pre' and 'Post' phases. The 'Control' group shows a decrease in 'Number of errors' from Pre to Post. The 'Patients' group shows an increase in 'Number of errors' from Pre to Post. The 'Number of errors' is represented by a bar chart with a y-axis from 0 to 10. The 'Control' group's bar height decreases from approximately 8 to 4. The 'Patients' group's bar height increases from approximately 4 to 8.

The program then closes the queue using the MQCLOSE call.



The Reference Message samples allow a large object to be transferred from one node to another (usually on different systems) without the need for the object to be stored on WebSphere MQ queues at either the source or the destination nodes.

A set of sample programs is provided to demonstrate how Reference Messages can be put to a queue, received by message exits, and taken from a queue. The sample

The MQSC definitions have used an AIX style for defining the exits, so if you

CRTMQMQ QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMOM/AMQSGRM')

CRTMQMQ QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)

Note: You can use either a forward slash (/) or a dash (-) when specifying parameters.

For example:

```
amqsprmq /i d:\files\infile.dat /o e:\files\outfile.dat /q QR  
/m QMGR1 /w 30 /t FLATFILE
```

Note: For UNIX platforms, you must use two backslashes (\\) instead of one to denote the destination file directory. Therefore, the above command looks like this:

```
amqsprmq -i /files/infile.dat -o e:\\files\\outfile.dat -q QR  
-m QMGR1 -w 30 -t FLATFILE
```

Running the put Reference Message program does the following:

- The Reference Message is put to queue QR on queue manager QMGR1.
- The source file and path is d:\files\infile.dat and exists on the system where the example command is issued.
- If the queue QR is a remote queue, the Reference Message is sent to another queue manager, on a different system, where a fta5a8tlp, wate333(comwind)-333(eue,)-333n.

9. The Get Reference Message sample, amqsgrm, gets messages from the queue specified in the input trigger message and checks the existence of the file.



This sample creates a Reference Message that refers to a file and puts it on a specified queue:

1. The sample connects to a local queue manager using MQCONN.
2. It then opens (MQOPEN) a model queue that is used to receive report messages.
- 3.

If you omit the `qmanagername`

Alternatively, you can use one of the sample queues defined when you ran `amqscos0.tst`, or any other queue that you have defined, for the Inquire sample.

Note: The numbers in Figure 60 on page 454 show the sequence of events.

To run the Request and Inquire samples, using triggering:

The program demonstrates how to clear the `std::cout` and

These programs are intended to run as triggered programs, so their only input is an MQTMC2 (trigger message) structure for i5/OS, Windows systems, and UNIX. This structure contains the name of a target queue whose attributes are to be inquired. The C version also uses the queue manager name. The COBOL version uses the default queue manager.

For the triggering process to work, ensure that the Inquire sample program that

For each request message removed from the request queue, the program reads the name of the queue (which we will call the *target queue*) contained in the data and opens that queue using the MQOPEN call with the MQOO_SET option. The

The function provided in the triggering sample is a subset of that provided in the trigger monitor in the **runmqtrm** program.

See “Features demonstrated in the sample programs” on page 426 for the names of these programs.

The program takes 2 parameters:

1. The name of the initiation queue (necessary)
2. The name of the queue manager (optional)

If a queue manager is not specified, it connects to the default one. A sample

1. AMQSXAS0 (in C) or AMQ0XAS0 (in COBOL), which updates a single database within a WebSphere MQ unit of work.
- 2.


```
EXEC SQL CREATE TABLE
MQBankT(Name      VARCHAR(40) NOT NULL,
        Account   INTEGER      NOT NULL,
        Balance   INTEGER      NOT NULL,
        PRIMARY KEY (Account))
END-EXEC.
```

```
EXEC SQL CREATE TABLE
MQBankTB(Name      VARCHAR(40) NOT NULL,
        Account   INTEGER      NOT NULL,
        Balance   INTEGER      NOT NULL,
        Transactions INTEGER,
        PRIMARY KEY (Account))
END-EXEC.
```

```
EXEC SQL CREATE TABLE
MQFeeTB(Account    INTEGER      NOT NULL,
        FeeDue      INTEGER      NOT NULL,
        TranFee     INTEGER      NOT NULL,
        Transactions INTEGER,
        PRIMARY KEY (Account))
END-EXEC.
```

```
db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset
```

```
db2 connect to MQFeedB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```


Assuming that you have created and configured a queue manager for the single database sample called singDBQM, with a queue called singDBQ, you increment Mr Fred Bloggs's account by 50 as follows:

```
AMQSPUT singDBQ singDBQM
```

Then key in the following message:

```
UPDATE Balance change=50 WHERE Account=1
```

You can put multiple messages on the queue.

```
AMQSXASO singDBQ singDBQM
```

The updated status of Mr Fred Bloggs's account is then printed.

Assuming that you have created and configured a queue manager for the multiple-database sample called multDBQM, with a queue called multDBQ, you decrement Ms Mary Brown's account by 75 as follows:

```
AMQSPUT multDBQ multDBQM
```

```
decrementMr Fred Bloggs/F66019T'sqCOBOL17.15.1(for)-3n's 76019T's accoM(Y)91.7(o)0(umusent)-333e9.978
```



```

$ buildserver -o MQSERV2 -f /usr/mqm/samp/amqstxsx.c \
  -f /usr/mqm/lib64/libmqm.a \
  -r MQSeries_XA_RM -s MPUT2:MPUT
  -s MGET2:MGET \
  -v -bshm
$ buildclient -o doputs -f /usr/mqm/samp/amqstxpx.c \
  -f /usr/mqm/lib64/libmqm.a
$ buildclient -o dogets -f /usr/mqm/samp/amqstxgx.c \
  -f /usr/mqm/lib64/libmqm.a

```

5. Edit ubbstxcx.cfg and add details of the machine name, working directories, and queue manager as necessary:

```
$ tnloadcf -y /usr/mqm/samp/ubbstxcx.cfg
```

6. Create the TLOGDEVICE:

```
$ tnadmin -c
```

A prompt then appears. At this prompt, enter:

```
> crdl -z /<APPDIR>/TLOG1
```

7. Start the queue manager:

```
$ strmqm
```

8. Start Tuxedo:

```
$ tnboot -y
```

You can now use the doputs and dogets programs to put messages to a queue and retrieve them from a queue.

Building the server environment for WebSphere MQ for Solaris (32-bit):

1. Create a directory (for example, <APPDIR>) in which the server environment is built and execute all commands in this directory.
- 2.

