MQSeries®

Administration Interface Programming Guide and Reference

SC34-5390-00

MQSeries®

Administration Interface Programming Guide and Reference

SC34-5390-00

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 123.

First edition (March 1999)

This edition applies to the following products:

- MQSeries for AIX® V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2® Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT® V5.1

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

Order publications through your IBM® representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

Contents

Figures

Tables

Tables

About this book

This book describes the administration interface for MQSeries. This part of the product is referred to as the *MQSeries Administration Interface (MQAI)*.

The MQAI is a programming interface that simplifies the use of PCF messages to configure MQSeries. It is supplied as part of the following products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

Who this book is for

The information in this book is intended for system and application programmers who want to administer MQSeries.

What you need to know to understand this book

To understand this book, you need to have some knowledge of MQSeries and you need to know how to write programs in the C programming language. An understanding of how to use Programmable Command Formats (PCFs) is also useful. For more information about PCFs, see Chapter 8, “Definitions of the Programmable Command Formats” in the *MQSeries Programmable System Management* book.

How to use this book

This book is divided into the following chapters:

Chapter 1, “Introduction to the MQSeries Administration Interface (MQAI)” on page 1
Introduces the MQAI and related terminology.

Chapter 2, “Using data bags” on page 5
Describes the use of data bags.

Chapter 3, “Configuring MQSeries using mqExecute” on page 13
Describes how to send administration commands.

Chapter 4, “Exchanging data between applications” on page 17
Describes other uses of the MQAI.

Chapter 5, “MQAI Reference” on page 21
Contains reference material necessary to use the MQAI.

Chapter 6, “Examples of using the MQAI” on page 83
Lists and explains sample programs.

Chapter 7, “Advanced topics” on page 101
Describes indexing, data conversion, and the use of the message descriptor.

Appendix A, "Return codes" on page 105	Lists return codes relating to the MQAI.
Appendix B, "Constants in C" on page 115	Lists MQAI constants.
Appendix C, "Header files" on page 119	Lists and explains header files used by the MQAI.
Appendix D, "Selectors" on page 121	Describes user and system selectors.

Appearance of text in this book

This book uses the following type style:

CompCode	Example of the name of a parameter of a call, or the attribute of an object

MQSeries publications

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries “family” books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX V5.1
- MQSeries for AS/400® V4R2M1
- MQSeries for AT&T GIS UNIX® V2.2
- MQSeries for Digital OpenVMS V2.2
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Sun Solaris V5.1
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows® V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.1

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in “MQSeries Level 1 product publications” on page xv. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

MQSeries Brochure
The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing
MQSeries: An Introduction to Messaging and Queuing, GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide
The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication
The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Clients

The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration

The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem determination, and the dead-letter queue handler. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Command Reference

The *MQSeries Command Reference*, SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management

The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, Programmable Command Format (PCF) messages, and installable services.

MQSeries Messages

The *MQSeries Messages* book, GC33-1876, which describes “AMQ” messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

MQSeries Application Programming Guide

The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference

The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary

The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference* manual.

MQSeries Using C++

MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V4R2M1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries C++ is also supported by MQSeries clients supplied with these products and installed in the following environments:

- AIX
- HP-UX
- OS/2
- Sun Solaris
- Windows NT
- Windows 3.1
- Windows 95 and Windows 98

MQSeries Using Java™

MQSeries Using Java, SC34-5456, provides both guidance and reference information for users of the MQSeries Bindings for Java and the MQSeries Client for Java. MQSeries Java is supported by these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Administration Interface Programming Guide and Reference

The *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390, provides information for users of the MQAI. The MQAI is a programming interface that simplifies the way in which applications manipulate Programmable Command Format (PCF) messages and their associated data structures.

This book applies to the following MQSeries products only:

MQSeries for AIX V5.1
MQSeries for HP-UX V5.1
MQSeries for OS/2 Warp V5.1
MQSeries for Sun Solaris V5.1
MQSeries for Windows NT V5.1

MQSeries Queue Manager Clusters

MQSeries Queue Manager Clusters, SC34-5349, describes MQSeries clustering. It explains the concepts and terminology and shows how you can benefit by taking advantage of clustering. It details changes to the MQI, and summarizes the syntax of new and changed MQSeries commands. It shows a number of examples of tasks you can perform to set up and maintain clusters of queue managers.

This book applies to the following MQSeries products only:

MQSeries for AIX V5.1
MQSeries for HP-UX V5.1
MQSeries for OS/2 Warp V5.1
MQSeries for OS/390 V2.1
MQSeries for Sun Solaris V5.1
MQSeries for Windows NT V5.1

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX

MQSeries for AIX Version 5 Release 1 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 Version 4 Release 2.1 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX

MQSeries for AT&T GIS UNIX Version 2 Release 2 System Management Guide, SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS Version 2 Release 2 System Management Guide, GC33-1791

MQSeries for Digital UNIX

MQSeries for Digital UNIX Version 2 Release 2.1 System Management Guide, GC34-5483

MQSeries for HP-UX

MQSeries for HP-UX Version 5 Release 1 Quick Beginnings, GC33-1869

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp Version 5 Release 1 Quick Beginnings, GC33-1868

MQSeries for OS/390®

MQSeries for OS/390 Version 2 Release 1 Licensed Program Specifications, GC34-5377

MQSeries for OS/390 Version 2 Release 1 Program Directory

MQSeries for OS/390 Version 2 Release 1 System Management Guide, SC34-5374

MQSeries for OS/390 Version 2 Release 1 Messages and Codes, GC34-5375

MQSeries for OS/390 Version 2 Release 1 Problem Determination Guide, GC34-5376

MQSeries link for R/3

MQSeries link for R/3 Version 1 Release 2 User’s Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx Version 2 Release 2 System Management Guide, GC33-1768

MQSeries for Sun Solaris

MQSeries for Sun Solaris Version 5 Release 1 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2 Release 2 System Management Guide, GC33-1893

MQSeries for VSE/ESA™

MQSeries for VSE/ESA Version 2 Release 1 Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA Version 2 Release 1 System Management Guide, GC34-5364

MQSeries for Windows

MQSeries for Windows Version 2 Release 0 User's Guide, GC33-1822
MQSeries for Windows Version 2 Release 1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT Version 5 Release 1 Quick Beginnings, GC34-5389
MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387
MQSeries LotusScript™ Extension, SC34-5404

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC33-1141

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for UnixWare Version 1 Release 4.1 User's Guide, SC33-1379

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

BookManager® format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

`http://www.software.ibm.com/ts/mqseries/`

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

`http://www.adobe.com/`

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

`http://www.software.ibm.com/ts/mqseries/`

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

MQSeries Web site

The MQSeries product family Web site is at:

```
http://www.software.ibm.com/ts/mqseries/
```

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Chapter 1. Introduction to the MQSeries Administration Interface (MQAI)

This chapter describes:

- The main MQSeries Administration Interface (MQAI) concepts and terminology
- When the MQAI can be used
- How to use the MQAI

MQAI concepts and terminology

The MQAI is a programming interface to MQSeries. It performs administration tasks on an MQSeries queue manager using *data bags*. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using the existing administration interface, Programmable Command Formats (PCFs). The MQAI offers easier manipulation of PCFs than using the MQGET and MQPUT calls. For more information about PCFs, see Chapter 6, “Introduction to Programmable Command Formats” in the *MQSeries Programmable System Management* book. For more information about data bags, see Chapter 2, “Using data bags” on page 5.

The data bag contains zero or more *data items*. These are ordered within the bag as they are placed into the bag. This is called the *insertion order*. Each data item contains a *selector* that identifies the data item and a *value* of that data item that can be either an integer, a string, or a handle of another bag.

There are two types of selector; *user selectors* and *system selectors*. These are described in Appendix D, “Selectors” on page 121. The selectors are usually unique, but it is possible to have multiple values for the same selector. In this case, an *index* identifies the particular occurrence of selector that is required. Indexes are described in “Indexing” on page 101.

A hierarchy of the above concepts is shown in Figure 1.

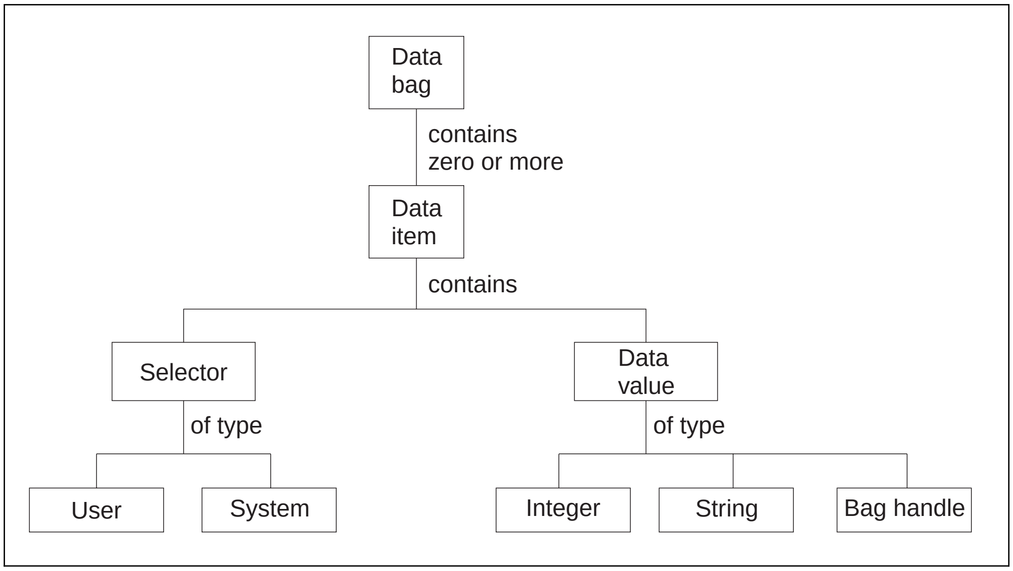


Figure 1. Hierarchy of MQAI concepts

Use of the MQAI

The MQAI can be used to do the following:

- Implement self-administering applications and administration tools. For example, the Active Directory Services provided on Windows NT Version 5 uses the MQAI. For more information about the Active Directory Service Interface, see Chapter 6, “Using the Active Directory Service Interfaces (ADSI)” in the *MQSeries for Windows NT Using the Component Object Model Interface* book.
- Simplify the use of PCF messages. The MQAI is an easy way to administer MQSeries; you do not have to write your own PCF messages and this avoids the problems associated with complex data structures.
- Handle error conditions more easily. It is difficult to get return codes back from the MQSeries commands (MQSC), but the MQAI makes it easier for the program to handle error conditions.

How do I use the MQAI?

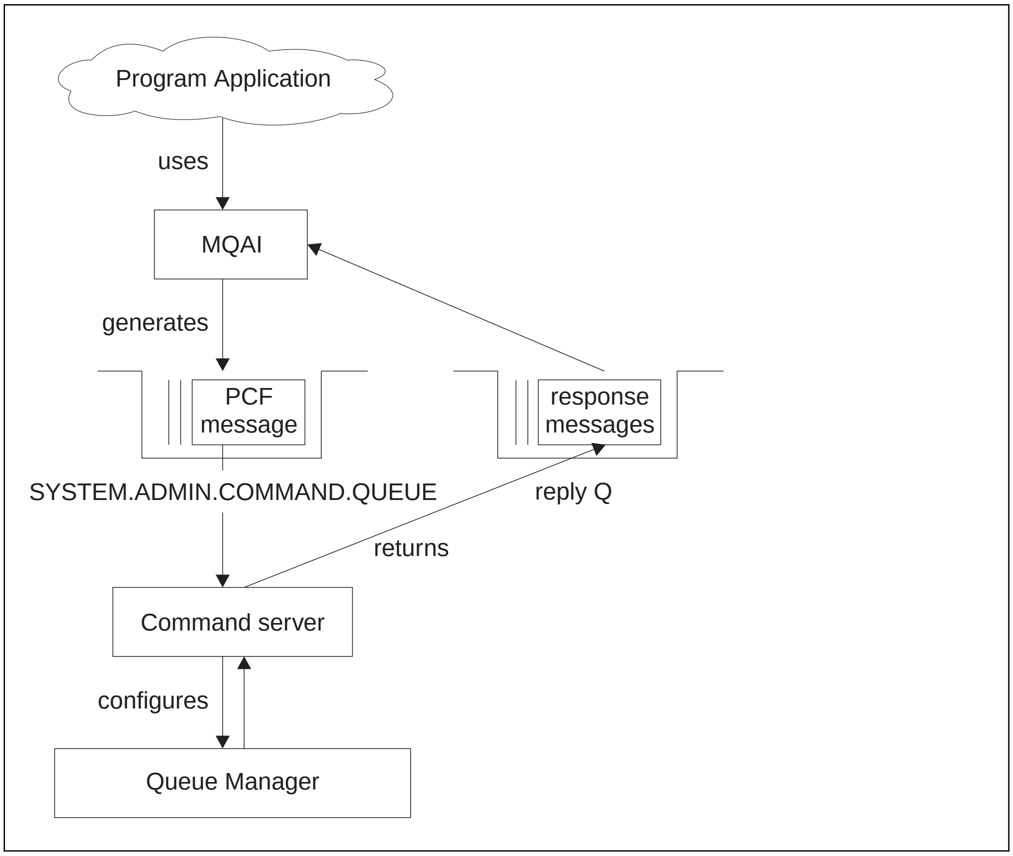


Figure 2. How the MQAI administers MQSeries

The MQAI provides easier programming access to PCF messages. To pass parameters in programs that are written using MQI calls, the PCF message must contain the command and details of the string or integer data. To do this, several statements are needed in your program for every structure and memory space must be allocated.

On the other hand, programs written using the MQAI pass parameters into the data bag and only one statement is required for each structure. The data bag removes the need for the programmer to handle arrays and allocate storage and provides some isolation from the details of PCF.

The MQAI administers MQSeries by sending PCF messages to the command server and waiting for a response as shown in Figure 2 on page 2.

Overview

The following instructions give a brief overview of 1) what you can use the MQAI for, and 2) how you use the MQAI. Further details are contained in the rest of this book.

To use the MQAI to administer MQSeries:

1. Decide on the task you want to carry out (for example, Change Queue).
2. Use Chapter 8, “Definitions of the Programmable Command Formats” in the *MQSeries Programmable System Management* guide as a reference to the commands and responses sent between an MQSeries systems management application program and an MQSeries queue manager.

   For example, look up the “Change Queue” command in the *MQSeries Programmable System Management* guide.
3. Choose the values of the selectors for the required parameters and any optional parameters that you want to set.
4. Create a data bag using the mqCreateBag call and enter values for each of these selectors using the mqAddInteger, mqAddString, and mqAddInquiry calls. This is described in Chapter 2, “Using data bags” on page 5.
5. Ensure the command server is running.
6. Using the mqExecute call, send the message to the command server and wait for a response. This is described in Chapter 3, “Configuring MQSeries using mqExecute” on page 13.

To use the MQAI to exchange data between applications:

- The sender must:
  1. Create a data bag intended to send the data using mqCreateBag. This is described in “Creating and deleting data bags” on page 5.
  2. Add the data to be sent in the bag using mqAddInteger or mqAddString. This is described in “Adding data items to bags” on page 6.
  3. Use the mqPutBag call to convert the data in the bag into a PCF message and put the message onto the required queue. This is described in “Putting and receiving data bags” on page 18.
- The receiver must:
  1. Create a data bag intended to receive the data using mqCreateBag. Again, this is described in “Creating and deleting data bags” on page 5.
  2. Use the mqGetBag call to get the PCF message from the queue and recreate a bag from the PCF message. For more information about this, see “Putting and receiving data bags” on page 18.

Using the MQAI is discussed in more detail in the chapters that follow.

Building your MQAI application

To build your application using the MQAI, you need to link to the same libraries as you do for MQSeries. For information on how to build your MQSeries applications, see Part 3, “Building an MQSeries application” in the *MQSeries Application Programming Guide*.

Chapter 2. Using data bags

A data bag is a means of handling properties (or parameters) of objects using the MQAI. This chapter discusses the configuration of data bags. It describes:

- The different types of bag and their uses
- How to create and delete data bags
- Types of data item
- How to add data items to data bags
- How to change information within a data bag
- How to count data items within a data bag
- How to delete data items
- How to inquire within data bags
- System items

Types of data bag

You can choose the type of data bag that you want to create depending on the task that you wish to perform:

user bag
A simple bag used for user data.

administration bag
A bag created for data used to administer MQSeries objects by sending administration messages to a command server. The administration bag automatically implies certain options as described in “Creating and deleting data bags.”

command bag
A bag also created for commands for administering MQSeries objects. However, unlike the administration bag, the command bag does not automatically imply certain options although these options are available. Again, these options are discussed in “Creating and deleting data bags.”

In addition, there is another type of data bag, the system bag. This is a bag created by the MQAI when a reply message is returned from the command server and placed into a user’s output bag. A system bag cannot be modified by the user.

Creating and deleting data bags

In order to use the MQAI, you must first create a data bag using the mqCreateBag call. As input to this call, you must supply one or more options to control the creation of the bag.

The *Options* parameter of the MQCreateBag call lets you choose whether you want to create a user bag, a command bag, or an administration bag. For a description of these types of bags, see “Types of data bag.”

If you choose to create a user bag or a command bag, you can choose one or more further options in order to:

- Use the list form when there are two or more adjacent occurrences of the same selector in a bag.
- Reorder the data items as they are added to a PCF message to ensure that the parameters are in their correct order.
- Check the values of user selectors for items that you add to the bag.

Administration bags automatically imply these options.

A data bag is identified by its handle. The bag handle is returned from mqCreateBag and must be supplied on all other calls that use the data bag.

For a full description of the mqCreateBag call, see “mqCreateBag” on page 37.

Deleting data bags

Any data bag that is created by the user must also be deleted using the mqDeleteBag call. For example, if a bag is created in the user code, it must also be deleted in the user code.

System bags are created and deleted automatically by the MQAI. For more information about this, see “Sending administration commands to the command server” on page 13. User code cannot delete a system bag.

For a full description of the mqDeleteBag call, see “mqDeleteBag” on page 41.

Types of data item

Here are the types of data item available within the MQAI:

- Integer
- Character-string
- Bag handle

When you have created a data bag, you can populate it with integer or character-string items. You can inquire about all three types of item.

Note: You cannot insert bag handles.

These data items can be user or system items. User items contain user data such as attributes of objects that are being administered. System items should be used for more control over the messages generated, for example, the generation of message headers. For more information about system items, see “System items” on page 11.

Adding data items to bags

The MQAI lets you add integer items and character-string items to bags and this is shown in Figure 3. The items are identified by a selector. Usually one selector identifies one item only, but this is not always the case. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag.

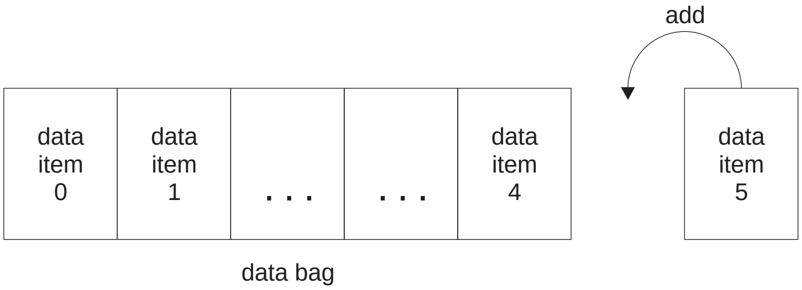


Figure 3. Adding data items

Add data items to a bag using the mqAdd* calls. To add integer items, use the mqAddInteger call as described in “mqAddInteger” on page 24. To add character-string items, use the mqAddString call as described in “mqAddString” on page 26.