```

        -f /opt/mqm/lib/libmqm.so \
        -f /opt/mqm/lib/libmqmzse.co \
        -f /opt/mqm/lib/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
        -f /opt/mqm/lib/libmqm.so
        -f /opt/mqm/lib/libmqmzse.co \
        -f /opt/mqm/lib/libmqmcs.so

```

5. Edit ubbstxcx.cfg and add details of the machine name, working directories, and queue manager as necessary:

```
$ tnlloadcf -y ubbstxcx.cfg
```

6. Create the TLOGDEVICE:

```
$ tsmadmin -c
```

```

        -f /opt/mqm/lib64/libmqnics.so
$ buildclient -o dogets -f amqstxgx.c \
        -f /opt/mqm/lib64/libmqm.so
        -f /opt/mqm/lib64/libmqnzse.co \
        -f /opt/mqm/lib64/libmqnics.so

```

5. Edit ubbstxcx.cfg and add details of the machine name, working directories, and queue manager as necessary:

```
$ tmloadcf -y ubbstxcx.cfg
```

6. Create the TLOGDEVICE:

```
$ tnadmin -c
```

A prompt then appears. At this prompt, enter:

```
> crdl -z /<APPDIR>/TLOG1
```

7. Start the queue manager:

```
$ strmqm
```

8. Start Tuxedo:

```
$ tnboot -y
```

You can now use the doputs and dogets programs to put messages to a queue and retrieve them from a queue.

Building the server environment for WebSphere MQ for HP-UX (32-bit):

- 1.

6. Create the TLOGDEVICE:

8.

Note: Change the directory names and directory paths to match your installation.

Bulding the server environment for WebSphere MQ for Windows (64-bit):

Note: Change the fields identified by <> in the following, to the directory paths:

<MQMDIR>	the directory path specified when WebSphere MQ was installed, for example g: \Program Files\IBM\WebSphere MQ

Note: Change the directory names and directory paths to match your installation. Also change the queue manager name MYQUEUEMANAGER to the name of the

0 e

You can modify the error message file, amqsdlq.msg, to suit your own requirements. The sample puts messages to stdout, **not** to the WebSphere MQ error log file.

The WebSphere MQ System Administration Guide or the *System Management Guide* for your platform explains how the dead-letter handler works, and how you run it.

The Connect sample program allows you to explore the MQCONNEX call and its options from a client. The sample connects to the queue manager using the MQCONNEX call, inquires about the name of the queue manager using the MQINQ call, and displays it.

Note: The Connect sample program is a client sample. You can compile and run it on a server but the function is meaningful only on a client, and only client executables are supplied.

WebSphere MQ for z/OS also provides a sample API-crossing exit program, described in “The API-crossing exit for z/OS” on page 308, and sample data-conversion exits, described in “Writing data-conversion exits” on page 194.

All the sample applications are supplied in source form; several are also supplied in executable form. The source modules include pseudocode that describes the program logic.

Note: Although some of the sample applications have basic(on)-333-2.4TDel-driven61T-TD[(in)

The program is delivered in the C language. The application runs in the batch environment. See Table 40 on page 493 for the batch application.



The Credit Check sample is a suite of programs that demonstrates these techniques:

- Developing an application that runs in more than one environment
- Using a model queue to create a temporary dynamic queue
- Using a correlation identifier
- Setting and passing context information
- Using message priority and persistence
- Starting programs by using triggering
- Using reply-to queues
- Using alias queues
- Using a dead-letter queue
- Using a namelist
- Testing return codes

The application uses these MQI calls:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET for browsing and getting messages, using the wait and signal options, and for getting a specific message
- MQINQ
- MQSET
- MQCLOSE

The sample can run as a stand-alone CICS application. However, to demonstrate

The library members to use are listed in Table 38, Table 39, and Table 40 on page 493.

You must edit the run JCL supplied for the samples that you want to use (see Table 38, Table 39, and Table 40 on page 493).

The PARM statement in the supplied JCL contains a number of parameters that you need to modify. To run the C sample programs, separate the parameters by spaces; to run the assembler, COBOL, and PL/I sample programs, separate them by commas. For example, if the name of your queue manager is CSQ1 and you want to run the application with a queue named LOCALQ1, in the COBOL, PL/I, and assembler-language JCL, your PARM statement should look like this:

```
PARM=(CSQ1,LOCALQ1)
```

In the C language JCL, your PARM statement should look like this:

```
PARM=('CSQ1 LOCALQ1')
```

You are now ready to submit the jobs.



The names of the programs supplied for each of the sample batch applications, and the libraries where the source, JCL, and, where applicable, the executables reside, are listed in the following tables:

Put and Get samples	Table 38
Browse sample	Table 39
Print message sample	Table 40 on page 493

--	--	--	--

Table 42. TSO Message Handler sample (continued)

[illegible]

Table 44. CICS Queue Attributes sample (continued)

Member name	For language	Description	Source supplied in library	Executable supplied in library
CSQ4CCMS	C	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

--	--	--

default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.

3. Create a loop within the program issuing MQPUT calls until the required number of messages are put on the queue. If an MQPUT call fails, the loop is abandoned early, no further MQPUT calls are attempted, and the completion and reason codes are returned.
4. Close the queue using the MQCLOSE call with the object handle returned in step 2 on page 502. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the

Note: If you are running the sample in a CICS environment, you do not need

GET - D
SYNCPPOINT - N

=====

```
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : *****
000000001 : 000000010 : *****
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

Usage notes:

-

2	429	CSQ4BQRM
3	429	CSQ4BQRM
4	429	CSQ4BQRM



The Print message sample application uses a single program written in the C language.

1.

e

e

e

190

- Define a nickname (for example, MARY) for that user and send them a message by putting MARY in the user name field and nothing in the queue manager name field.

Create nickname

You can define an easy-to-remember name that you can use when you send a message to another user who you contact frequently. You are prompted to enter the user ID of the other user and the name of the queue manager that owns their mail queue.

Nicknames are queues that have names of the form `CSQ4SAMP.MAILMGR.userid.nickname`, where *userid* is your own user ID and *nickname* is the nickname that you want to use. With names structured in this way, users can each have their own set of nicknames.

The type of queue that the program creates depends on how you fill in the fields of the Create Nickname panel:

-

Menu program:

In the TSO environment, the menu program is invoked by the CLIST. In the CICS environment, the program is invoked by transaction MAIL.

the queue or the queue manager do not exist; these are MQRC_UNKNOWN_OBJECT_NAME and MQRC_UNKNOWN_OBJECT_Q_MGR. The program displays its own error message for each of these errors; for other errors, the program displays the completion and reason codes returned by the call.

Nickname program:

When you start the conversational-mode CICS transaction MVB1, this starts the user interface program for the application.

This program puts inquiry messages on queue CSQ4SAMP.B2.INQUIRY and gets replies to those inquiries from a reply-to queue that it specifies when it makes the

If a user requests a loan that is larger than 10000 units, the CAM initiates

Other messages

The application does not expect other messages. However, the application

- Copies the `header` of the loan request message
- Uses the `MQPMO_PASS_IDENTITY_CONTEXT` option

How the sample handles errors:

The user interface program handles errors simply by reporting them directly to the user.