Adding an inquiry command to a bag

The mqAddInquiry call is used to add an inquiry command to a bag. The call is specifically for administration purposes so it can be used with administration bags only. It lets you specify the selectors of attributes that you wish to inquire from MQSeries.

For a full description of the mqAddInquiry call, see “mqAddInquiry” on page 22.

Filtering and querying data items

When using the MQAI to inquire the attributes of MQSeries objects, you can control the data that is returned to your program in two ways.

1. You can ***filter*** the data that is returned using the mqAddInteger and mqAddString calls. This approach lets you specify a *Selector* and *ItemValue* pair, for example:

   ```
   mqAddInteger(inputbag, MQIA_Q_TYPE, MQQT_LOCAL)
   ```

   This example specifies that the queue type (*Selector*) must be local (*ItemValue*) and this specification must match the attributes of the object (in this case, a queue) that we are inquiring about.

   Other attributes that can be filtered correspond to the PCF Inquire* commands that can be found in the *MQSeries Programmable System Management* book. For example, if you want to inquire about the attributes of a channel, look up “Inquire Channel” in the *MQSeries Programmable System Management* book. The “Required parameters” and “Optional parameters” of the Inquire Channel command identify the selectors that you can use for filtering.

2. You can ***query*** particular attributes of an object using the mqAddInquiry call. This specifies the selector that you are interested in. If you do not specify the selector, all attributes of the object are returned.

Here is an example of filtering and querying the attributes of a queue:

```
/* Request information about all queues */
mqAddString(adminbag, MQCA_Q_NAME, “*”)

/* Filter attributes so that local queues only are returned */
mqAddInteger(adminbag, MQIA_Q_TYPE, MQQT_LOCAL)

/* Query the names and current depths of the local queues */
mqAddInquiry(adminbag, MQCA_Q_NAME)
mqAddInquiry(adminbag, MQIA_CURRENT_Q_DEPTH)

/* Send inquiry to the command server and wait for reply */
mqExecute(MQCMD_INQUIRE_Q, ...)
```

For more examples of filtering and querying data items, see Chapter 6, “Examples of using the MQAI” on page 83.

Changing information within a bag

The MQAI lets you change information within a bag using the mqSet* calls. You can:

1. Modify data items within a bag. The index allows an individual instance of a parameter to be replaced by identifying the occurrence of the item to be modified (see Figure 4).

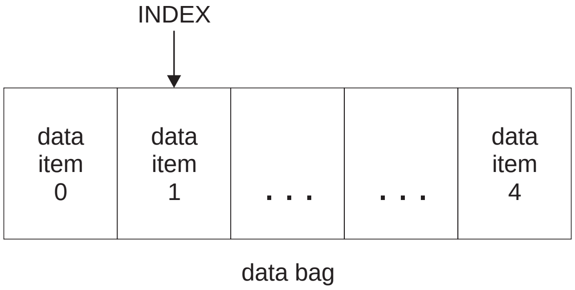


Figure 4. Modifying a single data item

2. Delete all existing occurrences of the specified selector and add a new occurrence to the end of the bag (see Figure 5). A special index value allows ***all*** instances of a parameter to be replaced.

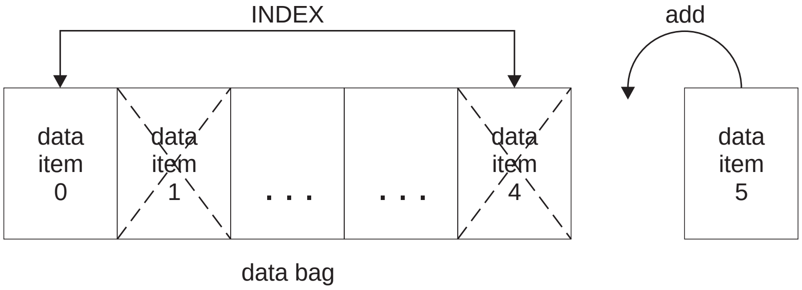


Figure 5. Modifying all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items.

The mqSetInteger call lets you modify integer items within a bag and the mqSetString call lets you modify character-string items. Alternatively, you can use these calls to delete all existing occurrences of the specified selector and add a new occurrence at the end of the bag. The data item can be a user item or a system item.

For a full description of these calls, see "mqSetInteger" on page 72 and "mqSetString" on page 75.

Counting data items

The mqCountItems call counts the number of user items, system items, or both, that are stored in a data bag, and returns this number. For example, `mqCountItems(`*`Bag`*`, 7, ...)`, returns the number of items in the bag with a selector of 7. It can count items by individual selector, by user selectors, by system selectors, or by all selectors.

Note: This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there may be fewer unique selectors in the bag than data items.

For a full description of the mqCountItems call, see "mqCountItems" on page 35.

Deleting data items

You can delete items from bags in a number of ways. You can:

- Remove one or more user items from a bag,
- Delete ***all*** user items from a bag, that is, *clear* a bag,
- Delete user items from the end of a bag, that is, *truncate* a bag.

Deleting data items from a bag using the mqDeleteItem call

The mqDeleteItem call removes one or more user items from a bag. The index is used to delete either:

1. A single occurrence of the specified selector (see Figure 6).

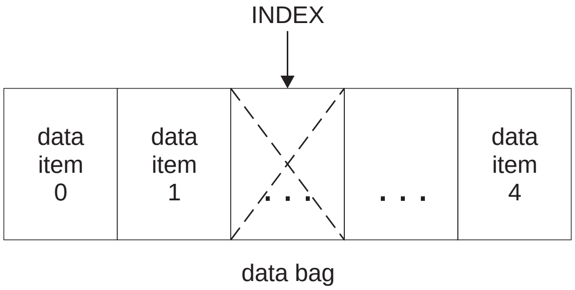


Figure 6. Deleting a single data item

or

2. All occurrences of the specified selector (see Figure 7).

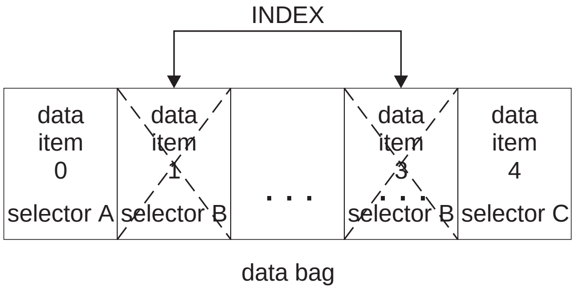


Figure 7. Deleting all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items. For example, the mqDeleteItem call does not preserve the index values of the data items that follow the deleted item because the indices are reorganized to fill the gap that remains from the deleted item.

For a full description of the mqDeleteItem call, see “mqDeleteItem” on page 43.

Clearing a bag using the mqClearBag call

The mqClearBag call removes all user items from a user bag and resets system items to their initial values. System bags contained within the bag are also deleted.

For a full description of the mqClearBag call, see “mqClearBag” on page 34.

Truncating a bag using the mqTruncateBag call

The mqTruncateBag call reduces the number of user items in a user bag by deleting the items from the end of the bag, starting with the most recently added item. For example, it can be used when using the same header information to generate more than one message.

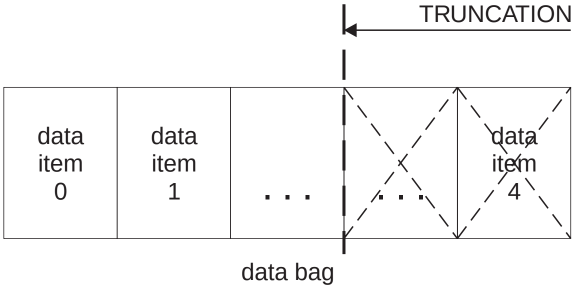


Figure 8. Truncating a bag

For a full description of the mqTruncateBag call, see “mqTruncateBag” on page 81.

Inquiring within data bags

You can inquire about:

- The value of an integer item using the mqInquireInteger call. For a full description of the mqInquireInteger call, see “mqInquireInteger” on page 57.
- The value of a character-string item using the mqInquireString call. For a full description of the mqInquireString call, see “mqInquireString” on page 63.
- The value of a bag handle using the mqInquireBag call. For a full description of the mqInquireBag call, see “mqInquireBag” on page 54.

You can also inquire about the type (integer, character string, or bag handle) of a specific item using the mqInquireItemInfo call. For a full description of the mqInquireItemInfo call, see “mqInquireItemInfo” on page 60.

System items

System items can be used for:

- The generation of PCF headers. System items can control the PCF command identifier, control options, message sequence number, and command type.
- Data conversion. System items handle the character set identifier for the character-string items in the bag.

Like all data items, system items consist of a selector and a value. For information about these selectors and what they are for, see Appendix D, “Selectors” on page 121.

System items are unique. One or more system items can be identified by a system selector. There is only one occurrence of each system selector.

Most system items can be modified (see “Changing information within a bag” on page 8), but the bag-creation options cannot be changed by the user. You cannot delete system items (see “Deleting data items” on page 9).

Chapter 3. Configuring MQSeries using mqExecute

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager and wait for any response messages. The easiest way to do this is by using the mqExecute call. This handles the exchange with the command server and returns responses in a bag.

Sending administration commands to the command server

The mqExecute call sends an administration command message as a nonpersistent message and waits for any responses. Responses are returned in a response bag. These might contain information about attributes relating to several MQSeries objects or a series of PCF error response messages, for example. Therefore, the response bag could contain a return code only or it could contain *nested bags*.

Response messages are placed into system bags that are created by the system. For example, for inquiries about the names of objects, a system bag is created to hold those object names and the bag is inserted into the user bag. Handles to these bags are then inserted into the response bag and the nested bag can be accessed by the selector MQHA_BAG_HANDLE. The system bag stays in storage, if it is not deleted, until the response bag is deleted.

The concept of *nesting* is shown in Figure 9.

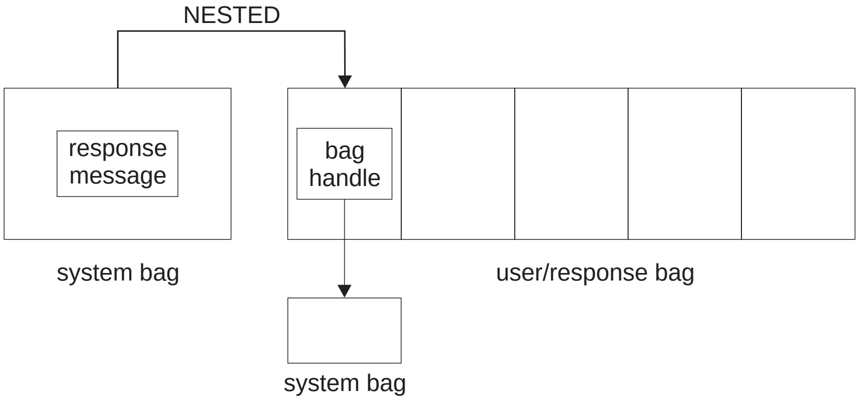


Figure 9. Nesting

As input to the mqExecute call, you must supply:

- An MQI connection handle.
- The command to be executed. This should be one of the MQCMD_* values.

  Note: If this value is not recognized by the MQAI, the value is still accepted. However, if the mqAddInquiry call was used to insert values into the bag, this parameter must be an INQUIRE command recognized by the MQAI. That is, the parameter should be of the form MQCMD_INQUIRE_*.
- Optionally, a handle of the bag containing options that control the processing of the call. This is also where you can specify the maximum time in milliseconds that the MQAI should wait for each reply message.

- A handle of the administration bag that contains details of the administration command to be issued.
- A handle of the response bag that receives the reply messages.

The following are optional:

- An object handle of the queue where the administration command is to be placed.

  If no object handle is specified, the administration command is placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. This is the default.
- An object handle of the queue where reply messages are to be placed.

  You can choose to place the reply messages on a dynamic queue that is created automatically by the MQAI. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

Example code

Here are some example uses of the mqExecute call.

The example shown in figure 10 creates a local queue (with a maximum message length of 100 bytes) on a queue manager:

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Create a queue    */
/* Supply queue name */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Supply queue type */
mqAddString(hbagRequest, MQIA_Q_TYPE, MQQT_LOCAL)

/* Maximum message length is an optional parameter */
mqAddString(hbagRequest, MQIA_MAX_MSG_LENGTH, 100)

/* Ask the command server to create the queue */
mqExecute(MQCMD_CREATE_Q, hbagRequest, hbagResponse)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 10. Using mqExecute to create a local queue

The example shown in Figure 11 inquires about all attributes of a particular queue. The mqAddInquiry call identifies all MQSeries object attributes of a queue to be returned by the Inquire parameter on mqExecute.

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Inquire about a queue by supplying its name */
/* (other parameters are optional) */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Request the command server to inquire about the queue */
mqExecute(MQCMD_INQUIRE_Q, hbagRequest, hbagResponse)

/* If it worked, the attributes of the queue are returned */
/* in a system bag within the response bag */
mqInquireBag(hbagResponse, MQHA_BAG_HANDLE, 0, &hbagAttributes)

/* Inquire the name of the queue and its current depth */
mqInquireString(hbagAttributes, MQCA_Q_NAME, &stringAttribute)
mqInquireString(hbagAttributes, MQIA_CURRENT_Q_DEPTH, &integerAttribute)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 11. Using mqExecute to inquire about queue attributes

Using mqExecute is the simplest way of administering MQSeries, but lower-level calls, mqBagToBuffer and mqBufferToBag, can be used. For more information about the use of these calls, see Chapter 4, “Exchanging data between applications” on page 17.

For sample programs, see Chapter 6, “Examples of using the MQAI” on page 83.

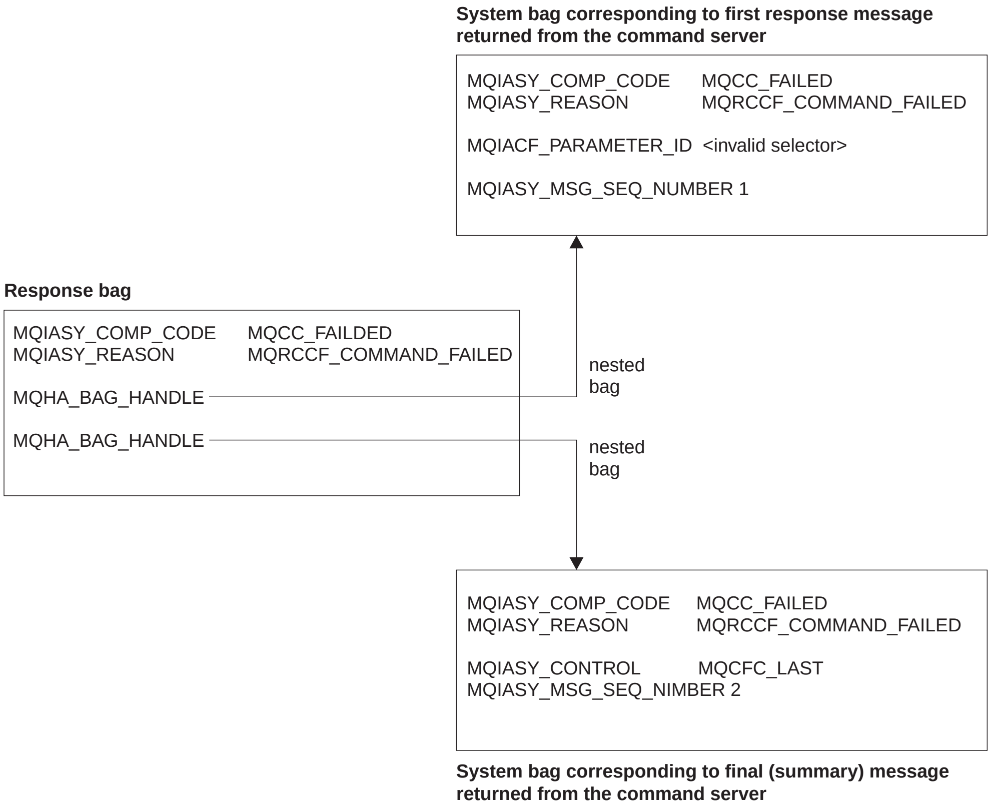
Hints and tips for configuring MQSeries

The MQAI is a programming interface that uses PCF messages to send administration commands to the command server rather than dealing directly with the command server itself. Here are some tips for configuring MQSeries using the MQAI:

- Character strings in MQSeries are blank padded to a fixed length. Using C, null-terminated strings can normally be supplied as input parameters to MQSeries’ programming interfaces.
- To clear the value of a string attribute, set it to a single blank rather than an empty string.
- It is recommended that you know in advance the attributes that you want to change and that you inquire on just those attributes. This is because the number of attributes that can be returned by the Inquire Queue (Response) command (see “Inquire Queue (Response)” in the *MQSeries Programmable System Management*) is higher than the number of attributes that can be changed using the Change Queue command (see “Change Queue” in the *MQSeries Programmable System Management*). Therefore, you are not recommended to attempt to modify all the attributes that you inquire.

- If an MQAI call fails, some detail of the failure is returned to the response bag. Further detail can then be found in a nested bag that can be accessed by the selector MQHA_BAG_HANDLE. For example, if an mqExecute call fails with a reason code of MQRCCF_COMMAND_FAILED, this information is returned in the response bag. However, a possible reason for this reason code is that a selector specified was not valid for the type of command message and this detail of information is found in a nested bag that can be accessed via a bag handle.

  The following diagram shows this:


System bag corresponding to first response message returned from the command server
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIACF_PARAMETER_ID <invalid selector>
MQIASY_MSG_SEQ_NUMBER 1
Response bag
MQIASY_COMP_CODE MQCC_FAILDED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQHA_BAG_HANDLE
MQHA_BAG_HANDLE
nested bag
nested bag
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIASY_CONTROL MQCFC_LAST
MQIASY_MSG_SEQ_NIMBER 2
System bag corresponding to final (summary) message returned from the command server

Chapter 4. Exchanging data between applications

The MQAI can also be used to exchange data between applications. The application data is sent in PCF format and packed and unpacked by the MQAI. If your message data consists of integers and character strings, you can use the MQAI to take advantage of MQSeries' built-in data conversion for PCF data. This avoids the need to write data-conversion exits. To exchange data, the sender must first create the message and send it to the receiving application. Then, the receiver must read the message and extract the data. This can be done in two ways:

1. Converting bags and buffers, that is, using the mqBagToBuffer and mqBufferToBag calls.
2. Putting and getting bags, that is, using the mqPutBag and mqGetBag calls to send and receive PCF messages.

Both of these options are described in this chapter.

Note: You cannot convert a bag containing nested bags into a message.

Converting bags and buffers

To send data between applications, firstly the message data is placed in a bag. Then, the data in the bag is converted into a PCF message using the mqBagToBuffer call. The PCF message is sent to the required queue using the MQPUT call. This is shown in Figure 12. For a full description of the mqBagToBuffer call, see "mqBagToBuffer" on page 29.

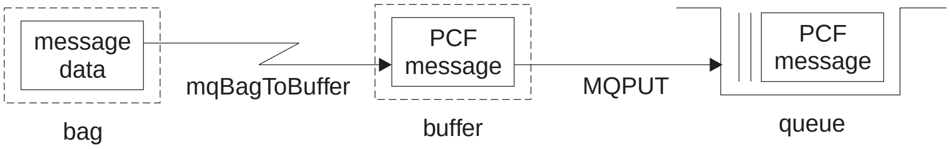


Figure 12. Converting bags to PCF messages

To receive data, the message is received into a buffer using the MQGET call. The data in the buffer is then converted into a bag using the mqBufferToBag call, providing the buffer contains a valid PCF message. This is shown in Figure 13. For a full description of the mqBufferToBag call, see "mqBufferToBag" on page 32.

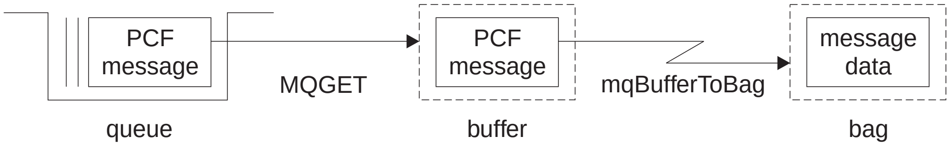


Figure 13. Converting PCF messages to bag form

Putting and receiving data bags

Data can also be sent between applications by putting and getting data bags using the mqPutBag and mqGetBag calls. This lets the MQAI handle the buffer rather than the application. The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue and the mqGetBag call removes the message from the specified queue and converts it back into a data bag. Therefore, the mqPutBag call is the equivalent of the mqBagToBuffer call followed by MQPUT, and the mqGetBag is the equivalent of the MQGET call followed by mqBufferToBag.

Note: If you choose to use the mqGetBag call, the PCF details within the message must be correct; if they are not, an appropriate error results and the PCF message is not returned.

Sending PCF messages to a specified queue

To send a message to a specified queue, the mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are left unchanged after the call.

As input to this call, you must supply:

- An MQI connection handle.
- An object handle for the queue on which the message is to be placed.
- A message descriptor. For more information about the message descriptor, see “MQMD - Message descriptor” in the *MQSeries Application Programming Reference*.
- Put Message Options using the MQPMO structure. For more information about the MQPMO structure, see “MQPMO - Put message options” in the *MQSeries Application Programming Reference*.
- The handle of the bag to be converted to a message.

  Note: If the bag contains an administration message and the mqAddInquiry call was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI.

For a full description of the mqPutBag call, see “mqPutBag” on page 69.

Receiving PCF messages from a specified queue

To receive a message from a specified queue, the mqGetBag call gets a PCF message from a specified queue and converts the message data into a data bag.

As input to this call, you must supply:

- An MQI connection handle.
- An object handle of the queue from which the message is to be read.
- A message descriptor. Within the MQMD structure, the `Format` parameter must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.

Note: If the message is received within a unit of work (that is, with the MQGMO_SYNCPOINT option) and the message has an unsupported format, the unit of work can be backed out. The message is then reinstated on the queue and can be retrieved using the MQGET call instead of the mqGetBag call.

For more information about the message descriptor, see “MQMD - Message descriptor” in the *MQSeries Application Programming Reference*.

- Get Message Options using the MQGMO structure. For more information about the MQGMO structure, see “MQGMO - Get-message options” in the *MQSeries Application Programming Reference*.
- The handle of the bag to contain the converted message.

For a full description of the mqGetBag call, see “mqGetBag” on page 51.

Chapter 5. MQAI Reference

This chapter contains reference information for the MQAI. There are three types of call:

- Data-bag manipulation calls for configuring data bags:
  - “mqAddInquiry” on page 22
  - “mqAddInteger” on page 24
  - “mqAddString” on page 26
  - “mqClearBag” on page 34
  - “mqCountItems” on page 35
  - “mqCreateBag” on page 37
  - “mqDeleteBag” on page 41
  - “mqDeleteItem” on page 43
  - “mqInquireBag” on page 54
  - “mqInquireInteger” on page 57
  - “mqInquireItemInfo” on page 60
  - “mqInquireString” on page 63
  - “mqSetInteger” on page 72
  - “mqSetString” on page 75
  - “mqTruncateBag” on page 81
- Command calls for sending and receiving administration commands and PCF messages:
  - “mqBagToBuffer” on page 29
  - “mqBufferToBag” on page 32
  - “mqExecute” on page 46
  - “mqGetBag” on page 51
  - “mqPutBag” on page 69
- Utility calls for handling blank-padded and null-terminated strings:
  - “mqPad” on page 67
  - “mqTrim” on page 79

These calls are described in alphabetical order in the following sections.

mqAddInquiry

Note: The mqAddInquiry call can be used with administration bags only; it is specifically for administration purposes.

The mqAddInquiry call adds a selector to an administration bag. The selector refers to an MQSeries object attribute that is to be returned by a PCF INQUIRE command. The value of the `Selector` parameter specified on this call is added to the end of the bag, as the value of a data item that has the selector value MQIACF_INQUIRY.

mqAddInquiry *(Bag, Selector, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input

Bag handle.

The bag must be an administration bag, that is, it must have been created with the MQCBO_ADMIN_BAG option on the mqCreateBag call. If the bag was not created this way, MQRC_BAG_WRONG_TYPE results.

Selector (MQLONG) – input

Selector of the MQSeries object attribute that is to be returned by the appropriate INQUIRE administration command.

CompCode (MQLONG) – output

Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInquiry call:

MQRC_BAG_WRONG_TYPE
Wrong type of bag for intended use.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. When the administration message is generated, the MQAI constructs an integer list with the MQIACF_*_ATTRS or MQIACH_*_ATTRS selector that is appropriate to the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call. It then adds the values of the attribute selectors specified by the mqAddInquiry call.
2. If the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call is not recognized by the MQAI, MQRC_INQUIRY_COMMAND_ERROR results. Instead of using the mqAddInquiry call, this can be overcome by using the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selector and the `ItemValue` parameter of the selector being inquired.

C language invocation

```
mqAddInquiry (Bag, Selector, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqAddInquiry Bag, Selector, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim Selector As Long 'Selector'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

Supported INQUIRE command codes

- MQCMD_INQUIRE_Q_MGR
- MQCMD_INQUIRE_PROCESS
- MQCMD_INQUIRE_Q
- MQCMD_INQUIRE_CHANNEL
- MQCMD_INQUIRE_CHANNEL_STATUS
- MQCMD_INQUIRE_NAMELIST
- MQCMD_INQUIRE_NAMELIST_NAMES
- MQCMD_INQUIRE_CLUSTER_Q_MGR

For an example that demonstrates the use of supported INQUIRE command codes, see “Inquiring about queues and printing information (amqsailq.c)” on page 88.

---

mqAddInteger

The mqAddInteger call adds an integer item identified by a user selector to the end of a specified bag.

```
mqAddInteger (Bag, Selector, ItemValue, CompCode, Reason)
```

Parameters

Bag (MQHBAG) – input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector (MQLONG) – input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results. If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the datatype of this occurrence must be the same as the datatype of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQLONG) – input
The integer value to be placed in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of this occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation

```
mqAddInteger (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   Item Value;/* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqAddInteger Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddString

The mqAddString call adds a character data item identified by a user selector to the end of a specified bag.

mqAddString (*Bag*, *Selector*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

Bag (MQHBAG) – input

Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) – input

Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the datatype of this occurrence must be the same as the datatype of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) – input

The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

> If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
>
> If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer (MQCHAR × *BufferLength*) – input

Buffer containing the character string.

The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of this occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation

```
mqAddString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer         /* Buffer containing item value */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqAddString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

---

mqBagToBuffer

The mqBagToBuffer call converts the bag into a PCF message in the supplied buffer.

```
mqBagToBuffer (OptionsBag, DataBag, BufferLength, Buffer, DataLength,
               CompCode, Reason)
```

Parameters

OptionsBag (MQHBAG) – input
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

DataBag (MQHBAG) – input
The handle of the bag to convert.

If the bag contains an administration message and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command that is recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

BufferLength (MQLONG) – input
Length in bytes of the buffer supplied.

If the buffer is too small to accommodate the message generated, MQRC_BUFFER_LENGTH_ERROR results.

Buffer (MQBYTE × *BufferLength*) – output
The buffer to hold the message.

DataLength (MQLONG) – output
The length in bytes of the buffer required to hold the entire bag. If the buffer is not long enough, the contents of the buffer are undefined but the `DataLength` is returned.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBagToBuffer call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid or buffer too small. (Required length returned in *DataLength*.)

MQRC_DATA_LENGTH_ERROR
DataLength parameter not valid (invalid parameter address).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry used with a command code that is not recognized as an INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested bags.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items or a supported option has an invalid value.

MQRC_PARAMETER_MISSING
An administration message requires a parameter that is not present in the bag.

Note: This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes

1. The PCF message is generated with an encoding of MQENC_NATIVE for the numeric data.
2. The buffer that holds the message can be null if the `BufferLength` is zero. This is useful if you use the mqBagToBuffer call to calculate the size of buffer necessary to convert your bag.

C language invocation

```
mqBagToBuffer (OptionsBag, DataBag, BufferLength, Buffer, &DataLength,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG   BufferLength;  /* Buffer length */
MQBYTE  Buffer[n];     /* Buffer to contain PCF */
MQLONG  DataLength;     /* Length of PCF returned in buffer */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqBagToBuffer OptionsBag, DataBag, BufferLength, Buffer, DataLength,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim DataBag      As Long 'Data bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer to contain PCF'
Dim DataLength   As Long 'Length of PCF returned in buffer'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBufferToBag

The mqBufferToBag call converts the supplied buffer into bag form.

mqBufferToBag *(OptionsBag, BufferLength, Buffer, DataBag, CompCode, Reason*)

Parameters

OptionsBag (MQHBAG) – input
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

BufferLength (MQLONG) – input
Length in bytes of the buffer.

Buffer (MQBYTE × *BufferLength*) – input
Pointer to the buffer containing the message to be converted.

Databag (MQHBAG) – input/output
Handle of the bag to receive the message. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBufferToBag call:

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag. This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of second occurrence of selector differs from datatype of first occurrence.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value that is not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

The buffer must contain a valid PCF message. The encoding of numeric data in the buffer must be MQENC_NATIVE.

The Coded Character Set ID of the bag is unchanged by this call.

C language invocation

```
mqBufferToBag (OptionsBag, BufferLength, Buffer, DataBag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQLONG   BufferLength;  /* Buffer length  */
MQBYTE   Buffer[n];     /* Buffer containing PCF */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqBufferToBag OptionsBag, BufferLength, Buffer, DataBag,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing PCF'
Dim DataBag      As Long 'Data bag handle'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqClearBag

The mqClearBag call deletes all user items from the bag, and resets system items to their initial values.

mqClearBag *(Bag, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input

Handle of the bag to be cleared. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

CompCode (MQLONG) – output

Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqClearBag call:

MQRC_HBAG_ERROR

Bag handle not valid.

MQRC_SYSTEM_BAG_NOT_ALTERABLE

System bag cannot be altered or deleted.

Usage notes

1. If the bag contains system bags, they are also deleted.
2. The call cannot be used to clear system bags.

C language invocation

```
mqClearBag (Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqClearBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqCountItems

The mqCountItems call returns the number of occurrences of user items, system items, or both, that are stored in a bag with the same specific selector.

mqCountItems (*Bag*, *Selector*, *ItemCount*, *CompCode*, *Reason*)

Parameters

Bag (MQHBAG) – input
Handle of the bag whose items are to be counted. This can be a user bag or a system bag.

Selector (MQLONG) – input
Selector of the data items to count.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI. MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the specified selector is not present in the bag, the call succeeds and zero is returned for *ItemCount*.

The following special values can be specified for *Selector*:

MQSEL_ALL_SELECTORS
All user and system items are to be counted.

MQSEL_ALL_USER_SELECTORS
All user items are to be counted; system items are excluded from the count.

MQSEL_ALL_SYSTEM_SELECTORS
All system items are to be counted; user items are excluded from the count.

ItemCount (MQLONG) – output
Number of items of the specified type in the bag (can be zero).

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCountItems call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

Usage notes

This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there may be fewer unique selectors in the bag than data items.

C language invocation

```
mqCountItems (Bag, Selector, &ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG  ItemCount;      /* Number of items */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqCountItems Bag, Selector, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;      As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemCount As Long 'Number of items'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqCreateBag

The mqCreateBag call creates a new bag.

mqCreateBag (*Options*, *Bag*, *CompCode*, *Reason*)

Parameters

Options (MQLONG) – input

Options for creation of the bag.

The following are valid:

MQCBO_ADMIN_BAG

Specifies that the bag is for administering MQSeries objects. MQCBO_ADMIN_BAG automatically implies the MQCBO_LIST_FORM_ALLOWED, MQCBO_REORDER_AS_REQUIRED, and MQCBO_CHECK_SELECTORS options.

Administration bags are created with the MQIASY_TYPE system item set to MQCFT_COMMAND.

MQCBO_COMMAND_BAG

Specifies that the bag is a command bag. This is an alternative to the administration bag (MQCBO_ADMIN_BAG) and MQRC_OPTIONS_ERROR results if both are specified.

A command bag is processed in the same way as a user bag except that the value of the MQIASY_TYPE system item is set to MQCFT_COMMAND when the bag is created.

The command bag is also created for administering objects but they are not used to send administration messages to a command server as an administration bag is. The bag options assume the following default values:

- MQCBO_LIST_FORM_INHIBITIED
- MQCBO_DO_NOT_REORDER
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI will not change the order of data items or create lists within a message as with administration bags.

MQCBO_USER_BAG

Specifies that the bag is a user bag. This is the default bag-type option. User bags can also be used for the administration of MQSeries objects, but the MQCBO_LIST_FORM_ALLOWED and MQCBO_REORDER_AS_REQUIRED options should be specified to ensure correct generation of the administration messages.

User bags are created with the MQIASY_TYPE system item set to MQCFT_USER.

For user bags, one or more of the following options can be specified:

MQCBO_LIST_FORM_ALLOWED
Specifies that the MQAI is allowed to use the more compact list form in the message sent whenever there are two or more adjacent occurrences of the same selector in the bag. However, this option does not allow the items to be reordered. Therefore, if the occurrences of the selector are not adjacent in the bag, and MQCBO_REORDER_AS_REQUIRED is not specified, the MQAI cannot use the list form for that particular selector.

If the data items are character strings, these strings must have the same Character Set ID as well as the same selector, in order to be compacted into list form. If the list form is used, the shorter strings are padded with blanks to the length of the longest string.

This option should be specified if the message to be sent is an administration message but MQCBO_ADMIN_BAG is not specified.

Note: MQCBO_LIST_FORM_ALLOWED does not imply that the MQAI will definitely use the list form. The MQAI considers various factors in deciding whether to use the list form.

MQCBO_LIST_FORM_INHIBITED
Specifies that the MQAI is not allowed to use the list form in the message sent, even if there are adjacent occurrences of the same selector in the bag. This is the default list-form option.

MQCBO_REORDER_AS_REQUIRED
Specifies that the MQAI is allowed to change the order of the data items in the message sent. This option does not affect the order of the items in the sending bag.

This means that you can insert items into a data bag in any order, that is, the items do not need to be inserted in the way that they must appear in the PCF message, because the MQAI can reorder these items as required.

If the message is a user message, the order of the items in the receiving bag will be the same as the order of the items in the message; this may be different from the order of the items in the sending bag.

If the message is an administration message, the order of the items in the receiving bag will be determined by the message received.

This option should be specified if the message to be sent is an administration message but MQCBO_ADMIN is not specified.

MQCBO_DO_NOT_REORDER
Specifies that the MQAI is not allowed to change the order of data items in the message sent. Both the message sent and the receiving bag contain the items in the same order as they occur in the sending bag. This is the default ordering option.

MQCBO_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) should be checked to ensure that the selector is consistent with the datatype implied by the mqAddInteger, mqAddString, mqSetInteger, or mqSetString call.

For the integer calls, the selector must be in the range MQIA_FIRST through MQIA_LAST.

For the string calls, the selector must be in the range MQCA_FIRST through MQCA_LAST.

For the handle calls, the selector must be in the range MQHA_FIRST through MQHA_LAST.

The call fails if the selector is outside the valid range. Note that system selectors (selectors less than zero) are always checked, and if a system selector is specified, it must be one that is supported by the MQAI.

MQCBO_DO_NOT_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) should not be checked. This option allows any selector that is zero or positive to be used with any call. This is the default selectors option. Note that system selectors (selectors less than zero) are always checked.

MQCBO_NONE
Specifies that all options should have their default values. This is provided to aid program documentation, and should not be specified with any of the options that has a nonzero value.

The following list summarizes the default option values:

- MQCBO_USER_BAG
  - MQCBO_LIST_FORM_INHIBITIED
  - MQCBO_DO_NOT_REORDER
  - MQCBO_DO_NOT_CHECK_SELECTORS

Bag (MQHBAG) – output
The handle of the bag created by the call.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCreateBag call:

MQRC_HBAG_ERROR
Bag handle not valid (invalid parameter address or the parameter location is read-only).

MQRC_OPTIONS_ERROR
Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes

Any options used for creating your bag are contained in a system item within the bag when it is created.

C language invocation

```
mqCreateBag (Options, &Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   Options;       /* Bag options */
MQHBAG  Bag;            /* Bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqCreateBag Options, Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Options  As Long 'Bag options'
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteBag

The mqDeleteBag call deletes the specified bag.

mqDeleteBag (*Bag*, *CompCode*, *Reason*)

Parameters

Bag (MQHBAG) – input/output
The handle of the bag to be deleted. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_DELETABLE results if you specify the handle of a system bag. The handle is reset to MQHB_UNUSABLE_HBAG.

If the bag contains system-generated bags, they are also deleted.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteBag call:

MQRC_HBAG_ERROR
Bag handle not valid, or invalid parameter address, or parameter location is read only.

MQRC_SYSTEM_BAG_NOT_DELETABLE
System bag cannot be deleted.

Usage notes

1. Delete any bags created with mqCreateBag.
2. Nested bags are deleted automatically when the containing bag is deleted.

C language invocation

```
mqDeleteBag (&Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqDeleteBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;     As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteItem

The mqDeleteItem call removes one or more user items from a bag.

mqDeleteItem *(Bag, Selector, ItemIndex, CompCode, Reason)*

Parameters

Hbag (MQHBAG) – input

Handle of the bag to be modified.

This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if it is a system bag.

Selector (MQLONG) – input

Selector identifying the user item to be deleted.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

The following special values are valid:

MQSEL_ANY_SELECTOR

The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of items that contains both user and system items.

MQSEL_ANY_USER_SELECTOR

The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of user items.

If an explicit selector value is specified, but the selector is not present in the bag, the call succeeds if MQIND_ALL is specified for `ItemIndex`, and fails with reason code MQRC_SELECTOR_NOT_PRESENT if MQIND_ALL is not specified.

ItemIndex (MQLONG) – input

Index of the data item to be deleted.

The value must be zero or greater, or one of the following special values:

MQIND_NONE

This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results. If MQIND_NONE is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results.

MQIND_ALL

This specifies that all occurrences of the selector in the bag are to be deleted. If MQIND_ALL is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results. If MQIND_ALL is specified when the selector is not present within the bag, the call succeeds.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater. If `ItemIndex` identifies a system selector MQRC_SYSTEM_ITEM_NOT_DELETABLE results. If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of user items, and must be zero or greater.

If an explicit selector value is specified, `ItemIndex` is the index relative to the set of items that have that selector value, and can be MQIND_NONE, MQIND_ALL, zero, or greater.

If an explicit index is specified (that is, not MQIND_NONE or MQIND_ALL) and the item is not present in the bag, MQRC_INDEX_NOT_PRESENT results.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteItem call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE or MQIND_ALL specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag is read only and cannot be altered.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
System item is read only and cannot be deleted.

Usage notes

1. Either a single occurrence of the specified selector can be removed, or all occurrences of the specified selector.
2. The call cannot remove system items from the bag, or remove items from a system bag. However, the call can remove the handle of a system bag from a user bag. This way, a system bag can be deleted.

C language invocation

```
mqDeleteItem(Bag, Selector, ItemIndex, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Hbag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqDeleteItem Bag, Selector, ItemIndex, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqExecute

The mqExecute call sends an administration command message and waits for the reply (if expected).