The other programs do not have user interfaces, so they have to handle errors in other ways. Also, in many situations (for example, if an MQGET call fails) these other programs do not know the identity of the user of the application.

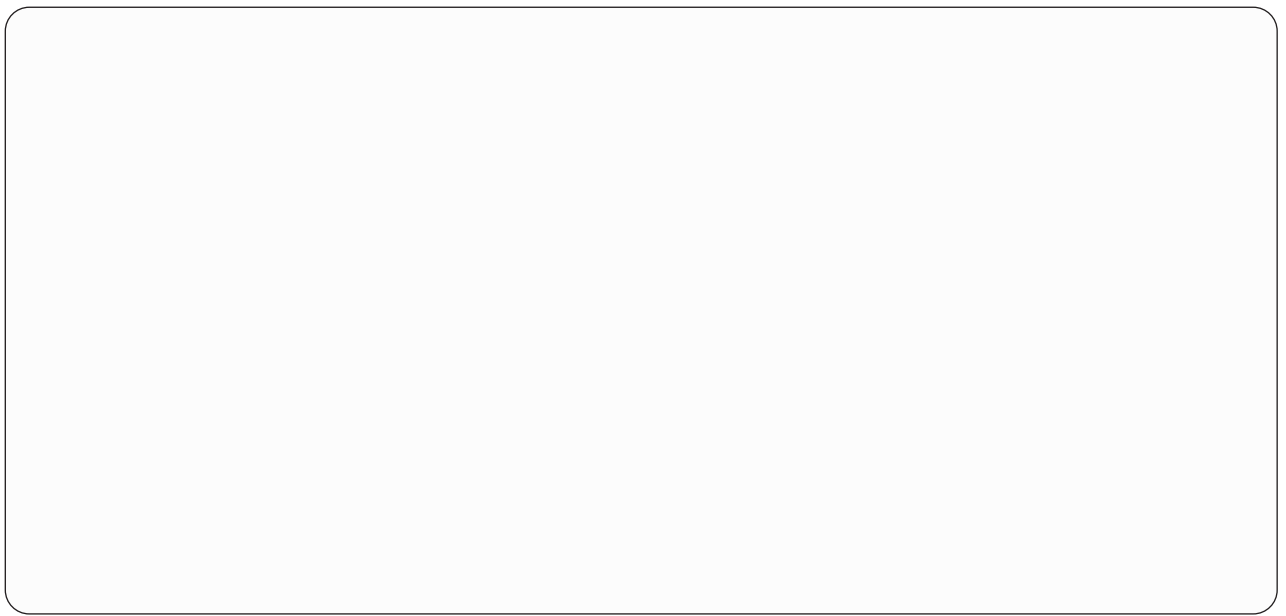
- Multiple copies of unexpected messages are never put on the sample's dead-letter queue
-



You can use the Credit Check sample application to demonstrate distributed queuing by installing the sample on two queue managers and CICS systems (with each queue manager connected to a different CICS system).

When the sample program is installed, and the trigger monitor (CKTI) is running on each system, you need to:

1. Set up the communication link between the two queue managers. For information on how to do this, see *WebSphere MQ Intercommunication*.
2. On one queue manager, create a local definition for each of the remote queues (on the other queue manager) that you want to use. These queues can be any of `CSQ4SAMP.Bn.MESSAGES`, where *n* is 3, 5, 6, or 7. (These are the queues that are served by the checking-account program and the agency-query program.) For information on how to do this, see the *WebSphere MQ Script (MQSC) Command Reference*.
3. Change the definition of the namelist (`CSQ4SAMP.B4.NAMELIST`) so that it contains the names of the remote queues that you want to use. For information on how to do this, see the *WebSphere MQ Script* (the queues (on tse



The sample design allows only messages with unique MsgId / CorrelId combinations to be selected and displayed, because the message is retrieved using the MsgId and CorrelId as the key. If the key is not unique the sample cannot retrieve the chosen message with certainty.



This section describes the design of each of the programs that comprise the Message Handler sample application.

Object validation program:

This requests a valid queue and queue manager name.

If you do not specify a queue manager name, the default queue manager is used, if

You can enter D for delete, or F for forward into the action field. If you choose to forward the message, the , , _ , , , _ e_e and _ e_e _ _ e _ e

```

/*
MQCONN(Parm1,
        &Hconn,
        &CompCode,
        &Reason);
if ((CompCode != MQCC_OK) | (Reason != MQRC_NONE))
{
    sprintf(pBuff, MESSAGE_4_E,
            ERROR_IN_MQCONN, CompCode, Reason);
    PrintLine(pBuff);
    RetCode = CSQ4_ERROR;
    goto AbnormalExit2;
}
:
}

```

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

The variables used in this code extract are those that were set in “Connecting to a queue manager” on page 539. This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the


```
MQLONG  CompCode;          /* Completion code      */
MQLONG  Reason;            /* Qualifying reason   */
MQOD     ObjDesc = { MQOD_DEFAULT };
                                /* Object descriptor   */
MQLONG  OpenOptions;       /* Options that control */
```

```
/* completion code and reason code. */
/* */
MQCLOSE(Hconn,
        &Hobj,
        MQCO_NONE,
        &CompCode,
        &Reason);
```



```

:
:
/*
/* Set the object descriptor, message descriptor and
/* put message options to the values required to
/* create the reply message.
/*
strncpy(ObjDesc.ObjectName, MsgDesc.ReplyToQ,
        MQ_Q_NAME_LENGTH);
strncpy(ObjDesc.ObjectQMgrName, MsgDesc.ReplyToQMGr,
        MQ_Q_MGR_NAME_LENGTH);
MsgDesc.MsgType = MQMT_REPLY;
MsgDesc.Report = MQRO_NONE;
memset(MsgDesc.ReplyToQ, ' ', MQ_Q_NAME_LENGTH);
memset(MsgDesc.ReplyToQMGr, ' ', MQ_Q_MGR_NAME_LENGTH);
memcpy(MsgDesc.MsgId, MQMI_NONE, sizeof(MsgDesc.MsgId));
PutMsgOpts.Options = MQPMO_SYNCPOINT +
                    MQPMO_PASS_IDENTITY_CONTEXT;
PutMsgOpts.Context = Hobj_CheckQ;
PutBuffLen = sizeof(PutBuffer);
MQPUT1(Hconn,
        &ObjDesc,
        &MsgDesc,
        &PutMsgOpts,
        PutBuffLen,
        &PutBuffer,
        &CompCode,
        &Reason);

if (CompCode != MQCC_OK)
{
    strncpy(TS_Operation, "MQPUT1",
            sizeof(TS_Operation));
    strncpy(TS_ObjName, ObjDesc.ObjectName,
            MQ_Q_NAME_LENGTH);
    Record_Call_Error();
    Forward_Msg_To_DLQ();
}
return;
}
:
:

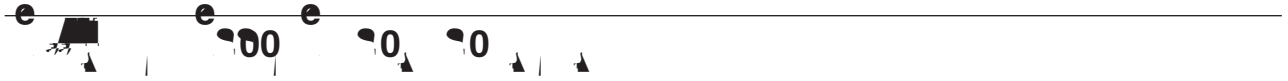
```

```
MQLONG  DataLength ;          /* Message descriptor */
MQCHAR  Buffer[BUFFERLENGTH+1]; /* Length of the message */
/* Area for message data */
MQGMD   GetMsgOpts = { MQGMD_DEFAULT };
```



```
Record_Call_Error();  
}
```

⋮




```
        ; /* Perform error processing. */
        break;
    case (MQEC_WAIT_INTERVAL_EXPIRED):
    case (MQEC_WAIT_CANCELED):
        endloop = 1;
        break;
    default:
        break;
    }
} while (endloop == 0);
}
return;
}
```



```

/* Issue the inquire call */
/* Test the output of the inquire call. If the */
/* call failed, display an error message */
/* showing the completion code and reason code, */
/* otherwise display the status of the */
/* INHIBIT-GET and INHIBIT-PUT attributes */
/* */
MQINQ(Hconn,
      *pHobj,
      SelectorCount,
      SelectorsTable,
      IntAttrCount,
      IntAttrsTable,
      CharAttrLength,
      CharAttrs,
      &CompCode,
      &Reason);
if (CompCode != MQCC_OK)
{
    sprintf(Message, MESSAGE_4_E,
            ERROR_IN_MQINQ, CompCode, Reason);
    SetMsg(Message);
}
else
{
    /* Process the changes */
} /* e*/

```

```

:
:
/*
/*      Open the queue.  If successful, do the
/*      inquire call.
/*
:
:
/*
/*      Initialize the variables for the set call:
/*      - Set SelectorTable to the attributes to be
/*      set
/*      - Set IntAttrsTable to the required status
/*      - All other variables are already set
/*
SelectorTable[0] = MQIA_INHIBIT_GET;
SelectorTable[1] = MQIA_INHIBIT_PUT;
IntAttrsTable[0] = MQQA_GET_INHIBITED;
IntAttrsTable[1] = MQQA_PUT_INHIBITED;
:
:
/*
/*      Issue the set call.
/*      Test the output of the set call.  If the
/*      call fails, display an error message
/*      showing the completion code and reason
/*      code; otherwise
/*
code; otherwise
8r500(tove8r500(INHIBITED)-500(to)-500(the)-3500(*) )TJ T* [(/*)-2500(re

```



```
int main(int argc, char **argv)
{
    /* Declare file and character for sample input */
```

```

} printf("dynamic queue name is %s\n", od.DynamicQName);
}

/*****/
/* */
/* Open the target message queue for output */
/* */
/*****/
if (argc > 3)
{
    _options = atoi( argv[3] );
    printf("open options are %d\n", _options);
}
else
{
    _options = MQOO_OUTPUT /* open queue for output */
              | MQOO_FAIL_IF_QUIESCING /* but not if MQM stopping */
              ; /* = 0x2010 = 8208 decimal */
}

MQOPEN(Hcon, /* connection handle */
      &od, /* object descriptor for queue */
      _options, /* open options */
      &Hobj, /* object handle */
      &OpenCode, /* MQOPEN completion code */
      &Reason); /* reason code */

/* report reason, if any; stop if failed */
if (Reason != MQRC_NONE)
{
    printf("MQOPEN ended with reason code %d\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
    printf("unable to open queue for output\n");
}

/*****/
/* */
/* Read lines from the file and put them to the message queue */
/* Loop until null line or end of file, or there is a failure */
/* */
/*****/
CompCode = OpenCode; /* use MQOPEN result for initial test */
fp = stdin;

memcpy(md.Format, /* character string format */
       MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);

/*****/
/* These options specify that put operation should occur */
/* asynchronously and the application will check the success */
/* using MQSTAT at a later time.later time.later time.later time.lat 0(of)-[4ECister
pmo.Options |= MQPMO_ASYNC_RESPONSEO(with)-500(reason)al 0(specify)-500(that)-500(put)-500(opera
that there is to reut themfore MQUTr */
{hapater5eC.Lngusag@xaompls555{*1 TD |e
```




The examples in this appendix are taken from the WebSphere MQ for z/OS sample

```

*
CALL 'MQCONN' USING W02-MQM
                    W03-HCONN
                    W03-COMPCODE
                    W03-REASON.
*
*   Test the output of the connect call.  If the call
*   fails, print an error message showing the
*   completion code and reason code.
*
IF (W03-COMPCODE NOT = MQCC-OK) THEN
:
:
END-IF.
:
:

```

0 e e e

' CSQ4SAMP. B1. MODEL ' .



This example demonstrates how to use the MQOPEN call to open an existing queue.

MOD
W02-OPTIONS
W02-HOBJ
W02-COMPCODE

MOVE MQCO-NONE TO W03-OPTIONS.

```
* Set the message descriptor and put-message options to
* the values required to create the message.
```

```

*
*   API control blocks
*
01  MQM-OBJECT-DESCRIPTOR.
    COPY CMQODV.
01  MQM-MESSAGE-DESCRIPTOR.
    COPY CMQMDV.
01  MQM-PUT-MESSAGE-OPTIONS.
    COPY CMQPMOV.
*
* CMQV contains constants (for filling in the
* control blocks) and return codes (for testing
* the result of a call).
*
01  MQM-MQV.
    COPY CMQV SUPPRESS.
* -----*
PROCEDURE DIVISION.
* -----*
.
.
.
*   Get the request message.
.
.
.
* -----*
PROCESS-QUERY SECTION.
* -----*
.
.
.
*   Build the reply message.
.
.
.
*
* Set the object descriptor, message descriptor and
* put-message options to the values required to create
* the message.
* Set the length of the message.
*
MOVE MQMD-REPLYTOQ    TO MQOD-OBJECTNAME.
MOVE MQMD-REPLYTOQMGR TO MQOD-OBJECTQMGRNAME.
MOVE MQMT-REPLY       TO MQMD-MSGTYPE.
MOVE SPACES           TO MQMD-REPLYTOQ.
MOVE SPACES           TO MQMD-REPLYTOQMGR.
MOVE LOW-VALUES       TO MQMD-MSGID.
COMPUTE MQPMD-OPTIONS = MQPMD-SYNCPPOINT +
                        MQPMD-PASS-IDENTITY-CONTEXT.
MOVE W03-HOBJ-CHECKQ  TO MQPMD-CONTEXT.
MOVE LENGTH OF CSQ4BQRM-MSG TO W03-BUFFLEN.
*
      CALL 'MQPUT1' USING W03-HCONN
                          MQOD
                          MQMD
                          MQPMD
                          W03-BUFFLEN
                          W03-PUT-BUFFER
                          W03-COMPCODE
                          W03-REASON.
      IF W03-COMPCODE NOT = MQCC-OK
          MOVE 'MQPUT1'          TO M02-OPERATION
          MOVE MQOD-OBJECTNAME    TO M02-OBJECTNAME
          PERFORM RECORD-CALL-ERROR
          PERFORM FORWARD-MSG-TO-DLQ
      END-IF.
*