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag, AdminQ,
           ResponseQ, CompCode, Reason)
```

Parameters

Hconn (MQHCONN) – input

MQI Connection handle.

This is returned by a preceding MQCONN call issued by the application.

Command (MQLONG) – input

The command to be executed.

This should be one of the MQCMD_* values. If it is a value that is not recognized by the MQAI servicing the mqExecute call, the value is still accepted. However, if mqAddInquiry was used to insert values in the bag, the `Command` parameter must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

OptionsBag (MQHBAG) – input

Handle of a bag containing options that affect the operation of the call.

This must be the handle returned by a preceding mqCreateBag call or the following special value:

MQHB_NONE

No options bag; all options assume their default values.

Only the options listed below can be present in the options bag (MQRC_OPTIONS_ERROR results if other data items are present).

The appropriate default value is used for each option that is not present in the bag. The following option can be specified:

MQIACF_WAIT_INTERVAL

This data item specifies the maximum time in milliseconds that the MQAI should wait for each reply message. The time interval must be zero or greater, or the special value MQWI_UNLIMITED; the default is thirty seconds. The mqExecute call completes either when all of the reply messages are received or when the specified wait interval expires without the expected reply message having been received.

Note: The time interval is an approximate quantity.

If the MQIACF_WAIT_INTERVAL data item has the wrong datatype, or there is more than one occurrence of that selector in the options bag, or the value of the data item is not valid, MQRC_WAIT_INTERVAL_ERROR results.

AdminBag (MQHBAG) – input

Handle of the bag containing details of the administration command to be issued.

All user items placed in the bag are inserted into the administration message that is sent. It is the application's responsibility to ensure that only valid parameters for the command are placed in the bag.

If the value of the MQIASY_TYPE data item in the command bag is not MQCFT_COMMAND, MQRC_COMMAND_TYPE_ERROR results. If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

ResponseBag (MQHBAG) – input

Handle of the bag where reply messages are placed.

The MQAI performs an mqClearBag call on the bag before placing reply messages in the bag. To retrieve the reply messages, the selector, MQIACF_CONVERT_RESPONSE, can be specified.

Each reply message is placed into a separate system bag, whose handle is then placed in the response bag. Use the mqInquireBag call with selector MQHA_BAG_HANDLE to determine the handles of the system bags within the reply bag, and those bags can then be inquired to determine their contents.

If some but not all of the expected reply messages are received, MQCC_WARNING with MQRC_NO_MSG_AVAILABLE results. If none of the expected reply messages is received, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE results.

AdminQ (MQHOBJ) – input

Object handle of the queue on which the administration message is to be placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

The following special value can be specified:

MQHO_NONE

This indicates that the administration message should be placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. If MQHO_NONE is specified, the application need not use MQOPEN to open the queue.

ResponseQ

Object handle of the queue on which reply messages are placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input and for inquiry.

The following special value can be specified:

MQHO_NONE

This indicates that the reply messages should be placed on a dynamic queue created automatically by the MQAI. The queue is created by opening SYSTEM.DEFAULT.MODEL.QUEUE, that must therefore have suitable characteristics. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

CompCode

Completion code.

Reason

Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqExecute call:

MQRC_*
Anything from the MQINQ, MQPUT, MQGET, or MQOPEN calls.

MQRC_CMD_SERVER_NOT_AVAILABLE
The command server that processes administration commands is not available.

MQRC_COMMAND_TYPE_ERROR
The value of the MQIASY_TYPE data item in the request bag is not MQCFT_COMMAND.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInteger call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested bags.

MQRC_NO_MSG_AVAILABLE
Some reply messages received, but not all. Reply bag contains system-generated bags for messages that were received.

MQRC_NO_MSG_AVAILABLE
No reply messages received during the specified wait interval.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value which is not valid.

MQRC_PARAMETER_MISSING
Administration message requires a parameter which is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_NOT_UNIQUE
Two or more instances of a selector exist within the bag for a mandatory parameter that permits one instance only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRCCF_COMMAND_FAILED
Command failed; details of failure are contained in system-generated bags within the reply bag.

Usage notes

1. If no *AdminQ* is specified, the MQAI checks to see if the command server is active before sending the administration command message. However, if the command server is not active, the MQAI does not start it. If you are sending a large number of administration command messages, you are recommended to open the SYSTEM.ADMIN.COMMAND.QUEUE yourself and pass the handle of the administration queue on each administration request.
2. Specifying the MQHO_NONE value in the *ResponseQ* parameter simplifies the use of the mqExecute call, but if mqExecute is issued repeatedly by the application (for example, from within a loop), the response queue will be created and deleted repeatedly. In this situation, it is better for the application itself to open the response queue prior to any mqExecute call, and close it after all mqExecute calls have been issued.
3. If the administration command results in a message being sent with a message type of MQMT_REQUEST, the call waits for the period of time given by the MQIACF_WAIT_INTERVAL data item in the options bag.
4. If an error occurs during the processing of the call, the response bag may contain some data from the reply message, but the data will usually be incomplete.

C language invocation

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;        /* MQI connection handle */
MQLONG   Command;      /* Command to be executed */
MQHBAG   OptionsBag;   /* Handle of a bag containing options */
MQHBAG   AdminBag;     /* Handle of administration bag containing
                       /* details of administration command */
MQHBAG   ResponseBag;  /* Handle of bag for response messages */
MQHOBJ   AdminQ        /* Handle of administration queue for
                          administration messages */
MQHOBJ   ResponseQ;    /* Handle of response queue for response
                          messages */
MQLONG   pCompCode;    /* Completion code */
MQLONG   pReason;      /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn       As Long 'MQI connection handle'
Dim Command     As Long 'Command to be executed'
Dim OptionsBag  As Long 'Handle of a bag containing options'
Dim AdminBag  As Long 'Handle of command bag containing details of
                         administration command'
Dim ResponseBag As Long 'Handle of bag for reply messages'
Dim AdminQ    As Long 'Handle of command queue for
                         administration messages'
Dim ResponseQ   As Long 'Handle of response queue for reply messages'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqGetBag

The mqGetBag call removes a message from the specified queue and converts the message data into a data bag.

mqGetBag *(Hconn, Hobj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason)*

Parameters

Hconn (MQHCONN) – input
MQI connection handle.

Hobj (MQHOBJ) – input
Object handle of the queue from which the message is to be retrieved. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input.

MsgDesc (MQMD) – input/output
Message descriptor (for more information, see “MQMD - Message descriptor” and “MQGMO - Get-message options” in the *MQSeries Application Programming Reference* manual.)

If the *Format* field in the message has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If, on entry to the call, the *Encoding* field in the application’s MQMD has a value other than MQENC_NATIVE and MQGMO_CONVERT is specified, MQRC_ENCODING_NOT_SUPPORTED results. Also, if MQGMO_CONVERT is not specified, the value of the *Encoding* parameter must be the retrieving application’s MQENC_NATIVE; if not, again MQRC_ENCODING_NOT_SUPPORTED results.

GetMsgOpts (MQGMO) – input/output
Get-message options (for more information, see Chapter 10, “Getting messages from a queue” and “Specifying MQGET options using the MQGMO structure” in the *MQSeries Application Programming Guide*).

MQGMO_ACCEPT_TRUNCATED_MSG cannot be specified; MQRC_OPTIONS_ERROR results if it is. MQGMO_LOCK and MQGMO_UNLOCK are not supported in a 16-bit or 32-bit Window environment. MQGMO_SET_SIGNAL is supported in a 32-bit Window environment only.

Bag (MQHBAG) – input/output
Handle of a bag into which the retrieved message is placed. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

MQHB_NONE
Gets the retrieved message. This provides a means of deleting messages from the queue.

If an option of MQGMO_BROWSE_* is specified, this value sets the browse cursor to the selected message; it is not deleted in this case.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating warning and error conditions can be returned from the mqGetBag call:

MQRC_*
Anything from the MQGET call or bag manipulation.

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag.

This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported; the value in the *Encoding* field of the MQMD must be MQENC_NATIVE.

MQRC_FORMAT_NOT_SUPPORTED
Format not supported; the *Format* name in the message is not MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF. If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of second occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. Only messages that have a supported format can be returned by this call. If the message has a format that is not supported, the message is discarded, and the call completes with an appropriate reason code.
2. If the message is retrieved within a unit of work (that is, with the MQGMO_SYNCPOINT option), and the message has an unsupported format, the unit of work can be backed out, reinstating the message on the queue. This allows the message to be retrieved by using the MQGET call in place of the mqGetBag call.

C language invocation

```
mqGetBag (hConn, hObj, &MsgDesc, &GetMsgOpts, hBag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  hConn;         /* MQI connection handle */
MQHOBJ   hObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Get-message options */
MQHBAG   hBag;          /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqGetBag (HConn, HObj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long 'MQI connection handle'
Dim HObj       As Long 'Object handle'
Dim MsgDesc    As Long 'Message descriptor'
Dim GetMsgOpts As Long 'Get-message options'
Dim Bag        As Long 'Bag handle'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

mqInquireBag

The mqInquireBag call inquires the value of a bag handle that is present in the bag. The data item can be a user item or a system item.

C language invocation

mqInquireBag (*Bag*, *Selector*, *ItemIndex*, *ItemValue*, *CompCode*, *Reason*)

Parameters

Bag (MQHBAG) – input

Bag handle to be inquired. The bag can be a user bag or a system bag.

Selector (MQLONG) – input

Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR

The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR

The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR

The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex (MQLONG) – input

Index of the data item to be inquired.

The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results.

The following special value can be specified:

MQIND_NONE

This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemValue (MQHBAG) – output
Value of the item in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
The `ItemValue` parameter is not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present within the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireBag (Bag, Selector, ItemIndex, &ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Index of the data item to be inquired */
MQHBAG   ItemValue;    /* Value of item in the bag */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqInquireBag (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item to be inquired'
Dim ItemValue As Long 'Value of item in the bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireInteger

The mqInquireInteger call requests the value of an integer data item that is present in the bag. The data item can be a user item or a system item.

mqInquireInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason*)

Parameters

Bag (MQHBAG) – input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) – input
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) – input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue (MQLONG) – output
The value of the item in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireInteger (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqInquireInteger Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

---

mqInquireItemInfo

The mqInquireItemInfo call returns information about a specified item in a bag. The data item can be a user item or a system item.

```
mqInquireItemInfo (Bag, Selector, ItemIndex, ItemType, OutSelector,
                   CompCode, Reason)
```

Parameters

Bag (MQHBAG) – input
Handle of the bag to be inquired.

The bag can be a user bag or a system bag.

Selector (MQLONG) – input
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not. The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex (MQLONG) – input
Index of the data item to be inquired.

The item must be present within the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The value must be zero or greater, or the following special value:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater. If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemType (MQLONG) – output

The datatype of the specified data item.

The following can be returned:

MQIT_BAG

Bag handle item.

MQIT_INTEGER

Integer item.

MQIT_STRING

Character-string item.

OutSelector (MQLONG) – output

Selector of the specified data item.

CompCode (MQLONG) – output

Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireItemInfo call:

MQRC_HBAG_ERROR

Bag handle not valid.

MQRC_INDEX_ERROR

MQIND_NONE specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT

No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_TYPE_ERROR

`ItemType` parameter not valid (invalid parameter address).

MQRC_OUT_SELECTOR_ERROR

`OutSelector` parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT

No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED

Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE

MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireItemInfo (Bag, Selector, ItemIndex, &OutSelector, &ItemType,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector identifying item */
MQLONG   ItemIndex;     /* Index of data item */
MQLONG   OutSelector;   /* Selector of specified data item */
MQLONG   ItemType;      /* Data type of data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqInquireItemInfo Bag, Selector, ItemIndex, OutSelector, ItemType,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag         As Long 'Bag handle'
Dim Selector    As Long 'Selector identifying item'
Dim ItemIndex   As Long 'Index of data item'
Dim OutSelector As Long 'Selector of specified data item'
Dim ItemType    As Long 'Data type of data item'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqInquireString

The mqInquireString call requests the value of a character data item that is present in the bag. The data item can be a user item or a system item.

mqInquireString *(Bag, Selector, ItemIndex, Bufferlength, Buffer,*
StringLength, CodedCharSetId, CompCode, Reason)

Parameters

Bag (MQHBAG) – input

Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) – input

Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The datatype of the item must be the same as the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR

The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR

The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR

The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) – input

Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE

This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) – input
Length in bytes of the buffer to receive the string. Zero is a valid value.

Buffer (MQCHAR × *BufferLength*) – output
Buffer to receive the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *StringLength* indicates the size of the buffer needed to accommodate the string without truncation.

StringLength (MQLONG) – output
The length in bytes of the string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId (MQLONG) – output
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation

```
mqInquireString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer to contain string */
MQLONG   StringLength;  /* Length of string returned */
MQLONG   CodedCharSetId /* Coded Character Set ID */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqInquireString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqPad

The mqPad call pads a null-terminated string with blanks.

mqPad (*String*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

String (PMQCHAR) – input

Null-terminated string. The null pointer is valid for the address of the *String* parameter, and denotes a string of zero length.

BufferLength (MQLONG) – input

Length in bytes of the buffer to receive the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR × *BufferLength*) – output

Buffer to receive the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

If the number of characters preceding the first null in the *String* parameter is greater than the *BufferLength* parameter, the excess characters are omitted and MQRC_DATA_TRUNCATED results.

CompCode (MQLONG) – output

Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqPad call:

MQRC_BUFFER_ERROR

Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR

Buffer length not valid.

MQRC_STRING_ERROR

String parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_STRING_TRUNCATED

Data too long for output buffer and has been truncated.

Usage notes

1. If the buffer pointers are the same, the padding is done in place. If not, at most *BufferLength* characters are copied into the second buffer; any space remaining, including the null-termination character, is overwritten with spaces.
2. If the *String* and *Buffer* parameters partially overlap, the result is undefined.

C language invocation

```
mqPad (String, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR   String;          /* String to be padded */
MQLONG   BufferLength;    /* Buffer length */
PMQCHAR  Buffer           /* Buffer to contain padded string */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqPutBag

The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are unchanged after the call.

mqPutBag *(Hconn, Hobj, MsgDesc, PutMsgOpts, Bag, CompCode, Reason)*

Parameters

Hconn (MQHCONN) – input
MQI connection handle.

Hobj (MQHOBJ) – input
Object handle of the queue on which the message is to be placed. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

MsgDesc (MQMD) – input/output
Message descriptor (for more information, see “MQMD - Message descriptor” and “MQPUT - Put message” in the *MQSeries Application Programming Reference* manual).

If the *Format* field has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If the *Encoding* field has a value other than MQENC_NATIVE, MQRC_ENCODING_NOT_SUPPORTED results.

PutMsgOpts (MQPMO) – input/output
Put-message options (for more information, see “MQPUT - Put message” and “MQPMO - Put message options” in the *MQSeries Application Programming Reference* manual.)

Bag (MQHBAG) – input
Handle of the data bag to be converted to a message.

If the bag contains an administration message, and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output

Reason code qualifying *CompCode*. The following reason codes indicating error and warning conditions can be returned from the mqPutBag call:

MQRC_*
Anything from the MQPUT call or bag manipulation.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported (value in *Encoding* field in MQMD must be MQENC_NATIVE).

MQRC_FORMAT_NOT_SUPPORTED
Format not supported (name in *Format* field in MQMD must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested bags.

MQRC_PARAMETER_MISSING
Administration message requires a parameter that is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqPutBag (HConn, HObj, &MsgDesc, &PutMsgOpts, Bag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  HConn;         /* MQI connection handle */
MQHOBJ   HObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Put-message options */
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqPutBag (HConn, HObj, MsgDesc, PutMsgOpts, Bag,
CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long  'MQI connection handle'
Dim HObj       As Long  'Object handle'
Dim MsgDesc    As MQMD  'Message descriptor'
Dim PutMsgOpts As MQPMO 'Put-message options'
Dim Bag        As Long  'Bag handle'
Dim CompCode   As Long  'Completion code'
Dim Reason     As Long  'Reason code qualifying CompCode'
```

mqSetInteger

The mqSetInteger call either modifies an integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

mqSetInteger (*Bag*, *Selector*, *ItemIndex*, *ItemValue*, *CompCode*, *Reason*)

Parameters

Bag (MQHBAG) – input

Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector (MQLONG) – input

Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) – input

This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described below; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue (MQLONG) – input
The integer value to be placed in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation

```
mqSetInteger (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   ItemValue;    /* Integer value */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqSetInteger Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetString

The mqSetString call either modifies a character data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

```
mqSetString (Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode,
             Reason)
```

Parameters

Bag (MQHBAG) – input

Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) – input

Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the datatype of the item must be the same as the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) – input

This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described below; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) – input
The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer (MQCHAR × *BufferLength*) – input
Buffer containing the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation

```
mqSetString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer containing string */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqSetString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqTrim

The mqTrim call trims the blanks from a blank-padded string, then terminates it with a null.

mqTrim *(BufferLength, Buffer, String, CompCode, Reason*)

Parameters

BufferLength (MQLONG) – input
Length in bytes of the buffer containing the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR × *BufferLength*) – input
Buffer containing the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

String (MQCHAR × (*BufferLength*+1)) – output
Buffer to receive the null-terminated string. The length of this buffer must be at least one byte greater than the value of the *BufferLength* parameter.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTrim call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_STRING_ERROR
String parameter not valid (invalid parameter address or buffer not completely accessible).

Usage notes

1. If the two buffer pointers are the same, the trimming is done in place. If they are not the same, the blank-padded string is copied into the null-terminated string buffer. After copying, the buffer is scanned backwards from the end until a nonspace character is found. The byte following the nonspace character is then overwritten with a null character.
2. If *String* and *Buffer* partially overlap, the result is undefined.

C language invocation

```
mqTrim (BufferLength, Buffer, String, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   BufferLength;     /* Buffer length */
PMQCHAR  Buffer;           /* Buffer containing blank-padded string */
MQCHAR   String[n+1];      /* String with blanks discarded */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqTruncateBag

The mqTruncateBag call reduces the number of user items in a user bag to the specified value, by deleting user items from the end of the bag.

mqTruncateBag *(Bag, ItemCount, CompCode, Reason*)

Parameters

Bag (MQHBAG) – input
Handle of the bag to be truncated. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

ItemCount (MQLONG) – input
The number of user items to remain in the bag after truncation. Zero is a valid value.

Note: The *ItemCount* parameter is the number of data items, not the number of unique selectors. (If there are one or more selectors that occur multiple times in the bag, there will be fewer selectors than data items before truncation.) Data items are deleted from the end of the bag, in the opposite order to which they were added to the bag.

If the number specified exceeds the number of user items currently in the bag, MQRC_ITEM_COUNT_ERROR results.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTruncateBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (value exceeds the number of user data items in the bag).

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. System items in a bag are not affected by mqTruncateBag; the call cannot be used to truncate system bags.
2. mqTruncateBag with an *ItemCount* of zero is not the same as the mqClearBag call. The former deletes all of the user items but leaves the system items intact, and the latter deletes all of the user items and resets the system items to their initial values.

C language invocation

```
mqTruncateBag (Bag, ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   ItemCount;     /* Number of items to remain in bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows NT only.)

```
mqTruncateBag Bag, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim ItemCount As Long 'Number of items to remain in bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

Chapter 6. Examples of using the MQAI

This chapter includes some example programs that demonstrate use of the MQAI. The samples perform the following tasks:

1. Create a local queue.
2. Print a list of all local queues and their current depths.
3. Display events on the screen using a simple event monitor.