```


MOVE MQM-NONE TO MQM-MSGID.

```
*      API control blocks
*
01  MQM-MESSAGE-DESCRIPTOR.
    COPY CMQMDV.
01  MQM-GET-MESSAGE-OPTIONS.
    COPY CMQGMV.
*
*      CMQV contains constants (for filling in the
```



```

05          PIC  X(02).
05  L02-INQUIRY-ECB1-CC  PIC  S9(04)  BINARY.
05          PIC  X(02).
05  L02-REPLY-ECB2-CC   PIC  S9(04)  BINARY.
*
* -----*
PROCEDURE DIVISION.
* -----*
.
.
* Initialize variables, open queues, set signal on
* inquiry queue.
.
.
* -----*

```

```

ELSE
    PERFORM TEST-REPLYQ-ECB
END-IF.
*
EXTERNAL-WAIT-EXIT.
*
*   Return to performing section.
*
EXIT.
EJECT
.
.
* -----*
REPLYQ-GETSIGNAL SECTION.
* -----*
*
* This section performs an MQGET call (in syncpoint with
* signal) on the reply queue. The signal field in the
* MQMD is set to the address of the ECB.
* Response handling is done by the performing section.
* -----*
*
    COMPUTE MQMD-OPTIONS          = MQMD-SYNCPPOINT +
                                   MQMD-SET-SIGNAL.
    MOVE W00-WAIT-INTERVAL        TO MQMD-WAITINTERVAL.
    MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
    MOVE ZEROS                    TO L02-REPLY-ECB2.
    SET MQMD-SIGNAL1 TO ADDRESS OF L02-REPLY-ECB2.
*
* Set msgid and correlid to nulls so that any message
* will qualify.
*
    MOVE MQMI-NONE TO MQMD-MSGID.
    MOVE MQCI-NONE TO MQMD-CORRELID.
*
    CALL 'MQGET' USING W03-HCONN
                      W03-HOBJ-REPLYQ
                      MQMD
                      MQGMD
                      W03-BUFFLEN
                      W03-GET-BUFFER
                      W03-DATALEN
                      W03-COMPCODE
                      W03-REASON.
*
REPLYQ-GETSIGNAL-EXIT.
*
*   Return to performing section.
*
EXIT.
EJECT
.
.

```

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CVC1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see "Sample programs (all platforms except z/OS)" on page 425.

.
.
.
*
*
*
*

-----*
WORKING-STORAGE SECTION.
-----*


```

*
* Initialize the variables required for the set call:
* - Set W02-SELECTORS-TABLE to the attributes to be set
* - Set W02-INTATTRS-TABLE to the required status
* - All other variables are already set
*
      MOVE MQIA-INHIBIT-GET    TO W02-SELECTORS(1).
      MOVE MQIA-INHIBIT-PUT    TO W02-SELECTORS(2).
      MOVE MQQA-GET-INHIBITED TO W02-INTATTRS(1).
      MOVE MQQA-PUT-INHIBITED TO W02-INTATTRS(2).
*
* Set the attributes.
*
      CALL 'MQSET' USING W02-HCONN, W02-INTA5-1.FBJ, *W02-SELECTOCOUNT, *W02-SELECTORS-TABL, *W02-INTATTRS-TABL,
*
* eset the the call:
*
* - the ompletion odhe* -*
*
      WHCONN, NOTHCONN, ==-PUT-INHCC-OK

```


e e e e e
 7 0 /39 0 00 e e e e


```
          BNE   BADCALL
:
:
BADCALL  DS     OH
:
:
*                CONSTANTS
*
          CMQA
*
*      WORKING STORAGE (RE-ENTRANT)
*
WEG3     DSECT
*
```



```

*
.
.
.
BADCALL  DS   OH
.
.
*
*
*   CONSTANTS:
*
Q_NAME   DC   CL48' REQUEST. QUEUE'  NAME OF QUEUE TO OPEN
*
CMQODA DSECT=NO, LIST=YES  CONSTANT VERSION OF MQOD

```

```

:
*
*          CONSTANTS
*
*          CMQA
*
*          WORKING STORAGE (REENTRANT)
*
WEG4      DSECT
*
CALLST    CALL , (0, 0, 0, 0, 0, 0, 0, 0, 0, 0), VL, MF=L
*
HCONN     DS    F
HOBJ      DS    F
OPTIONS   DS    F
COMPCODE  DS    F
REASON    DS    F
*
*
LEG4      EQU   *-WKEG4
END

```

This example demonstrates how to use the MQPUT call to put a message on a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

:


```

PUT      DS   OH
*
      MVC   WOD_AREA, MQOD_AREA      INITIALIZE WORKING VERSION OF
*                                     MQOD WITH DEFAULTS
      MVC   WOD_OBJECTNAME, Q_NAME   SPECIFY Q NAME FOR PUT1
*
      LA    R4, MQMD                 SET UP ADDRESSES AND
      LA    R5, MQMD_LENGTH          LENGTH FOR USE BY MCL
      LA    R6, WMD                  INSTRUCTION, AS MQMD IS
      LA    R7, WMD_LENGTH           OVER 256 BYES LONG.
      MCL   R6, R4                   INITIALIZE WORKING VERSION
*                                     OF MESSAGE DESCRIPTOR
*
      MVC   WPMO_AREA, MQPMO_AREA    INITIALIZE WORKING MQPMO
*

```

```

BUFFER_LEN EQU *-BUFFER
*
WOD      CMQODA DSECT=NO, LIST=YES      WORKING VERSION OF MQOD
WMD      CMQMDA DSECT=NO, LIST=NO
WPM      CMQPMDA DSECT=NO, LIST=NO
*
CALLLST  CALL , (0, 0, 0, 0, 0, 0, 0, 0, 0, 0), VL, MF=L
*
:
:
      END

```

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```

      LA    R5,MQCC_OK
      C     R5,COMPCODE
      BNE  BADCALL

*
*
*
BADCALL DS  OH
*
*
*
      CONSTANTS

*
*
      CMQMDA DSECT=NO, LIST=YES
      CMQGMDA DSECT=NO, LIST=YES
      CMQA

*
*
      WORKING STORAGE DSECT

*
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER    DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO, LIST=NO
WGM      CMQGMDA DSECT=NO, LIST=NO

```