Creating a local queue (amqsaicq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAICQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to create a local queue using the MQSeries  */
/*               Administration Interface (MQAI).                             */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAICQ is a sample C program that creates a local queue and is an     */
/*    example of the use of the mqExecute call.                               */
/*                                                                            */
/*     - The name of the queue to be created is a parameter to the program.   */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI bag.            */
/*       These are:-                                                          */
/*            - The name of the queue                                         */
/*            - The type of queue required, which, in this case, is local.    */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_CREATE_Q.      */
/*       The call generates the correct PCF structure.                        */
/*       The call receives the reply from the command server and formats into */
/*       the response bag.                                                    */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server then the code returned by the   */
/*       command server is retrieved from the system bag that is              */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAICQ has 2 parameters - the name of the local queue to be created      */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************/
```

Figure 14 (Part 1 of 5). AMQSAICQ.C: Creating a local queue

```
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

void CheckCallResult(MQCHAR *, MQLONG , MQLONG );
void CreateLocalQueue(MQHCONN, MQCHAR *);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                          /* handle to MQ™ connection        */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQLONG reason;                          /* reason code                     */

   /***************************************************************************/
   /* First check the required parameters                                     */
   /***************************************************************************/
   printf("Sample Program to Create a Local Queue\n");
   if (argc < 2)
   {
     printf("Required parameter missing - local queue name\n");
     exit(99);
   }

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 2)
      strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
      MQCONN(QMName, &hConn, &compCode, &connReason);

/******************************************************************************/
/* Report reason and stop if connection failed                                */
/******************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }

/******************************************************************************/
/* Call the routine to create a local queue, passing the handle to the        */
/* queue manager and also passing the name of the queue to be created.        */
/******************************************************************************/
   CreateLocalQueue(hConn, argv[1]);

   /***************************************************************************/
   /* Disconnect from the queue manager if not already connected              */
   /***************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }
   return 0;

}
```

Figure 14 (Part 2 of 5). AMQSAICQ.C: Creating a local queue

```
/******************************************************************************/
/*                                                                            */
/* Function:    CreateLocalQueue                                              */
/* Description: Create a local queue by sending a PCF command to the command  */
/*              server.                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the queue to be created                         */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: The mqExecute call is executed with the command MQCMD_CREATE_Q.     */
/*        The call generates the correct PCF structure.                       */
/*        The default options to the call are used so that the command is sent*/
/*        to the SYSTEM.ADMIN.COMMAND.QUEUE.                                  */
/*        The reply from the command server is placed on a temporary dynamic  */
/*        queue.                                                              */
/*        The reply is read from the temporary queue and formatted into the   */
/*        response bag.                                                       */
/*                                                                            */
/*        The completion code from the mqExecute call is checked and if there */
/*        is a failure from the command server then the code returned by the  */
/*        command server is retrieved from the system bag that is             */
/*        embedded in the response bag to the mqExecute call.                 */
/*                                                                            */
/******************************************************************************/
void CreateLocalQueue(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG reason;                         /* reason code                      */
   MQLONG compCode;                       /* completion code                  */
   MQHBAG commandBag = MQHB_UNUSABLE_HBAG; /* command bag for mqExecute       */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG resultBag;                      /* result bag from mqExecute        */
   MQLONG mqExecuteCC;                    /* mqExecute completion code        */
   MQLONG mqExecuteRC;                    /* mqExecute reason code            */

   printf("\nCreating Local Queue %s\n\n", qName);

   /***************************************************************************/
   /* Create a command Bag for the mqExecute call. Exit the function if the   */
   /* create fails.                                                           */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &commandBag, &compCode, &reason);
   CheckCallResult("Create the command bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;

   /***************************************************************************/
   /* Create a response Bag for the mqExecute call, exit the function if the  */
   /* create fails.                                                           */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
   CheckCallResult("Create the response bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;

   /***************************************************************************/
   /* Put the name of the queue to be created into the command bag. This will */
   /* be used by the mqExecute call.                                          */
   /***************************************************************************/
   mqAddString(commandBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, qName, &compCode,
               &reason);
   CheckCallResult("Add q name to command bag", compCode, reason);
```

Figure 14 (Part 3 of 5). AMQSAICQ.C: Creating a local queue

```
/**************************************************************************/
/* Put queue type of local into the command bag. This will be used by the  */
/* mqExecute call.                                                         */
/**************************************************************************/
mqAddInteger(commandBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type to command bag", compCode, reason);

/**************************************************************************/
/* Send the command to create the required local queue.                    */
/* The mqExecute call will create the PCF structure required, send it to   */
/* the command server and receive the reply from the command server into   */
/* the response bag.                                                       */
/**************************************************************************/
mqExecute(hConn,                    /* MQ connection handle                 */
          MQCMD_CREATE_Q,           /* Command to be executed               */
          MQHB_NONE,                /* No options bag                       */
          commandBag,               /* Handle to bag containing commands    */
          responseBag,              /* Handle to bag to receive the response*/
          MQHO_NONE,                /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,                /* Create a dynamic q for the response  */
          &compCode,                /* Completion code from the mqExecute   */
          &reason);                 /* Reason code from mqExecute call      */

if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n")
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("MQDISC", compCode, reason);
   exit(98);
}

/**************************************************************************/
/* Check the result from mqExecute call and find the error if it failed.   */
/**************************************************************************/
if ( compCode == MQCC_OK )
   printf("Local queue %s successfully created\n", qName);
else
{
   printf("Creation of local queue %s failed: Completion Code = %d
            qName, compCode, reason);
   if (reason == MQRCCF_COMMAND_FAILED)
   {
      /*********************************************************************/
      /* Get the system bag handle out of the mqExecute response bag.      */
      /* This bag contains the reason from the command server why the      */
      /* command failed.                                                   */
      /*********************************************************************/
      mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &resultBag, &compCode,
                   &reason);
      CheckCallResult("Get the result bag handle", compCode, reason);

      /*********************************************************************/
      /* Get the completion code and reason code, returned by the command  */
      /* server, from the embedded error bag.                              */
      /*********************************************************************/
      mqInquireInteger(resultBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                       &compCode, &reason);
      CheckCallResult("Get the completion code from the result bag",
                       compCode, reason);
      mqInquireInteger(resultBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                       &compCode, &reason);
      CheckCallResult("Get the reason code from the result bag", compCode,
                       reason);
      printf("Error returned by the command server: Completion code = %d :
              Reason = %d\n", mqExecuteCC, mqExecuteRC);
   }
}
```

Figure 14 (Part 4 of 5). AMQSAICQ.C: Creating a local queue

```
   /**************************************************************************/
   /* Delete the command bag if successfully created.                        */
   /**************************************************************************/
   if (commandBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&commandBag, &compCode, &reason);
      CheckCallResult("Delete the command bag", compCode, reason);
   }

   /**************************************************************************/
   /* Delete the response bag if successfully created.                       */
   /**************************************************************************/
   if (responseBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&responseBag, &compCode, &reason);
      CheckCallResult("Delete the response bag", compCode, reason);
   }
} /* end of CreateLocalQueue */

/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the   */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d :
                 Reason = %d\n", callText, cc, rc);

}
```

Figure 14 (Part 5 of 5). AMQSAICQ.C: Creating a local queue

Inquiring about queues and printing information (amqsailq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAILQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to inquire the current depth of the local   */
/*               queues using the MQSeries Administration Interface (MQAI).   */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAILQ is a sample C program that demonstrates how to inquire         */
/*    attributes of the local queue manager using the MQAI interface. In      */
/*    particular, it inquires the current depths of all the local queues.     */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI administration  */
/*       bag.                                                                 */
/*       These are:-                                                          */
/*            - The generic queue name "*"                                    */
/*            - The type of queue required. In this sample we want to         */
/*              inquire local queues.                                         */
/*            - The attribute to be inquired. In this sample we want the      */
/*              current depths.                                               */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_INQUIRE_Q.     */
/*       The call generates the correct PCF structure.                        */
/*       The default options to the call are used so that the command is sent */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                   */
/*       The reply from the command server is placed on a temporary dynamic   */
/*       queue.                                                               */
/*       The reply from the MQCMD_INQUIRE_Q command is read from the          */
/*       temporary queue and formatted into the response bag.                 */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server, then the code returned by      */
/*       command server is retrieved from the system bag that has been        */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/*     - If the call is successful, the depth of each local queue is placed   */
/*       in system bags embedded in the response bag of the mqExecute call.   */
/*       The name and depth of each queue is obtained from each of the bags   */
/*       and the result displayed on the screen.                              */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAILQ has 1 parameter - the queue manager name (optional)               */
/*                                                                            */
/******************************************************************************/
```

Figure 15 (Part 1 of 5). AMQSAILQ.C: Inquiring queues and printing information

```
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   /***************************************************************************/
   /* MQAI variables                                                          */
   /***************************************************************************/
   MQHCONN hConn;                          /* handle to MQ connection         */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;   /* admin bag for mqExecute         */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG qAttrsBag;                       /* bag containing q attributes     */
   MQHBAG errorBag;                        /* bag containing cmd server error */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */
   MQLONG qNameLength;                     /* Actual length of q name         */
   MQLONG qDepth;                          /* depth of queue                  */
   MQLONG i;                               /* loop counter                    */
   MQLONG numberOfBags;                    /* number of bags in response bag  */
   MQCHAR qName[MQ_Q_NAME_LENGTH+1];       /* name of queue extracted from bag*/


   printf("Display current depths of local queues\n\n");

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn, &compCode, &connReason);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed.                    */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason
      exit( (int)connReason);
   }
   /***************************************************************************/
   /* Create an admin bag for the mqExecute call                              */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &adminBag, &compCode, &reason);
   CheckCallResult("Create admin bag", compCode, reason);
```

Figure 15 (Part 2 of 5). AMQSAILQ.C: Inquiring queues and printing information

```
/***************************************************************************/
/* Create a response bag for the mqExecute call                            */
/***************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
CheckCallResult("Create response bag", compCode, reason);

/***************************************************************************/
/* Put the generic queue name into the admin bag                           */
/***************************************************************************/
mqAddString(adminBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, "*",
            &compCode, &reason);
CheckCallResult("Add q name", compCode, reason);

/***************************************************************************/
/* Put the local queue type into the admin bag                             */
/***************************************************************************/
mqAddInteger(adminBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type", compCode, reason);

/***************************************************************************/
/* Add an inquiry for current queue depths                                 */
/***************************************************************************/
mqAddInquiry(adminBag, MQIA_CURRENT_Q_DEPTH, &compCode, &reason);
CheckCallResult("Add inquiry", compCode, reason);

/***************************************************************************/
/* Send the command to find all the local queue names and queue depths.    */
/* The mqExecute call creates the PCF structure required, sends it to      */
/* the command server, and receives the reply from the command server into */
/* the response bag. The attributes are contained in system bags that are  */
/* embedded in the response bag, one set of attributes per bag.            */
/***************************************************************************/
mqExecute(hConn,                   /* MQ connection handle                 */
          MQCMD_INQUIRE_Q,         /* Command to be executed               */
          MQHB_NONE,               /* No options bag                       */
          adminBag,                /* Handle to bag containing commands    */
          responseBag,             /* Handle to bag to receive the response*/
          MQHO_NONE,               /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,               /* Create a dynamic q for the response  */
          &compCode,               /* Completion code from the mqExecute   */
          &reason);                /* Reason code from mqExecute call      */


/***************************************************************************/
/* Check the command server is started. If not exit.                       */
/***************************************************************************/
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n");
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("Disconnect from Queue Manager", compCode, reason);
   exit(98);
}
/***************************************************************************/
/* Check the result from mqExecute call. If successful find the current    */
/* depths of all the local queues. If failed find the error.               */
/***************************************************************************/
if ( compCode == MQCC_OK )                      /* Successful mqExecute    */
{
  /*************************************************************************/
  /* Count the number of system bags embedded in the response bag from the */
  /* mqExecute call. The attributes for each queue are in a separate bag.  */
  /*************************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags, &compCode,
               &reason);
  CheckCallResult("Count number of bag handles", compCode, reason);
```

Figure 15 (Part 3 of 5). AMQSAILQ.C: Inquiring queues and printing information

```
   for ( i=0; i<numberOfBags; i++)
   {
     /***********************************************************************/
     /* Get the next system bag handle out of the mqExecute response bag.   */
     /* This bag contains the queue attributes                              */
     /***********************************************************************/
     mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &qAttrsBag, &compCode,
                  &reason);
     CheckCallResult("Get the result bag handle", compCode, reason);

     /***********************************************************************/
     /* Get the queue name out of the queue attributes bag                  */
     /***********************************************************************/
     mqInquireString(qAttrsBag, MQCA_Q_NAME, 0, MQ_Q_NAME_LENGTH, qName,
                     &qNameLength, NULL, &compCode, &reason);
     CheckCallResult("Get queue name", compCode, reason);

     /***********************************************************************/
     /* Get the depth out of the queue attributes bag                       */
     /***********************************************************************/
     mqInquireInteger(qAttrsBag, MQIA_CURRENT_Q_DEPTH, MQIND_NONE, &qDepth,
                      &compCode, &reason);
     CheckCallResult("Get depth", compCode, reason);

     /***********************************************************************/
     /* Use mqTrim to prepare the queue name for printing.                  */
     /* Print the result.                                                   */
     /***********************************************************************/
     mqTrim(MQ_Q_NAME_LENGTH, qName, qName, &compCode, &reason)
     printf("%4d  %-48s\n", qDepth, qName);
   }
 }

 else                                               /* Failed mqExecute     */
 {
   printf("Call to get queue attributes failed: Completion Code = %d :
           Reason = %d\n", compCode, reason);
   /*************************************************************************/
   /* If the command fails get the system bag handle out of the mqExecute   */
   /* response bag. This bag contains the reason from the command server    */
   /* why the command failed.                                               */
   /*************************************************************************/
   if (reason == MQRCCF_COMMAND_FAILED)
   {
     mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag, &compCode,
                  &reason);
     CheckCallResult("Get the result bag handle", compCode, reason);

    /************************************************************************/
    /* Get the completion code and reason code, returned by the command     */
    /* server, from the embedded error bag.                                 */
    /************************************************************************/
    mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                     &compCode, &reason );
    CheckCallResult("Get the completion code from the result bag",
                     compCode, reason);
    mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                      &compCode, &reason);
    CheckCallResult("Get the reason code from the result bag",
                     compCode, reason);
    printf("Error returned by the command server: Completion Code = %d :
            Reason = %d\n", mqExecuteCC, mqExecuteRC);
  }
}
```

Figure 15 (Part 4 of 5). AMQSAILQ.C: Inquiring queues and printing information

```
  /****************************************************************************/
  /* Delete the admin bag if successfully created.                            */
  /****************************************************************************/
  if (adminBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&adminBag, &compCode, &reason);
     CheckCallResult("Delete the admin bag", compCode, reason);
  }

  /****************************************************************************/
  /* Delete the response bag if successfully created.                         */
  /****************************************************************************/
  if (responseBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&responseBag, &compCode, &reason);
     CheckCallResult("Delete the response bag", compCode, reason);
  }

  /****************************************************************************/
  /* Disconnect from the queue manager if not already connected               */
  /****************************************************************************/
  if (connReason != MQRC_ALREADY_CONNECTED)
  {
     MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("Disconnect from queue manager", compCode, reason);
  }
  return 0;
}
*******************************************************************************/
*                                                                              */
* Function: CheckCallResult                                                    */
*                                                                              */
*******************************************************************************/
*                                                                              */
* Input Parameters:  Description of call                                      */
*                    Completion code                                          */
*                    Reason code                                              */
*                                                                              */
* Output Parameters: None                                                      */
*                                                                              */
* Logic: Display the description of the call, the completion code and the     */
*        reason code if the completion code is not successful                  */
*                                                                              */
*******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
  if (cc != MQCC_OK)
        printf("%s failed: Completion Code = %d : Reason = %d\n",
                callText, cc, rc);
}
```

Figure 15 (Part 5 of 5). AMQSAILQ.C: Inquiring queues and printing information

Displaying events using an event monitor (amqsaiem.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAIEM.C                                                   */
/*                                                                            */
/* Description:  Sample C program to demonstrate a basic event monitor       */
/*               using the MQSeries Administration Interface (MQAI).         */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                        */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAIEM is a sample C program that demonstrates how to write a simple  */
/*    event monitor using the mqGetBag call and other MQAI calls.             */
/*                                                                            */
/*    The name of the event queue to be monitored is passed as a parameter    */
/*    to the program. This would usually be one of the system event queues:-  */
/*            SYSTEM.ADMIN.QMGR.EVENT        queue manager events             */
/*            SYSTEM.ADMIN.PERFM.EVENT       Performance events               */
/*            SYSTEM.ADMIN.CHANNEL.EVENT     Channel events                   */
/*    To monitor the queue manager event queue or the Performance event queue */
/*    the attributes of the queue manager will need to be changed to enable   */
/*    the events, refer to                                                    */
/*    Part 1, “Event monitoring”                                              */
/*    in the MQSeries Programmable System Management guide                    */
/*    for more information.                                                   */
/*    The queue manager attributes can be changed either by                   */
/*    MQSC commands or using the MQAI interface.                              */
/*    Channel events are enabled by default.                                  */
/*                                                                            */
/* Program logic                                                              */
/*    Connect to the queue manager.                                           */
/*    Open the requested event queue with the wait unlimited option.          */
/*    Wait for a message and when it arrives get the message from the queue   */
/*    and format it into an MQAI bag with the mqGetBag call.                  */
/*    There are many types of event messages and it is beyond the scope of    */
/*    this sample to program for all event messages. Instead print out the    */
/*    contents of the formatted bag.                                          */
/*    Loop around to wait for another message until either there is an error  */
/*    or the program is stopped by a user interrupt.                          */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAIEM has 2 parameters - the name of the event queue to be monitored    */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */
```

Figure 16 (Part 1 of 8). AMQSAIEM.C: Displaying events

```
/*****************************************************************************/
/* Function prototypes                                                       */
/*****************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);
void GetQEvents(MQHCONN, MQCHAR *);
int PrintBag(MQHBAG);
int PrintBagContents(MQHBAG, int);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                         /* handle to connection            */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QM name              */
   MQLONG reason;                         /* reason code                     */
   MQLONG connReason;                     /* MQCONN reason code              */
   MQLONG compCode;                       /* completion code                 */

   /**************************************************************************/
   /* First check the required parameters                                    */
   /**************************************************************************/
   printf("Sample Event Monitor (^C to stop)\n");
     if (argc < 2)
   {
     printf("Required parameter missing - event queue to be monitored\n")
     exit(99);
   }

   /**************************************************************************/
   /* Connect to the queue manager                                           */
   /**************************************************************************/
   if (argc > 2)
     strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(QMName, &hConn, &compCode, &connReason);

   /**************************************************************************/
   /* Report the reason and stop if the connection failed                    */
   /**************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }

   /**************************************************************************/
   /* Call the routine to open the event queue and format any event message  */
   /* read from the queue.                                                   */
   /**************************************************************************/
   GetQEvents(hConn, argv[1]);

   /**************************************************************************/
   /* Disconnect from the queue manager if not already connected             */
   /**************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }

   return 0;

}
```

Figure 16 (Part 2 of 8). AMQSAIEM.C: Displaying events

```
/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the   */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d : Reason = %d\n",
                 callText, cc, rc);

}


/******************************************************************************/
/*                                                                            */
/* Function: GetQEvents                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the event queue to be monitored                 */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic:   Open the event queue.                                             */
/*          Get a message off the event queue and format the message into     */
/*          a bag.                                                            */
/*          A real event monitor would need to be programmed to deal with     */
/*          each type of event that it receives from the queue. This is       */
/*          outside the scope of this sample so instead the contents of       */
/*          the bag are printed.                                              */
/*          The program waits forever for a message on the queue so the       */
/*          program must be terminated by a user interrupt (^C).              */
/*                                                                            */
/******************************************************************************/
void GetQEvents(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG openReason;                     /* MQOPEN reason code               */
   MQLONG reason;                         /* reason code                      */
   MQLONG compCode;                       /* completion code                  */
   MQHOBJ eventQueue;                     /* handle to event queue            */
   MQHBAG eventBag = MQHB_UNUSABLE_HBAG;  /* event bag to receive event msg */
   MQOD   od = {MQOD_DEFAULT};            /* Object Descriptor                */
   MQMD   md = {MQMD_DEFAULT};            /* Message Descriptor               */
   MQGMO  gmo = {MQGMO_DEFAULT};          /* get message options              */
   MQLONG bQueueOK = 1;                   /* keep reading msgs while true     */
   /***************************************************************************/
   /* Create an Event Bag in which to receive the event. Message exit the     */
   /* function if the create fails.                                           */
   /***************************************************************************/
   mqCreateBag(MQCBO_USER_BAG, &eventBag, &compCode, &reason);
   CheckCallResult("Create event bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;
```

Figure 16 (Part 3 of 8). AMQSAIEM.C: Displaying events

```
  /**************************************************************************/
  /* Open the event queue chosen by the user                                */
  /**************************************************************************/
  strncpy(od.ObjectName, qName, (size_t)MQ_Q_NAME_LENGTH);
  MQOPEN(hConn, &od, MQOO_INPUT_AS_Q_DEF+MQOO_FAIL_IF_QUIESCING, &eventQueue,
         &compCode, &openReason);
  CheckCallResult("Open event queue", compCode, openReason);

  /**************************************************************************/
  /* Set the GMO options to control the action of the get message from the  */
  /* queue.                                                                 */
  /**************************************************************************/
  gmo.WaitInterval = MQWI_UNLIMITED;      /* Wait forever for a message      */
  gmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING;
  gmo.Version = MQGMO_VERSION_2;          /* Avoid need to reset Message ID  */
  gmo.MatchOptions = MQMO_NONE;           /* and Correlation ID after every  */
                                          /* mqGetBag                        */

  /**************************************************************************/
  /* If open failed we cannot access the queue and must stop the monitor.   */
  /**************************************************************************/
  if (compCode != MQCC_OK)
    bQueueOK = 0;

  /**************************************************************************/
  /* Main loop to get an event message when it arrives                      */
  /**************************************************************************/
  while (bQueueOK)
  {
    printf("\nWaiting for an event\n");

    /************************************************************************/
    /* Get the message from the event queue and convert it into the event   */
    /* bag.                                                                 */
    /************************************************************************/
    mqGetBag(hConn, eventQueue, &md, &gmo, eventBag, &compCode, &reason);
    CheckCallResult("Get bag", compCode, reason);


    if (compCode != MQCC_OK)
       bQueueOK = 0;

    else
    {
      /**********************************************************************/
      /* Event message read - Print the contents of the event bag           */
      /**********************************************************************/
      if ( PrintBag(eventBag) )
          printf("\nError found while printing bag contents\n");

  }  /* end of msg found */
} /* end of main loop */
  /**************************************************************************/
  /* Close the event queue if successfully opened                           */
  /**************************************************************************/
  if (openReason == MQRC_NONE)
  {
     MQCLOSE(hConn, &eventQueue, MQCO_NONE, &compCode, &reason);
     CheckCallResult("Close event queue", compCode, reason);
  }
```

Figure 16 (Part 4 of 8). AMQSAIEM.C: Displaying events

```
   /***************************************************************************/
   /* Delete the event bag if successfully created.                          */
   /***************************************************************************/
   if (eventBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&eventBag, &compCode, &reason);
      CheckCallResult("Delete the event bag", compCode, reason);
   }

} /* end of GetQEvents */

/******************************************************************************/
/*                                                                            */
/* Function: PrintBag                                                         */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Calls PrintBagContents to display the contents of the bag.          */
/*                                                                            */
/******************************************************************************/
int PrintBag(MQHBAG dataBag)
{
    int errors;

    printf("\n");
    errors = PrintBagContents(dataBag, 0);
    printf("\n");

    return errors;
}

/******************************************************************************/
/*                                                                            */
/* Function: PrintBagContents                                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                    Indentation level of bag                                */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Count the number of items in the bag                                */
/*        Obtain selector and item type for each item in the bag.             */
/*        Obtain the value of the item depending on item type and display the */
/*        index of the item, the selector and the value.                      */
/*        If the item is an embedded bag handle then call this function again */
/*        to print the contents of the embedded bag increasing the            */
/*        indentation level.                                                  */
/*                                                                            */
/******************************************************************************/
```

Figure 16 (Part 5 of 8). AMQSAIEM.C: Displaying events

```
int PrintBagContents(MQHBAG dataBag, int indent)
{
   #define LENGTH 500                       /* Max length of string to be read*/
   #define INDENT 4                         /* Number of spaces to indent     */
                                            /* embedded bag display           */

   MQLONG  itemCount;                       /* Number of items in the bag     */
   MQLONG  itemType;                        /* Type of the item               */
   int     i;                               /* Index of item in the bag       */
   char    stringVal[LENGTH+1];             /* Value if item is a string      */
   MQLONG  stringLength;                    /* Length of string value         */
   MQLONG  ccsid;                           /* CCSID of string value          */
   MQLONG  iValue;                          /* Value if item is an integer    */
   MQLONG  selector;                        /* Selector of item               */
   MQHBAG  bagHandle;                       /* Value if item is a bag handle  */
   MQLONG  reason;                          /* reason code                    */
   MQLONG  compCode;                        /* completion code                */
   MQLONG  trimLength;                      /* Length of string to be trimmed */
   int     errors = 0;                      /* Count of errors found          */
   char    blanks[]= "                       "; /* Blank string used to       */
                                                 /* indent display            */

   /***************************************************************************/
   /* Count the number of items in the bag                                    */
   /***************************************************************************/
   mqCountItems(dataBag, MQSEL_ALL_SELECTORS, &itemCount, &compCode, &reason);

   if (compCode != MQCC_OK)
      errors++;
   else
   {
      printf("%.*sHandle:%d ", indent, blanks, dataBag);
      printf("%.*sSize:%d\n", indent, blanks, itemCount);
      printf("%.*sIndex: Selector: Value:\n", indent, blanks);
   }

   /***************************************************************************/
   /* If no errors found then display each item in the bag                    */
   /***************************************************************************/
   if (!errors)
   {
      for (i = 0; i < itemCount; i++)
      {
          /********************************************************************/
          /* First inquire the type of the item for each item in the bag      */
          /********************************************************************/
          mqInquireItemInfo(dataBag,             /* Bag handle                */
                            MQSEL_ANY_SELECTOR,  /* Item can have any selector*/
                            i,                   /* Index position in the bag */
                            &selector,           /* Actual value of selector  */
                                                 /* returned by call          */
                            &itemType,           /* Actual type of item       */
                                                 /* returned by call          */
                            &compCode,           /* Completion code           */
                            &reason);            /* Reason Code               */

          if (compCode != MQCC_OK)
             errors++;
```

Figure 16 (Part 6 of 8). AMQSAIEM.C: Displaying events

```
switch(itemType)
{
case MQIT_INTEGER:
     /***************************************************************/
     /* Item is an integer. Find its value and display its index,   */
     /* selector and value.                                         */
     /***************************************************************/
     mqInquireInteger(dataBag,        /* Bag handle               */
                      MQSEL_ANY_SELECTOR, /* Allow any selector     */
                      i,              /* Index position in the bag */
                      &iValue,        /* Returned integer value    */
                      &compCode       /* Completion code           */
                      &reason);       /* Reason Code               */

     if (compCode != MQCC_OK)
        errors++;
     else
        printf("%.*s  %-2d  %-4d  (%d)\n",
                indent, blanks, i, selector, iValue);
     break;

case MQIT_STRING:
     /***************************************************************/
     /* Item is a string. Obtain the string in a buffer, prepare    */
     /* the string for displaying and display the index, selector,  */
     /* string and character set ID.                                */
     /***************************************************************/
     mqInquireString(dataBag,         /* Bag handle               */
                     MQSEL_ANY_SELECTOR, /* Allow any selector      */
                     i,               /* Index position in the bag */
                     LENGTH,          /* Maximum length of buffer  */
                     stringVal,       /* Buffer to receive string  */
                     &stringLen       /* Actual length of string   */
                     &ccsid,          /* Coded character set ID    */
                     &reason);        /* Reason Code               */

     if (compCode == MQCC_FAILED)
         errors++;
     else
     {
        /************************************************************/
        /* Remove trailing blanks from the string and terminate with*/
        /* a null. First check that the string should not have been */
        /* longer than the maximum buffer size allowed.             */
        /************************************************************/
        if (stringLength > LENGTH)
           trimLength = LENGTH;
        else
           trimLength = stringLength;
        mqTrim(trimLength, stringVal, stringVal, &compCode, &reason);
        printf("%.*s  %-2d     %-4d'%s' %d\n",
                indent, blanks, i, selector, stringVal, ccsid);
     }
     break;
```

Figure 16 (Part 7 of 8). AMQSAIEM.C: Displaying events

```
          case MQIT_BAG:
               /**************************************************************/
               /* Item is an embedded bag handle, so call the function again  */
               /* to display the contents.                                    */
               /**************************************************************/
               mqInquireBag(dataBag,            /* Bag handle               */
                            MQSEL_ANY_SELECTOR,  /* Allow any selector       */
                            i,                   /* Index position in the bag */
                            &bagHandle,          /* Returned embedded bag hdle*/
                            &compCode,           /* Completion code          */
                            &reason);            /* Reason Code              */

               if (compCode != MQCC_OK)
                  errors++;
               else
               {
                  printf("%.*s  %-2d     %-4d     (%d)\n", indent, blanks,
                          i, selector, bagHandle);
                  printf("%.*sSystem Bag:\n", indent+INDENT, blanks);
                  PrintBagContents(bagHandle, indent+INDENT);
               }
               break;