```
A      R5,=AL4(MQGM_ACCEPT_TRUNCATED_MSG)
```

```

COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WM      CMQMDA DSECT=NO, LIST=NO
WGM     CMQGMDA DSECT=NO, LIST=NO
*
CALLST   CALL , (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), VL, MF=L
*
:
:
:
END

```

This example demonstrates how to use the MQGET call to set a signal so that you are notified when a suitable message arrives on a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```

:
:
*
*   CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
:
*
*   OPEN A QUEUE FOR GET
*
OPEN     DS  0H
:
:
*
*   R4, R5, R6, R7 = WORK REGISTER.
*
GET      DS  0H
LA       R4, MQMD
LA       R5, MQMD_LENGTH
LA       R6, WM
LA       R7, WM_LENGTH
MVC     R6, R4
*
*                                     SET UP ADDRESSES AND
*                                     LENGTH FOR USE BY MVC
*                                     INSTRUCTION, AS MQMD IS
*                                     OVER 256 BYES LONG.
*                                     INITIALIZE WORKING VERSION
*                                     OF MESSAGE DESCRIPTOR
*
MVC     WGM_AREA, MQGMD_AREA
LA      R5, MQGMD_SET_SIGNAL
ST      R5, WGM_OPTIONS
MVC     WGM_WAITINTERVAL, FIVE_MINUTES
*                                     WAIT UP TO FIVE
*                                     MINUTES BEFORE

```


	ST	R0, SELECTORCOUNT	selectors add
*	ST	R0, INTATTRCOUNT	integer attributes



The use of PL/I is supported by z/OS only.

The examples demonstrate the following techniques:

- “Connecting to a queue manager”
- “Disconnecting from a queue manager” on page 594
-


```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS */
*****/
DCL COMPCODE          BINARY FIXED (31);
DCL REASON            BINARY FIXED (31);
DCL HCONN             BINARY FIXED (31);
DCL HOBJ              BINARY FIXED (31);
DCL OPTIONS           BINARY FIXED (31);
:
DCL MODEL_QUEUE_NAME CHAR(48) INIT(' PL1.REPLo.S1Pu1Kd11YCLDDEL_QUEUE_NAME CHAR(4TEMPQ.*IT

```



```
OPTIONS=MQCO_NONE;
```



```

/*****
/* WORKING STORAGE DECLARATIONS
/*****
DCL COMPCODE                                BINARY FIXED (31);
DCL REASON                                  BINARY FIXED (31);
DCL HCONN                                  BINARY FIXED (31);
DCL HOBJ                                  BINARY FIXED (31);
DCL BUFFLEN                                BINARY FIXED (31);
DCL DATALEN                              BINARY FIXED (31);
DCL BUFFER                                CHAR(80);

:
:
/*****
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND
/* GET MESSAGE OPTIONS
/*****
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQGMD LIKE MQGMD;

:
:
/*****
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST
/* AVAILABLE MESSAGE WILL BE RETRIEVED.
/*****
LQMMD.MSGID = MQM_NONE;
LQMMD.CORRELID = MQCI_NONE;

/*****
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.
/*****
LQMMD.OPTIONS = MQGMD_NO_SYNCPOINT;

/*****
/* SET UP LENGTH OF MESSAGE BUFFER.
/*****
BUFFLEN = LENGTH(BUFFER);

/*****
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.
/*
/*****

CALL MQGET (HCONN,
            HOBJ,
            LQMMD,
            LMQGMD,
            BUFFERLEN,
            BUFFER,
            DATALEN,
            COMPCODE,
            REASON);

```



This example demonstrates how to use the MQGET call with the wait option and

```

/*****
/* TEST THE COMPLETION CODE OF THE GET CALL.                               */

```



```

:
:
        CALL DO_WORK;

:
:
        END;
        OTHERWISE DO;          /* FAILURE CASE */
/*****
/* ISSUE AN ERROR MESSAGE SHOWING THE COMPLETION CODE    */
/* AND THE REASON CODE.                                   */
*****/

:
:
        CALL ERROR_ROUTINE;

:
:
        END;
        END;

:
:
DO_WORK: PROC;

:
:
        IF ECB_POSTED
           THEN DO;

```



```

/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.          */
/* MD AND GMD SET UP AS REQUIRED.                      */
/*                                                     */
/*****

```

```

    CALL MQGET (HCONN,
                HOBJ,
                LMQMD,
                LMQGMD,
                BUFFLEN,
                BUFFER,
                DATALEN,
                COMPCODE,
                REASON);

```

```

END GET_MSG;

```

```

NO_MSG: PROC;

```

```

:
:
END NO_MSG;

```

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```

%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****
/* WORKING STORAGE DECLARATIONS          */
/*****
    DCL COMPCODE          BINARY FIXED (31);
    DCL REASON            BINARY FIXED (31);
    DCL HCONN             BINARY FIXED (31);
    DCL HOBJ              BINARY FIXED (31);
    DCL OPTIONS           BINARY FIXED (31);
    DCL SELECTORCOUNT    BINARY FIXED (31);
    DCL INTATTRCOUNT    BINARY FIXED (31);
    DCL 1 SELECTOR_TABLE,
        3 SELECTORS(5YFIXED-2*****/

/*                                     */
01FIXED                                */

```

```
/* HOBJ WAS SET BY PREVIOUS MOPEN REQUEST. */
/* */
/*****/
CALL MQINQ (HCONN,
```

```

        SELECTORCOUNT = 2;
        INTATTRCOUNT  = 2;

        CHARATTRLENGTH = 0;

/******
/*
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.
/*
/******

        CALL MQSET (HCONN,
                    HOBJ,
                    SELECTORCOUNT,
                    SELECTORS,
                    INTATTRCOUNT,
                    INTATTRS,
                    CHARATTRLENGTH,
                    CHARATTRS,
                    COMPCODE,
                    REASON);

/******
/* TEST THE COMPLETION CODE OF THE SET CALL.
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING
/* THE COMPLETION CODE AND THE REASON CODE.
/******