          default:
               printf("%.*sUnknown item type", indent, blanks);
          }
      }
    }
    return errors;
 }
```

Figure 16 (Part 8 of 8). AMQSAIEM.C: Displaying events

Chapter 7. Advanced topics

This chapter discusses the following:

1. Indexing
2. Data conversion
3. Use of the message descriptor

Indexing

Each selector and value within a data item in a bag have three associated index numbers:

- The index relative to other items that have the same selector.
- The index relative to the category of selector to which the item belongs, that is user or system.
- The index relative to all the data items in the bag, that is user and system.

This allows indexing by user selectors, system selectors, or both as shown in Figure 17.

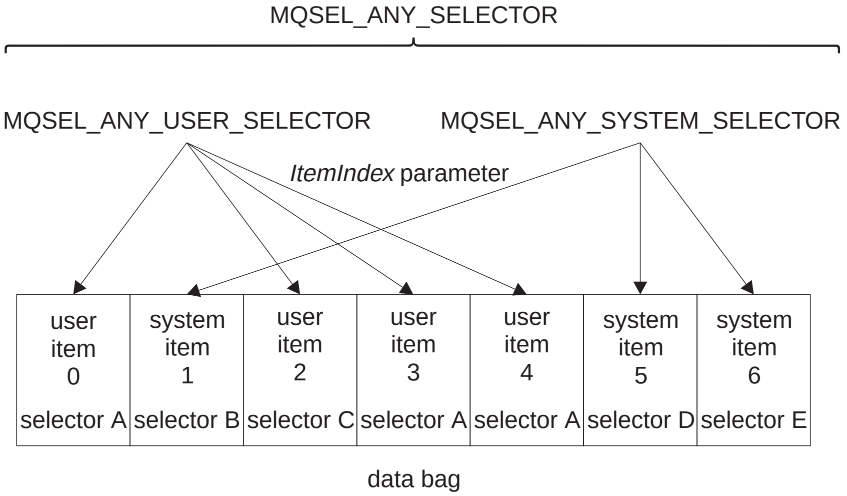


Figure 17. Indexing

In Figure 17, user item 3 (selector A) can be referred to by the following index pairs:

Selector	*ItemIndex*
selector A	1
MQSEL_ANY_USER_SELECTOR	2
MQSEL_ANY_SELECTOR	3

The index is zero-based like an array in C; if there are ‘n’ occurrences, the index ranges from zero through ‘n-1’, with no gaps.

Indexes are used when replacing or removing existing data items from a bag. When used in this way, the insertion order is preserved, but indexes of other data items can be affected. For examples of this, see “Changing information within a bag” on page 8 and “Deleting data items” on page 9.

The three types of indexing allow easy retrieval of data items. For example, if there are three instances of a particular selector in a bag, the mqCountItems call can count the number of instances of that selector and the mqInquire* calls can specify both the selector and the index to inquire those values only. This is useful for attributes that can have a list of values such as some of the exits on channels.

Data conversion

Like PCF messages, the strings contained in an MQAI data bag can be in a variety of coded character sets. Usually, all of the strings in a PCF message are in the same coded character set, that is, the same set as the queue manager.

Each string item in a data bag contains two values; the string itself and the CCSID. The string that is added to the bag is obtained from the *Buffer* parameter of the mqAddString or mqSetString call. The CCSID is obtained from the system item containing a selector of MQIASY_CODED_CHAR_SET_ID. This is known as the *bag CCSID* and can be changed using the mqSetInteger call.

When you inquire the value of a string contained in a data bag, the CCSID is an output parameter from the call.

Table 1 shows the rules applied when converting data bags into messages and vice versa:

Table 1 (Page 1 of 2). CCSID processing

MQAI call	**CCSID**	**Input to call**	**Output to call**
mqBagToBuffer	Bag CCSID (1)	Ignored	Unchanged
	String CCSIDs in bag	Used	Unchanged
	String CCSIDs in buffer	Not applicable	Copied from string CCSIDs in bag
mqBufferToBag	Bag CCSID (1)	Ignored	Unchanged
	String CCSIDs in buffer	Used	Unchanged
	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in buffer
mqPutBag	MQMD CCSID	Used	Unchanged (2)
	Bag CCSID (1)	Ignored	Unchanged
	String CCSIDs in bag	Used	Unchanged
	String CCSIDs in message sent	Not applicable	Copied from string CCSIDs in bag

Table 1 (Page 2 of 2). CCSID processing

MQAI call	CCSID	Input to call	Output to call
mqGetBag	MQMD CCSID	Used for data conversion of message	Set to CCSID of data returned (3)
	Bag CCSID (1)	Ignored	Unchanged
	String CCSIDs in message	Used	Unchanged
	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in message
mqExecute	Request-bag CCSID	Used for MQMD of request message (4)	Unchanged
	Reply-bag CCSID	Used for data conversion of reply message (4)	Set to CCSID of data returned (3)
	String CCSIDs in request bag	Used for request message	Unchanged
	String CCSIDs in reply bag	Not applicable	Copied from string CCSIDs in reply message

Notes:

1. Bag CCSID is the system item with selector MQIASY_CODED_CHAR_SET_ID.
2. MQCCSI_Q_MGR is changed to the actual queue manager CCSID.
3. If data conversion is requested, the CCSID of data returned is the same as the output value. If data conversion is not requested, the CCSID of data returned is the same as the message value. Note that no message is returned if data conversion is requested but fails.
4. If the CCSID is MQCCSI_DEFAULT, the queue manager's CCSID is used.

Use of the message descriptor

The PCF command type is obtained from the system item with selector MQIASY_TYPE. When you create your data bag, the initial value of this item is set depending on the type of bag you create:

Table 2. PCF command type

Type of bag	Initial value of MQIASY_TYPE item
MQCBO_ADMIN_BAG	MQCFT_COMMAND
MQCBO_COMMAND_BAG	MQCFT_COMMAND
MQCBO_*	MQCFT_USER

When the MQAI generates a message descriptor, the values used in the *Format* and *MsgType* parameters depend on the value of the system item with selector MQIASY_TYPE as shown in Table 2.

Table 3. Format and MsgType parameters of the MQMD

PCF command type	**Format**	**MsgType**
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_*	MQFMT_PCF	MQMT_DATAGRAM

Table 3 shows that if you create an administration bag or a command bag, the *Format* of the message descriptor is MQFMT_ADMIN and the *MsgType* is MQMT_REQUEST. This is suitable for a PCF request message sent to the command server when a response is expected back.

Other parameters in the message descriptor take the values shown in Table 4.

Table 4. Message descriptor values

Parameter	**Value**
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	see Table 3
Expiry	30 seconds (note 1)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	depends on the bag CCSID (note 2)
Format	see Table 3
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorelId	MQCI_NONE
BackoutCount	0
ReplyToQ	see note 3
ReplyToQMgr	blank

Notes:

1. This value can be overriden on the the mqExecute call by using the *OptionsBag* parameter. For information about this, see “mqExecute” on page 46.
2. See “Data conversion” on page 102.
3. Name of the user-specified reply queue or MQAI-generated temporary dynamic queue for messages of type MQMT_REQUEST. Blank otherwise.

Appendix A. Return codes

For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed.

The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK

Successful completion.

The call completed fully; all output parameters have been set. The *Reason* parameter always has the value MQRC_NONE in this case.

MQCC_WARNING

Warning (partial completion).

The call completed partially. Some output parameters may have been set in addition to the *CompCode* and *Reason* output parameters. The *Reason* parameter gives additional information about the partial completion.

MQCC_FAILED

Call failed.

The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason may be a fault in the application program, or it may be a result of some situation external to the program, for example the application's authority may have been revoked. The *Reason* parameter gives additional information about the error.

Reason codes

The reason code parameter (*Reason*) is a qualification to the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

The following is a list of reason codes, in alphabetic order, with more detail than is given in the call descriptions.

MQRC_BAG_CONVERSION_ERROR
(2303, X'8FF') Data could not be converted into a bag.

The mqBufferToBag or mqGetBag call was issued, but the data in the buffer or message could not be converted into a bag. This occurs when the data to be converted is not valid PCF.

Corrective action: Check the logic of the application that created the buffer or message to ensure that the buffer or message contains valid PCF.

If the message contains PCF that is not valid, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_BAG_WRONG_TYPE
(2326, X'916') Bag has wrong type for intended use.

The *Bag* parameter specifies the handle of a bag that has the wrong type for the call. The bag must be an administration bag, that is, it must be created with the MQCBO_ADMIN_BAG option specified on the mqCreateBag call.

Corrective action: Specify the MQCBO_ADMIN_BAG option when the bag is created.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

The *Buffer* parameter is not valid for one of the following reasons:

- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The parameter pointer points to storage that cannot be accessed for the entire length specified by *BufferLength*.
- For calls where *Buffer* is an output parameter: the parameter pointer points to read-only storage.

Corrective action: Correct the parameter.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

The *BufferLength* parameter is not valid, or the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Specify a value that is zero or greater. For the mqAddString and mqSetString calls, the special value MQBL_NULL_TERMINATED is also valid.

MQRC_CMD_SERVER_NOT_AVAILABLE

(2322, X'`912`') Command server not available.

The command server that processes administration commands is not available.

Corrective action: Start the command server.

MQRC_CODED_CHAR_SET_ID_ERROR

(2330, X'`91A`') Coded character set identifier parameter not valid.

The *CodedCharSetId* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_COMMAND_TYPE_ERROR

(2300, X'`8FC`') Command type not valid.

The mqExecute call was issued, but the value of the MQIASY_TYPE data item in the administration bag is not MQCFT_COMMAND.

Corrective action: Ensure that the MQIASY_TYPE data item in the administration bag has the value MQCFT_COMMAND.

MQRC_DATA_LENGTH_ERROR

(2010, X'`7DA`') Data length parameter not valid.

The *DataLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ client program that is putting and getting messages, if the application message data is longer than the negotiated maximum message size for the channel.

Corrective action: Correct the parameter.

If the error occurs for an MQ client program, also check that the maximum message size for the channel is big enough to accommodate the message being sent; if it is not big enough, increase the maximum message size for the channel.

MQRC_ENCODING_NOT_SUPPORTED

(2308, X'`904`') Encoding not supported.

The *Encoding* field in the message descriptor MQMD contains a value that is not supported:

- For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call.
- For the mqGetBag call, the field in error resides in:
  - The *MsgDesc* parameter of the call if the MQGMO_CONVERT option was specified.
  - The message descriptor of the message about to be retrieved if MQGMO_CONVERT was *not* specified.

Corrective action: The value must be MQENC_NATIVE.

If the value of the *Encoding* field in the message is not valid, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_⋆ options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_FORMAT_NOT_SUPPORTED
(2317, X'90D') Format not supported.

The *Format* field in the message descriptor MQMD contains a value that is not supported:

- For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call.
- For the mqGetBag call, the field in error resides in the message descriptor of the message about to be retrieved.

Corrective action: The value must be one of the following:

MQFMT_ADMIN
MQFMT_EVENT
MQFMT_PCF

If the value of the *Format* field in the message is none of these values, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_⋆ options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_HBAG_ERROR
(2320, X'910') Bag handle not valid.

A call was issued that has a parameter that is a bag handle, but the handle is not valid. For output parameters, this reason also occurs if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_INCONSISTENT_ITEM_TYPE
(2313, X'909') Data type of item differs from previous occurrence of selector.

The mqAddInteger or mqAddString call was issued to add another occurrence of the specified selector to the bag, but the data type of this occurrence differed from the data type of the first occurrence.

This reason can also occur on the mqBufferToBag and mqGetBag calls, where it indicates that the PCF in the buffer or message contains a selector that occurs more than once but with inconsistent data types.

Corrective action: For the mqAddInteger and mqAddString calls, use the call appropriate to the data type of the first occurrence of that selector in the bag.

For the mqBufferToBag and mqGetBag calls, check the logic of the application that created the buffer or sent the message to ensure that multiple-occurrence selectors occur with only one data type. A message that contains a mixture of data types for a selector cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_INDEX_ERROR
(2314, X`'90A'`) Index not valid.

An index parameter to a call or method has a value that is not valid. The value must be zero or greater. For bag calls, certain MQIND_* values can also be specified:

- For the mqDeleteItem, mqSetInteger and mqSetString calls, MQIND_ALL and MQIND_NONE are valid.
- For the mqInquireBag, mqInquireInteger, mqInquireString, and mqInquireItemInfo calls, MQIND_NONE is valid.

Corrective action: Specify a valid value.

MQRC_INDEX_NOT_PRESENT
(2306, X`'902'`) Index not present.

The specified index is not present:

- For a bag, this means that the bag contains one or more data items that have the selector value specified by the *Selector* parameter, but none of them has the index value specified by the *ItemIndex* parameter. The data item identified by the *Selector* and *ItemIndex* parameters must exist in the bag.
- For a namelist, this means that the index parameter value is too large, and outside the range of valid values.

Corrective action: Specify the index of a data item that does exist in the bag or namelist. Use the mqCountItems call to determine the number of data items with the specified selector that exist in the bag, or the nameCount method to determine the number of names in the namelist.

MQRC_INQUIRY_COMMAND_ERROR
(2324, X`'914'`) Command code is not a recognized inquiry command.

The mqAddInquiry call was used previously to add attribute selectors to the bag, but the command code to be used for the mqBagToBuffer, mqExecute, or mqPutBag call is not recognized. As a result, the correct PCF message cannot be generated.

Corrective action: Remove the mqAddInquiry calls and use instead the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selectors.

MQRC_ITEM_COUNT_ERROR
(2316, X'90C') ItemCount parameter not valid.

The mqTruncateBag call was issued, but the *ItemCount* parameter specifies a value that is not valid. The value is either less than zero, or greater than the number of user-defined data items in the bag.

This reason also occurs on the mqCountItems call if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Specify a valid value. Use the mqCountItems call to determine the number of user-defined data items in the bag.

MQRC_ITEM_TYPE_ERROR
(2327, X'917') ItemType parameter not valid.

The mqInquireItemInfo call was issued, but the *ItemType* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_ITEM_VALUE_ERROR
(2319, X'90F') ItemValue parameter not valid.

The mqInquireBag or mqInquireInteger call was issued, but the *ItemValue* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_MULTIPLE_INSTANCE_ERROR
(2301, X'8FD') Multiple instances of system data item not valid.

The *Selector* parameter specifies a system selector (one of the MQIASY_* values), but the value of the *ItemIndex* parameter is not MQIND_NONE. Only one instance of each system selector can exist in the bag.

Corrective action: Specify MQIND_NONE for the *ItemIndex* parameter.

MQRC_NESTED_BAG_NOT_SUPPORTED
(2325, X'915') Input bag contains one or more nested bags.

A bag which is input to the call contains nested bags. Nested bags are supported only for bags which are output from the call.

Corrective action: Use a different bag as input to the call.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

An MQGET call was issued, but there is no message on the queue satisfying the selection criteria specified in MQMD (the *MsgId* and *CorrelId* fields), and in MQGMO (the *Options* and *MatchOptions* fields). Either the MQGMO_WAIT option was not specified, or the time interval specified by the *WaitInterval* field in MQGMO has expired. This reason is also returned for an MQGET call for browse, when the end of the queue has been reached.

This reason code can also be returned by the mqGetBag and mqExecute calls. mqGetBag is similar to MQGET. For the mqExecute call, the completion code can be either MQCC_WARNING or MQCC_FAILED:

- If the completion code is MQCC_WARNING, some response messages were received during the specified wait interval, but not all. The response bag contains system-generated nested bags for the messages that were received.
- If the completion code is MQCC_FAILED, no response messages were received during the specified wait interval.

Corrective action: If this is an expected condition, no corrective action is required.

If this is an unexpected condition, check whether the message was put on the queue successfully, and whether the options controlling the selection criteria are specified correctly. All of the following can affect the eligibility of a message for return on the MQGET call:

MQGMO_LOGICAL_ORDER
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_COMPLETE_MSG
MQMO_MATCH_MSG_ID
MQMO_MATCH_CORREL_ID
MQMO_MATCH_GROUP_ID
MQMO_MATCH_MSG_SEQ_NUMBER
MQMO_MATCH_OFFSET
MsgId field
CorrelId field

Consider waiting longer for the message.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

The *Options* parameter contains options that are not valid, or a combination of options that is not valid.

Corrective action: Specify valid options. Check the description of the *Options* parameter to determine which options and combinations of options are valid. If multiple options are being set by adding the individual options together, ensure that the same option is not added twice.

MQRC_OUT_SELECTOR_ERROR
(2310, X'906') OutSelector parameter not valid.

The *OutSelector* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_PARAMETER_MISSING
(2321, X'911') Parameter missing.

An administration message requires a parameter that is not present in the administration bag. This reason code occurs only for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options.

Corrective action: Review the description of the administration command being issued, and ensure that all required parameters are present in the bag.

MQRC_SELECTOR_NOT_PRESENT
(2309, X`'905'`) Selector not present in bag.

The *Selector* parameter specifies a selector that does not exist in the bag.

Corrective action: Specify a selector that does exist in the bag.

MQRC_SELECTOR_NOT_SUPPORTED
(2318, X`'90E'`) System selector not supported.

The *Selector* parameter specifies a value that is a system selector (a value that is negative), but the system selector is not one that is supported by the call.

Corrective action: Specify a selector value that is supported.

MQRC_SELECTOR_NOT_UNIQUE
(2305, X`'901'`) Selector occurs more than once in bag.

The *ItemIndex* parameter has the value MQIND_NONE, but the bag contains more than one data item with the selector value specified by the *Selector* parameter. MQIND_NONE requires that the bag contain only one occurrence of the specified selector.

This reason code also occurs on the mqExecute call when the administration bag contains two or more occurrences of a selector for a required parameter that permits only one occurrence.

Corrective action: Check the logic of the application that created the bag. If correct, specify for *ItemIndex* a value that is zero or greater, and add application logic to process all of the occurrences of the selector in the bag.

Review the description of the administration command being issued, and ensure that all required parameters are defined correctly in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
(2304, X`'900'`) Selector not within valid range for call.

The *Selector* parameter has a value that is outside the valid range for the call. If the bag was created with the MQCBO_CHECK_SELECTORS option:

- For the mqAddInteger call, the value must be within the range MQIA_FIRST through MQIA_LAST.
- For the mqAddString call, the value must be within the range MQCA_FIRST through MQCA_LAST.

If the bag was not created with the MQCBO_CHECK_SELECTORS option:

- The value must be zero or greater.

Corrective action: Specify a valid value.

MQRC_SELECTOR_TYPE_ERROR
(2299, X`'8FB'`) Selector has wrong data type.

The *Selector* parameter has the wrong data type; it must be of type `Long`.

Corrective action: Declare the *Selector* parameter as `Long`.

MQRC_SELECTOR_WRONG_TYPE
(2312, X'908') Selector implies a data type not valid for call.

A data item with the specified selector exists in the bag, but has a data type that conflicts with the data type implied by the call being used. For example, the data item might have an integer data type, but the call being used might be mqSetString, which implies a character data type.

This reason code also occurs on the mqBagToBuffer, mqExecute, and mqPutBag calls when mqAddString or mqSetString was used to add the MQIACF_INQUIRY data item to the bag.

Corrective action: For the mqSetInteger and mqSetString calls, specify MQIND_ALL for the *ItemIndex* parameter to delete from the bag all existing occurrences of the specified selector before creating the new occurrence with the required data type.

For the mqInquireBag, mqInquireInteger, and mqInquireString calls, use the mqInquireItemInfo call to determine the data type of the item with the specified selector, and then use the appropriate call to determine the value of the data item.

For the mqBagToBuffer, mqExecute, and mqPutBag calls, ensure that the MQIACF_INQUIRY data item is added to the bag using the mqAddInteger or mqSetInteger calls.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

The call failed because there is insufficient main storage available.

Corrective action: Ensure that active applications are behaving correctly, for example, that they are not looping unexpectedly. If no problems are found, make more main storage available.

MQRC_STRING_ERROR
(2307, X'903') String parameter not valid.

The *String* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_STRING_LENGTH_ERROR
(2323, X'913') StringLength parameter not valid.

The *StringLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

MQRC_STRING_TRUNCATED
(2311, X'907') String truncated (too long for output buffer).

The string returned by the call is too long to fit in the buffer provided. The string has been truncated to fit in the buffer.

Corrective action: If the entire string is required, provide a larger buffer. On the mqInquireString call, the *StringLength* parameter is set by the call to indicate the size of the buffer required to accommodate the string without truncation.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
(2315, X'90B') System bag is read-only and cannot be altered.

A call was issued to add a data item to a bag, modify the value of an existing data item in a bag, or retrieve a message into a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be modified by the application.

Corrective action: Specify the handle of a bag created by the application, or remove the call.

MQRC_SYSTEM_BAG_NOT_DELETABLE
(2328, X'918') System bag is read-only and cannot be deleted.

An mqDeleteBag call was issued to delete a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be deleted by the application.

Corrective action: Specify the handle of a bag created by the application, or remove the call.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
(2302, X'8FE') System data item is read-only and cannot be altered.

A call was issued to modify the value of a system data item in a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be altered by the application.

Corrective action: Specify the selector of a user-defined data item, or remove the call.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
(2329, X'919') System data item is read-only and cannot be deleted.

A call was issued to delete a system data item from a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be deleted by the application.

Corrective action: Specify the selector of a user-defined data item, or remove the call.

The following MQRCCF_* reason code is also mentioned in this book; for details of other MQRCCF_* reason codes associated with PCF commands, refer to the *MQSeries Programmable System Management* book:

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

The MQCFH *Command* field value was not valid.

Corrective action: Specify a valid command identifier.

For a summary of these reason codes in numerical order, see Appendix B, “Constants in C” on page 115.

Appendix B. Constants in C

This appendix specifies the values of all of the named constants that are mentioned in this book. For MQI constants, refer to Appendix B, “Constants for channels and exits” in the *MQSeries Intercommunication* book and Appendix B, “Constants” in the *MQSeries Programmable System Management* guide.

List of constants

The following sections list all of the named constants mentioned in this book, and show their values.

```
/* Create-bag options for mqCreateBag */
#define MQCBO_NONE   (0x00000000L)
#define MQCBO_USER_BAG   (0x00000000L)
#define MQCBO_ADMIN_BAG   (0x00000001L)
#define MQCBO_COMMAND_BAG   (0x00000010L)
#define MQCBO_SYSTEM_BAG   (0x00000020L)
#define MQCBO_LIST_FORM_ALLOWED   (0x00000002L)
#define MQCBO_LIST_FORM_INHIBITED   (0x00000000L)
#define MQCBO_REORDER_AS_REQUIRED   (0x00000004L)
#define MQCBO_DO_NOT_REORDER   (0x00000000L)
#define MQCBO_CHECK_SELECTORS   (0x00000008L)
#define MQCBO_DO_NOT_CHECK_SELECTORS   (0x00000000L)

/* Special selector values */
#define MQSEL_ANY_SELECTOR         (-30001L)
#define MQSEL_ANY_USER_SELECTOR    (-30002L)
#define MQSEL_ANY_SYSTEM_SELECTOR  (-30003L)
#define MQSEL_ALL_SELECTORS        (-30001L)
#define MQSEL_ALL_USER_SELECTORS   (-30002L)
#define MQSEL_ALL_SYSTEM_SELECTORS (-30003L)

/* Integer user selectors */
#define MQIACF_ALL           1009L
#define MQIACF_INQUIRY       1074L
#define MQIACF_WAIT_INTERVAL 1075L

/* Handle user selectors */
#define MQHA_BAG_HANDLE 4001L

/* Limits for handle user selectors */
#define MQHA_FIRST                4001L
#define MQHA_LAST_USED            4001L
#define MQHA_LAST                 6000L

/* Limits for selectors for object attributes */
#define MQOA_FIRST  1L
#define MQOA_LAST   6000L
```

```
/* Integer system selectors */
#define MQIASY_FIRST   (-1L)
#define MQIASY_CODED_CHAR_SET_ID (-1L)
#define MQIASY_TYPE              (-2L)
#define MQIASY_COMMAND           (-3L)
#define MQIASY_MSG_SEQ_NUMBER    (-4L)
#define MQIASY_CONTROL           (-5L)
#define MQIASY_COMP_CODE         (-6L)
#define MQIASY_REASON            (-7L)
#define MQIASY_BAG_OPTIONS         (-8L)
#define MQIASY_LAST_USED         (-8L)
#define MQIASY_LAST         (-2000L)

/* Limits for integer system selectors */
#define MQIASY_FIRST             (-1L)
#define MQIASY_LAST_USED         (-7L)
#define MQIASY_LAST              (-2000L)

/* Special index values */
#define MQIND_NONE (-1L)
#define MQIND_ALL  (-2L)

/* Bag handles */
#define MQHB_UNUSABLE_HBAG (-1L)
#define MQHB_NONE          (-2L)

/* Queue handles */
#define MQHO_NONE (-2L)


/* Values for "BufferLength" parameter on mqAddString/mqSetString */
#define MQBL_NULL_TERMINATED (-1L)

/* Values for "ItemType" parameter on mqInquireItemInfo */
#define MQIT_INTEGER 1L
#define MQIT_STRING  2L
#define MQIT_BAG     3L

/* Values for PCF "Command" field (MQIASY_COMMAND) */
#define MQCMD_NONE 0L
#define MQCMD_INQUIRE_Q_MGR  2L
#define MQCMD_INQUIRE_PROCESS  7L
#define MQCMD_CREATE_Q  11L
#define MQCMD_INQUIRE_Q  13L
#define MQCMD_INQUIRE_Q_NAMES  18L
#define MQCMD_INQUIRE_PROCESS_NAMES  19L
#define MQCMD_INQUIRE_CHANNEL_NAMES  20L
#define MQCMD_INQUIRE_CHANNEL  25L
#define MQCMD_INQUIRE_NAMELIST  36L

/* Values for PCF "Type" field (MQIASY_TYPE) */
#define MQCFT_USER 8L

/* Coded character set identifiers */
#define MQCCSI_DEFAULT  0L

/* Character-attribute selectors */
#define MQCA_Q_NAME   2016L
```

```
/* Integer-attribute selectors */
#define MQIA_Q_TYPE   20L
#define MQIA_SCOPE       45L

/* Queue types */
#define MQQT_LOCAL  1L

/* Queue definition scope */
#define MQSCO_Q_MGR   1L

/* Control options */
#define MQCFC_LAST   1L

/* Formats */
#define MQFMT_EVENT   “MQEVENT  ”
#define MQFMT_PCF     “MQPCF   ”
#define MQFMT_ADMIN   “MQADMIN   ”

/* Reason codes */
#define MQRC_STORAGE_NOT_AVAILABLE      2071L
#define MQRC_COMMAND_TYPE_ERROR        2300L
#define MQRC_BUFFER_ERROR            2004L
#define MQRC_BUFFER_LENGTH_ERROR            2005L
#define MQRC_DATA_LENGTH_ERROR        2010L
#define MQRC_OPTIONS_ERROR  2046L
#define MQRC_MULTIPLE_INSTANCE_ERROR   2301L
#define MQRC_SYSTEM_ITEM_NOT_ALTERABLE 2302L
#define MQRC_BAG_CONVERSION_ERROR      2303L
#define MQRC_SELECTOR_OUT_OF_RANGE     2304L
#define MQRC_SELECTOR_NOT_UNIQUE       2305L
#define MQRC_INDEX_NOT_PRESENT         2306L
#define MQRC_STRING_ERROR              2307L
#define MQRC_ENCODING_NOT_SUPPORTED    2308L
#define MQRC_SELECTOR_NOT_PRESENT      2309L
#define MQRC_OUT_SELECTOR_ERROR        2310L
#define MQRC_DATA_TRUNCATED            2311L
#define MQRC_STRING_TRUNCATED       2311L
#define MQRC_SELECTOR_WRONG_TYPE       2312L
#define MQRC_INCONSISTENT_ITEM_TYPE    2313L
#define MQRC_INDEX_ERROR               2314L
#define MQRC_SYSTEM_BAG_NOT_ALTERABLE  2315L
#define MQRC_ITEM_COUNT_ERROR          2316L
#define MQRC_FORMAT_NOT_SUPPORTED      2317L
#define MQRC_SELECTOR_NOT_SUPPORTED    2318L
#define MQRC_ITEM_VALUE_ERROR          2319L
#define MQRC_HBAG_ERROR                2320L
#define MQRC_PARAMETER_MISSING         2321L
#define MQRC_CMD_SERVER_NOT_AVAILABLE  2322L
#define MQRC_STRING_LENGTH_ERROR       2323L
#define MQRC_INQUIRY_COMMAND_ERROR     2324L
#define MQRC_NESTED_BAG_NOT_SUPPORTED  2325L
#define MQRC_BAG_WRONG_TYPE            2326L
#define MQRC_ITEM_TYPE_ERROR           2327L
#define MQRC_SYSTEM_BAG_NOT_DELETABLE  2328L
#define MQRC_SYSTEM_ITEM_NOT_DELETABLE 2329L
#define MQRC_CODED_CHAR_SET_ID_ERROR    2330L
#define MQRCCF_COMMAND_FAILED  3008L
```