        IF COMPCODE /= MQCC_OK
            THEN DO;

```

The WebSphere MQ C include files are listed in WebSphere MQ Constants. They are installed in the following directories or libraries:

Platform	Installation directory or library
AIX	/usr/mqm/inc/
i5/OS	QMQM/H
UNIX platforms	/opt/mqm/inc/
Windows systems	\Program Files\IBM\WebSphere MQ\Tools\c\include
z/OS	thlqual.SCSQC370

Note: For UNIX platforms, the include files are symbolically linked into /usr/include.

For more information on the structure of directories, see the WebSphere MQ System Administration LOi(o)560.7m00i:othelanguage include filesebSpherf.AsbSpherf.u

Platform	Installation directory or library
Windows	\Program Files\IBM\WebSphere MQ\Tools\cobol\copybook (for Micro Focus COBOL) \Program Files\IBM\WebSphere MQ\Tools\cobol\copybook\VAcobol (for IBM VisualAge COBOL)
z/OS	thlqual.SCSQCOBC

Include in your program only those files that you need. Do this with one or more COPY statements after a level-01 declaration. This means that you can include multiple versions of the structures in a program if necet23(meam)-331(if)-333(OCes)-3330gaiMQV,

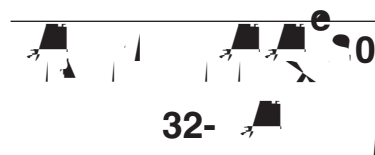
WebSphere MQ for z/OS provides two assembler-language macros containing the named constants, and one macro to generate each structure.

They are listed in WebSphere MQ Constants and installed in



These types never change size and are available on both 32-bit and 64-bit WebSphere MQ platforms:

Name	Length
MQLONG	4 bytes
MQULONG	4 bytes
MQINT32	4 bytes
MQUINT32	4 bytes
MQINT64	8 bytes
MQUINT64	8 bytes



This section is included for comparison and is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes
long double	16 bytes

Note that on AIX and Linux PPC a long double is 8 bytes.

pointer	4 bytes
ptrdiff_t	4 bytes
size_t	4 bytes
time_t	4 bytes
clock_t	4 bytes
wchar_t	4 bytes

Note that on AIX a wchar_t is 2 bytes.



This section is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	8 bytes

Name	Length
float	4 bytes
double	8 bytes
long double	16 bytes

Note that on AIX and Linux PPC a long double is 8 bytes.

pointer	8 bytes
ptrdiff_t	8 bytes
size_t	8 bytes
time_t	8 bytes
clock_t	8 bytes

Note that on the other UNIX platforms a clock_t is 4 bytes.

wchar_t	4 bytes
---------	---------

Note that on AIX a wchar_t is 2 bytes.



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

Licensees of this program who wish to have information about it for the purpose

UNIX
VSE/ESA
zSeries®

VisualAge
WebSphere

VM/ESA
z/OS

64 bit platforms, coding standards 613

abend

AEY9 307

accounting using message context 49

AccountingToken field 49

adapter

batch 303

CICS 304

IMS 307

trace points 306

Address space models

HP-UX on IA64 (IPF) 383

ADS

in CICS 3270 bridge vectors 336

terminology with the CICS

bridge 354

used in CICS 3270 bridge vectors 332

AEY9 abend 307

alias queue

examples of when to use 56

overview 53

resolving queue name 105

- assembler language *(continued)*
 - preparing your program to run 405
 - support for 85
 - using constants and structures 612
 - using the MQI 85

- attributes
 - DefInputOpenOption 107
 - DefPriority 26
 - HardenGetBackout 47, 66
 - IndexType 144
 - inquiring about 211
 - MaxMsgLength 116, 132
 - MaxPriority 26
 - MsgDeliverySequence 31, 133
 - ProcessName 232
 - queue manager 51
 - queues 55
 - selectors 212
 - setting 211
 - Shareability 107
 - TrigData 232
 - TriggerControl 237
 - TriggerData 227
 - TriggerDepth 237
 - TriggerMsgPriority 237
 - TriggerType 237

- audit trail using message context 48

- authentication information 61

- authentication information object
 - attributes 61

- authority checking
 - alternate user authority on
 - MQOPEN 108
 - by MQCLOSE 101
 - by MQDISC 101
 - by MQOPEN 101

- automatically starting an application
 - an example 453
 - how triggering works 226
 - introduction 13

- backing out changes 47, 214

- backout, skipping 154

- BackoutCount field 47, 66

- base queue 56

- Basocsa1000(47apping)-332.8(Support)TJ1.5-1.25TD[(wiSad.071Tm.t000(47.)-3]bridgPr)17.7(o30ly)-332.7(starting)-3batc.7(co2W)92(ebSphe1Tf1(Basocsa1M-3

CICS adapter (*continued*)

using CEDF 307

CICS bridge

Application Data Structure

(ADS) 317

Application Data Structure

terminology 354

applications on z/OS 317

COMMAREA data 317

debugging applications 352

distributed programming 321, 340

DPL programs 317

error handling 350

legacy applications 317

managing MsgId and CorrelId 346

setting bridge request queue

options 350

setting MQCIH fields 322, 344

setting MQMD fields 321, 343

unit of work 346

using DPL programs 318

CICS DPL bridge

application programming 320

COMMAREA data 318

managing MsgId and CorrelId 323

managing units of work 323

message structure 319

transactions in the distributed

environment 321

CICS Execution Diagnostic Facility

(CEDF) 307

CICS sample transaction 471

CKQC transaction 94, 414

CKTI transaction 241, 246

client (WebSphere MQ)

LU 6.2 link library 381

triggering support 226

WebSphere MQ clients and servers 6

what it is 4

cluster

what it is 3

cluster queue

MQOPEN option 106

overview 54

clusters (message affinities)

WebSphere MQ techniques 14

COBOL

CICS applications 374, 400, 401

copy files 610

examples

MQCLOSE 563

MQCONN 559

MQDISC 560

MQGET 567

MQGET with signaling 570

MQGET with wait option 568

MQINQ 572

MQOPEN fd[(Applcds-1000(317TJ1.5-1r-29.MQOPyi-1.25T(2C(with)-332.82.8(tn)-1000(568)]PUT*[(MQDIS4)-1000(568)]PUT1*[(MQDIS5)-1000(568)]

data conversion (*continued*)
MQXCNVC call 74
UNIX environment 205

fields (*continued*)

MsgId 140
Persistence 46
Priority 26
PutAppName 49
PutApplType 49
PutDate 49
PutMsgRecFields 114
PutMsgRecOffset 114
PutMsgRecPtr 114
PutTime 49
RecsPresent 114
ReplyToQ 48
ReplyToQMgr 48
Report 19
ResolvedQMgrName 113
ResolvedQName 113
ResponseRecOffset 115
ResponseRecPtr 115
StrucId 113
UserIdentifier 49
Version 113
WaitInterval 130, 151

format

control information 23

- MCA (message channel agent), definition
 - of 2
- message
 - backed out 47
 - browsing 158
 - using index 160
 - browsing and removing 161
 - browsing in logical order 161
 - browsing when message length unknown 161
 - channel agent definition 2
 - confirm arrival 19
 - confirm delivery 19
 - context
 - MQOPEN options 108
 - MQPUT options 118
 - types 48
 - control information 17
 - copying 158
 - creating 17
 - data 16, 115
 - data conversion
 - considerations 24
 - MQGET 157
 - data format 23
 - datagram 18
 - definition 2
 - descriptor

MQGET, using the call with signaling
Assembler example 588
C language example 548
COBOL example 570
PL/I example 602

MQGET, using the call with the wait option
Assembler example 586
C language example 547
COBOL example 568
PL/I example 601

MQGMO 129

MQGMO_*
ACCEPT_TRUNCATED_MSG 132

MQGMO_BROWSE_*
MSG_UNDER_CURSOR 161

MQGMO_BROWSE_FIRST 159

MQGMO_BROWSE_MSG_UNDER_CURSOR 159

MQGMO_BROWSE_NEXT 159

MQGMO_CONVERT 157

MQGMO_MARK_SKIP_BACKOUT 67
explanation 154

MQGMO_MSG_UNDER_CURSOR 161

MQGMO_WAIT 151

MQI (Message Queue Interface)

calls 72
client library files 74
data definition files 74
dealing with failure of a call 65
elementary data types 74
IMS applications 356
library files 74
overview 14
structures 74
stub programs 74
using System/390 assembler 85

MQI client

LU 6.2 link library 381

MQIA_* values 212

MQIIH 359

MQINQ

call parameters 212
use of selectors 212
when it fails 213

MQINQ, using the call

C language example 550
COBOL example 572
PL/I example 605

MQINQ, using the MQINQ and MQSET calls

Assembler example 590

MQMD

overview 17
setting fields with the CICS
bridge 321, 343
versions 17
when using MQGET 128
when using MQPUT 112

MQMT_* values 18

MQOD 103

MQOO_* values 106

MQOPEN

browse cursor 159
call parameters 102
MQOO_* values 106
object handle 101
resolving local queue names 108

MQOPEN (*continued*)

using MQOD 103

using options parameter 106

MQOPEN, using the call to create a dynamic queue

Assembler example 579
C language example 540
COBOL example 560
PL/I example 594

MQOPEN, using the call to open aoA dynamic queue

Assembler example 579
C language example 540
COBOL example 572
PL/I 605

MQOPENreou

MQOPEN_* 4ursor 159

call par7(MQPUT)(MQPUT)rsor17

obgA_* 4ursor

expl7(MQPUT)-1(values)-10i[(use)-3[(when)-332.8(it)-332.12values

MQINQ

MQOPENvalsing,shewh itpar7(MQPUT)(MQPUT)rsor 605

Assembler 83ample 579

C language e4ample 540

COBOL e5ample 572

PL/I 590

LU call

MQODN(to)-332.7dic queue

use par(00019)1000(605)TJ-1.SEin(ACCE1.25TD[(MQINQ,)-338.7(using)-332.7(the)-332.7(call)]TJ1.5

order of message retrieval 133
origin context 49
OTMA sense codes 361

parameters

Buffer 115
BufferLength 132
DataLength 132
Options 106

performance

design hints and tips 15
MQGET and buffer size 132
MQGET for a particular message 141
MQPUT1 111
persistent messages 47

permanent dynamic queue,

properties 58

Persistence field 46

PL/I

CMQEPP 612

CMQP 612

examples

MQCLOSE 596
MQCONN 593
MQDISC 594
MQGET 599
MQGET with signaling 602
MQGET with wait option 601
MQINQ 605
MQOPEN for dynamic queue 594
MQOPEN for existing queue 595
MQPUT 597
MQPUT1 598
MQSET 606

include files 612

support for 88

planning a WebSphere MQ

application 9

platform support

list of 15

positive action notification (PAN)

report 19

print message (sample for WebSphere

MQ for z/OS) 506

Priority field 26

priority in messages 26

problem delivering a message,

overview 47

problem determination

abend codes issued by the CICS

adapter 306

trace points in CICS adapter 306

using CEDF with the CICS

adapter 307

problem determination, use of report

message 67

process definition object

attributes 61

example to create one 231

opening 101

rules for naming 63

triggering prerequisite 231

what it is 227

ProcessName 242

ProcessName attribute 232

programming languages 81

properties

message properties 26

Publication consumer 175

publisher

applications

similarity with point-to-point 166

types 166

writing 166

fixed topics 166

variable topics 166

Publisher

application 167, 170

fixed topic 167

variable topic 170

put (sample for WebSphere MQ for

z/OS) 499

put-message options 112

PutApplName field 49

PutApplType field 49

PutDate field 49

PutMsgRecFields field 114

PutMsgRecOffset field 114

PutMsgRecPtr field 114

PutTime field 49

putting

messages 111

one message 120

putting messages to a distribution list

the MQPMR structure 125

QMQM library 610

QSG (queue-sharing group) 8

what it is 4, 52

queue

alias 53, 56

application 227

attributes 55

authority check on MQOPEN 101

base 56

channel 54

closing 101, 110

cluster 54

creating 53

dead-letter 59, 69

dead-letter on WebSphere MQ for

z/OS 530

definition 3

dynamic

permanent 58

temporary 57

dynamic, creation of 109

event 55

exclusive access 107

handle 101

initiation 59, 228

introduction to 52

local definition 53

model 57, 109

name resolution 64

name resolution when remote 109

object handle 101

opening 101

order of messages 31

queue (continued)

remote

definition 53

putting messages 117

using 2.7(naming)-nusing 2.7(naming)-nusing 2.7(naming)-nusing

reenterable assembler-language

programs 87

reference messages 149

remote queue

definition 53

using 55

using loi9n2c*TD[(definiti9n2c*ofsages)-10000(53)]TJT*[(usi9n2c*MQOPENSages)-1000(149-)]TJ1.5-1.25T*[(r)1ply(ence)-332.7(messages)-180(149)]TJT*

r

rmesueueusing 149

r

rRablerence medqages

uplicaefin-gen(eetedqages)-100(149-)]TJ1.5-1.25o(es[(r)1portqti9n2c332.7(messages)-180(149)]TJsolu2c*TD[(definitir)17.8(emote-33cefinam)-332.7(m643

using emote-33ce8inamrsend

qstubicaefistdy \mathbf{m} th

trigger (*continued*)

monitor

for WebSphere MQ for i5/OS 243

what it is 228

- WebSphere MQ for Solaris
 - build TUXEDO server
 - environment 473, 474
 - building your application 392
 - C compiler 392
 - CICS support 396
 - link libraries 394
 - sample programs 425
- WebSphere MQ for Windows
 - amqrspin.dll 486
 - amqsspin.c 486
 - authentication
 - Kerberos 403
 - NTLM 403
 - build TUXEDO server
 - environment 477, 479
 - building your application 397
 - channel-exit program 402
 - CICS support 399
 - context acceptor, security exit 402
 - context initiator, security exit 402
 - Kerberos
 - authentication 403

Spine information:

WebSphere MQ

A ca

P a

G de

Ve. 7.0