```
/* Function names */
#define mqAddInquiry      MQADDIQ
#define mqAddInteger      MQADDIN
#define mqAddString       MQADDST
#define mqBagToBuffer     MQBG2BF
#define mqBufferToBag     MQBF2BG
#define mqClearBag        MQCLRBG
#define mqCountItems      MQCNTIT
#define mqCreateBag       MQCRTBG
#define mqDeleteBag       MQDELBG
#define mqDeleteItem      MQDELIT
#define mqExecute         MQEXEC
#define mqGetBag          MQGETBG
#define mqInquireBag      MQINQBG
#define mqInquireInteger  MQINQIN
#define mqInquireItemInfo MQINQII
#define mqInquireString   MQINQST
#define mqPad             MQPAD
#define mqPutBag          MQPUTBG
#define mqSetInteger      MQSETIN
#define mqSetString       MQSETST
#define mqTrim            MQTRIM
#define mqTruncateBag     MQTRNBG
```

Elementary datatypes in C

```
typedef MQLONG MQHBAG;
typedef MQHBAG MQPOINTER PMQHBAG;
```

Appendix C. Header files

MQSeries provides C and Visual Basic header files to help you write your MQAI applications:

Table 5. Header files

C	Visual Basic	Description
cmqbc.h	CMQBB.BAS	Contains prototypes, datatypes (MQHBAG), and named constants for the MQAI.
cmqcfc.h	CMQCFB.BAS	Contains elementary datatypes and named constants for events and PCF commands.
cmqc.h	CMQB.BAS	Contains prototypes, data types, and named constants for the main MQI.

Appendix D. Selectors

Items in bags are identified by a *selector* that acts as an identifier for the item. There are two types of selector, *user selector* and *system selector*.

User selectors

User selectors have values that are zero or positive. For the administration of MQSeries objects, valid user selectors are already defined by the following constants:

- MQCA_* and MQIA_* (object attributes)
- MQCACF_* and MQIACF_* (items relating specifically to PCF)
- MQCACH_* and MQIACH_* (channel attributes)

For user messages, the meaning of a user selector is defined by the application.

The following additional user selectors are introduced by the MQAI:

MQIACF_INQUIRY
Identifies an MQSeries object attribute to be returned by an Inquire command.

MQHA_BAG_HANDLE
Identifies a bag handle residing within another bag.

MQHA_FIRST
Lower limit for handle selectors.

MQHA_LAST
Upper limit for handle selectors.

MQHA_LAST_USED
Upper limit for last handle selector allocated.

MQCA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports character type and represents the default value used if the *Selector* parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

MQIA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports integer type and represents the default value used if the *Selector* parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

System selectors

System selectors have negative values. The following system selectors are included in the bag when it is created:

MQIASY_BAG_OPTIONS
Bag-creation options. A summation of the options used to create the bag. This selector cannot be changed by the user.

MQIASY_CODED_CHAR_SET_ID
Character set identifier for the character data items in the bag. The initial value is the queue-manager's character set.

The value in the bag is used on entry to the mqExecute call and set on exit from the mqExecute call. This also applies when character strings are added to or modified in the bag.

MQIASY_COMMAND
PCF command identifier. Valid values are the MQCMD_* constants. For user messages, the value MQCMD_NONE should be used. The initial value is MQCMD_NONE.

The value in the bag is used on entry to the mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_COMP_CODE
Completion code. Valid values are the MQCC_* constants. The initial value is MQCC_OK.

The value in the bag is used on entry to the mqExecute, mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_CONTROL
PCF control options. Valid values are the MQCFC_* constants. The initial value is MQCFC_LAST.

The value in the bag is used on entry to the mqExecute, mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_MSG_SEQ_NUMBER
PCF message sequence number. Valid values are 1 or greater. The initial value is 1.

The value in the bag is used on entry to the mqExecute, mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_REASON
Reason code. Valid values are the MQRC_* constants. The initial value is MQRC_NONE.

The value in the bag is used on entry to the mqExecute, mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_TYPE
PCF command type. Valid values are the MQCFT_* constants. For user messages, the value MQCFT_USER should be used. The initial value is MQCFT_USER for bags created as user bags and MQCFT_COMMAND for bags created as administration or command bags.

The value in the bag is used on entry to the mqExecute, mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

Appendix E. Notices

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

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

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

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

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

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

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
CICS
IBM
MQ
OS/390
AS/400
FFST
IBMLink
MQSeries
RACF
BookManager
First Failure Support Technology
IMS
OS/2
VSE/ESA

Lotus is a trademark of Lotus Development Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

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

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

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

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with MQSeries for OS/390. A complete list of MQSeries for OS/390 abend reason codes and their explanations is contained in the *MQSeries for OS/390 Messages and Codes* manual.

active log. See *recovery log*.

adapter. An interface between MQSeries for OS/390 and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

address space. The area of virtual storage available for a particular job.

address space identifier (ASID). A unique, system-assigned identifier for an address space.

administration bag. In the MQAI, a type of data bag that is created for administering MQSeries by implying that it can change the order of data items, create lists, and check selectors within a message.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

alert monitor. In MQSeries for OS/390, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to MQSeries for OS/390.

alias queue object. An MQSeries object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

allied address space. See *ally*.

ally. An OS/390 address space that is connected to MQSeries for OS/390.

alternate user security. A security feature in which the authority of one user ID can be used by another user ID; for example, to open an MQSeries object.

APAR. Authorized program analysis report.

application environment. The software facilities that are accessible by an application program. On the OS/390 platform, CICS® and IMS™ are examples of application environments.

application log. In Windows NT, a log that records significant application events.

application queue. A queue used by an application.

archive log. See *recovery log*.

ASID. Address space identifier.

asynchronous messaging. A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute. One of a set of properties that defines the characteristics of an MQSeries object.

authorization checks. Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

authorization file. In MQSeries on UNIX systems, a file that provides security definitions for an object, a class of objects, or all classes of objects.

authorization service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

backout. An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

bag. See *data bag*.

basic mapping support (BMS). An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS. Basic mapping support.

bootstrap data set (BSDS). A VSAM data set that contains:

- An inventory of all active and archived log data sets known to MQSeries for OS/390
- A wrap-around inventory of all recent MQSeries for OS/390 activity

The BSDS is required if the MQSeries for OS/390 subsystem has to be restarted.

browse. In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor. In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

BSDS. Bootstrap data set.

buffer pool. An area of main storage used for MQSeries for OS/390 queues, messages, and object definitions. See also *page set*.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel control function (CCF). In MQSeries, a program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. (1) A time when significant information is written on the log. Contrast with *syncpoint*. (2) In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

CI. Control interval.

circular logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping all restart data in a ring of log files. Logging fills the first file in the ring and then moves on to the next, until all the files are full. At this point, logging goes back to the first file in the ring and starts again, if the space has been freed or is no longer needed. Circular logging is used during restart recovery, using the log to roll back transactions that were in progress when the system stopped. Contrast with *linear logging*.

CL. Control Language.

client. A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQSeries client*.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client connection channel type. The type of MQI channel definition associated with an MQSeries client. See also *server connection channel type*.

cluster. A network of queue managers that are logically associated in some way.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an administration instruction that can be carried out by the queue manager.

command bag. In the MQAI, a type of bag that is created for administering MQSeries objects, but cannot change the order of data items nor create lists within a message.

command prefix (CPF). In MQSeries for OS/390, a character string that identifies the queue manager to which MQSeries for OS/390 commands are directed, and from which MQSeries for OS/390 operator messages are received.

command processor. The MQSeries component that processes commands.

command server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code. A return code indicating how an MQI call has ended.

configuration file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file that contains configuration information related to, for example, logs, communications, or installable services. Synonymous with *.ini file*. See also *stanza*.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN call, or automatically by the MQOPEN call.

connection handle. The identifier or token by which a program accesses the queue manager to which it is connected.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control command. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. Such a command requires only that the MQSeries product be installed; it does not require a special utility or program to run it.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

Control Language (CL). In MQSeries for AS/400, a language that can be used to issue commands, either at the command line or by writing a CL program.

controlled shutdown. See *quiesced shutdown*.

CPF. Command prefix.

D

DAE. Dump analysis and elimination.

data bag. In the MQAI, a bag that allows you to handle properties (or parameters) of objects.

data item. In the MQAI, an item contained within a data bag. This can be an integer item or a character-string item, and a user item or a system item.

data conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCE. Distributed Computing Environment.

DCI. Data conversion interface.

dead-letter queue (DLQ). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

default object. A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

deferred connection. A pending event that is activated when a CICS subsystem tries to connect to MQSeries for OS/390 before MQSeries for OS/390 has been started.

distributed application. In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Distributed Computing Environment (DCE). Middleware that provides some basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dual logging. A method of recording MQSeries for OS/390 activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. Contrast with *single logging*.

dual mode. See *dual logging*.

dump analysis and elimination (DAE). An OS/390 service that enables an installation to suppress SVC dumps and ABEND SYSUDUMP dumps that are not needed because they duplicate previously written dumps.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

environment. See *application environment*.

ESM. External security manager.

ESTAE. Extended specify task abnormal exit.

event. See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

event data. In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

event header. In an event message, the part of the message data that identifies the event type of the reason code for the event.

event log. See *application log*.

event message. Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

event queue. The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

Event Viewer. A tool provided by Windows NT to examine and manage log files.

extended specify task abnormal exit (ESTAE). An OS/390 macro that provides recovery capability and gives control to the specified exit routine for processing, diagnosing an abend, or specifying a retry address.

external security manager (ESM). A security product that is invoked by the OS/390 System Authorization Facility. RACF® is an example of an ESM.

F

FFST™. First Failure Support Technology™.

FIFO. First-in-first-out.

First Failure Support Technology (FFST). Used by MQSeries on UNIX systems, MQSeries for OS/2 Warp, MQSeries for Windows NT, and MQSeries for AS/400 to detect and report software problems.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

forced shutdown. A type of shutdown of the CICS adapter where the adapter immediately disconnects from MQSeries for OS/390, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

Framework. In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:

- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

FRR. Functional recovery routine.

functional recovery routine (FRR). An OS/390 recovery/termination manager facility that enables a recovery routine to gain control in the event of a program interrupt.

G

GCPC. Generalized command preprocessor.

generalized command preprocessor (GCPC). An MQSeries for OS/390 component that processes MQSeries commands and runs them.

Generalized Trace Facility (GTF). An OS/390 service program that records significant system events, such as supervisor calls and start I/O operations, for the purpose of problem determination.

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

global trace. An MQSeries for OS/390 trace option where the trace data comes from the entire MQSeries for OS/390 subsystem.

GTF. Generalized Trace Facility.

H

handle. See *connection handle* and *object handle*.

I

immediate shutdown. In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

index. In the MQAI, a means of referencing data items.

in-doubt unit of recovery. In MQSeries, the status of a unit of recovery for which a syncpoint has been requested but not yet confirmed.

.ini file. See *configuration file*.

initialization input data sets. Data sets used by MQSeries for OS/390 when it starts up.

initiation queue. A local queue on which the queue manager puts trigger messages.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

insertion order. In the MQAI, the order that data items are placed into a data bag.

installable services. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

instrumentation event. A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

Interactive Problem Control System (IPCS). A component of OS/390 that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. It is used for writing application programs, and provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

IPCS. Interactive Problem Control System.

ISPF. Interactive System Productivity Facility.

L

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

listener. In MQSeries distributed queuing, a program that monitors for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

locale. On UNIX systems, a subset of a user's environment that defines conventions for a specific culture (such as time, numeric, or monetary formatting and character classification, collation, or conversion). The queue manager CCSID is derived from the locale of the user ID that created the queue manager.

local queue. A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

log control file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file in which all significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, MQSeries allocates secondary log files.

logical unit of work (LUW). See *unit of work*.

M

machine check interrupt. An interruption that occurs as a result of an equipment malfunction or error. A machine check interrupt can be either hardware recoverable, software recoverable, or nonrecoverable.

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. (1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. (2) In system programming, information intended for the terminal operator or system administrator.

message channel. In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

message channel interface (MCI). The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

message descriptor. Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

message priority. In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue. Synonym for *queue*.

message queue interface (MQI). The programming interface provided by the MQSeries queue managers. This programming interface allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

messaging. See *synchronous messaging* and *asynchronous messaging*.

model queue object. A set of queue attributes that act as a template when a program creates a dynamic queue.

MQAI. MQSeries Administration Interface.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries Administration Interface (MQAI). A programming interface to MQSeries.

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQSeries commands (MQSC). Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects. Contrast with *programmable command format (PCF)*.

N

namelist. An MQSeries object that contains a list of names, for example, queue names.

name service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the facility that determines which queue manager owns a specified queue.

name service interface (NSI). The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

name transformation. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, an internal process that changes a queue manager name so that it is unique and valid for the system being used. Externally, the queue manager name remains unchanged.

nested bag. In the MQAI, a system bag that is inserted into another data bag

nesting. In the MQAI, a means of grouping information returned from MQSeries.

New Technology File System (NTFS). A Windows NT recoverable file system that provides security for files.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

NTFS. New Technology File System.

null character. The character that is represented by `X'00'`.

O

OAM. Object authority manager.

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist, or a storage class (OS/390 only).

object authority manager (OAM). In MQSeries on UNIX systems and MQSeries for Windows NT, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

object descriptor. A data structure that identifies a particular MQSeries object. Included in the descriptor are the name of the object and the object type.

object handle. The identifier or token by which a program accesses the MQSeries object with which it is working.

off-loading. In MQSeries for OS/390, an automatic process whereby a queue manager's active log is transferred to its archive log.

output log-buffer. In MQSeries for OS/390, a buffer that holds recovery log records before they are written to the archive log.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

page set. A VSAM data set used when MQSeries for OS/390 moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

PCF. Programmable command format.

PCF command. See *programmable command format*.

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation. In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event. A category of event indicating that a limit condition has occurred.

performance trace. An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue. A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

persistent message. A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

ping. In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

point of recovery. In MQSeries for OS/390, the term used to describe a set of backup copies of MQSeries for OS/390 page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

preemptive shutdown. In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

principal. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a term used for a user identifier. Used by the object authority manager for checking authorizations to system resources.

process definition object. An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

programmable command format (PCF). A type of MQSeries message used by:

- User administration applications, to put PCF commands onto the system command input queue of a specified queue manager
- User administration applications, to get the results of a PCF command from a specified queue manager
- A queue manager, as a notification that an event has occurred

Contrast with *MQSC*.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF. Program temporary fix.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager. (1) A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. (2) An MQSeries object that defines the attributes of a particular queue manager.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queuing. See *message queuing*.

quiesced shutdown. (1) In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. (2) A type of shutdown of the CICS adapter where the adapter disconnects from MQSeries, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing. In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

recovery log. In MQSeries for OS/390, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for OS/390 writes each record to a data set called the *active log*. When the active log is full, its contents are off-loaded to a DASD or tape data set called the *archive log*. Synonymous with *log*.

recovery termination manager (RTM). A program that handles all normal and abnormal termination of tasks by passing control to a recovery routine associated with the terminating function.

Registry. In Windows NT, a secure database that provides a single source for system and application configuration data.

Registry Editor. In Windows NT, the program item that allows the user to edit the Registry.

Registry Hive. In Windows NT, the structure of the data stored in the Registry.

relative byte address (RBA). The displacement in bytes of a stored record or control interval from the beginning of the storage space allocated to the data set to which it belongs.

remote queue. A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To a program, a queue manager that is not the one to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message. A type of message used for replies to request messages. Contrast with *request message* and *report message*.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. Contrast with *reply message* and *request message*.

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

request message. A type of message used to request a reply from another program. Contrast with *reply message* and *report message*.

RESLEVEL. In MQSeries for OS/390, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for OS/390.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource. Any facility of the computing system or operating system required by a job or task. In MQSeries for OS/390, examples of resources are buffer pools, page sets, log data sets, queues, and messages.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

responder. In distributed queuing, a program that replies to network connection requests from another system.

resynch. In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

RTM. Recovery termination manager.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

SAF. System Authorization Facility.

SDWA. System diagnostic work area.

security enabling interface (SEI). The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

SEI. Security enabling interface.

selector. Used to identify a data item. In the MQAI there are two types of selector: a user selector and a system selector.

sender channel. In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server. (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel. In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

server connection channel type. The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

service interval. A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event. An event related to the service interval.

session ID. In MQSeries for OS/390, the CICS-unique identifier that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

signaling. In MQSeries for OS/390 and MQSeries for Windows 2.1, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

single logging. A method of recording MQSeries for OS/390 activity where each change is recorded on one data set only. Contrast with *dual logging*.

single-phase backout. A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

SIT. System initialization table.

stanza. A group of lines in a configuration file that assigns a value to a parameter modifying the behavior of a queue manager, client, or channel. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a configuration (.ini) file may contain a number of stanzas.

storage class. In MQSeries for OS/390, a storage class defines the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

subsystem. In OS/390, a group of modules that provides function that is dependent on OS/390. For example, MQSeries for OS/390 is an OS/390 subsystem.

supervisor call (SVC). An OS/390 instruction that interrupts a running program and passes control to the supervisor so that it can perform the specific service indicated by the instruction.

SVC. Supervisor call.

switch profile. In MQSeries for OS/390, a RACF profile used when MQSeries starts up or when a refresh security command is issued. Each switch profile that MQSeries detects turns off checking for the specified resource.

symptom string. Diagnostic information displayed in a structured format designed for searching the IBM software support database.

synchronous messaging. A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

System Authorization Facility (SAF). An OS/390 facility through which MQSeries for OS/390 communicates with an external security manager such as RACF.

system bag. A type of data bag that is created by the MQAI.

system.command.input queue. A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

system control commands. Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

system diagnostic work area (SDWA). Data recorded in a SYS1.LOGREC entry, which describes a program or hardware error.

system initialization table (SIT). A table containing parameters used by CICS on start up.

system item. A type of data item that is created by the MQAI.

system selector. In the MQAI, used to identify a system item. A system selector is included in the data bag when it is created.

SYS1.LOGREC. A service aid containing information about program and hardware errors.

T

TACL. Tandem Advanced Command Language.

target library high-level qualifier (thlqual). High-level qualifier for OS/390 target data set names.

task control block (TCB). An OS/390 control block used to communicate information about tasks within an address space that are connected to an OS/390 subsystem such as MQSeries for OS/390 or CICS.

task switching. The overlapping of I/O operations and processing between several tasks. In MQSeries for OS/390, the task switcher optimizes performance by allowing some MQI calls to be executed under subtasks rather than under the main CICS TCB.

TCB. Task control block.

temporary dynamic queue. A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

termination notification. A pending event that is activated when a CICS subsystem successfully connects to MQSeries for OS/390.

thlqual. Target library high-level qualifier.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

trace. In MQSeries, a facility for recording MQSeries activity. The destinations for trace entries can include GTF and the system management facility (SMF). See also *global trace* and *performance trace*.

tranid. See *transaction identifier*.

transaction identifier. In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

triggering. In MQSeries, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI). The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

UIS. User identifier service.

undelivered-message queue. See *dead-letter queue*.

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

unit of recovery. A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

user bag. In the MQAI, a type of data bag that is created by the user.

user identifier service (UIS). In MQSeries for OS/2 Warp, the facility that allows MQI applications to associate a user ID, other than the default user ID, with MQSeries messages.

user item. In the MQAI, a type of data item that is created by the user.

user selector. In the MQAI, used to identify a user item. For the administration of MQSeries objects, valid user selectors are already defined.

utility. In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

V

value. Value of a data item. This can be an integer, a string, or a handle of another bag.

Index

Sending your comments to IBM

MQSeries®

Administration Interface
Programming Guide and Reference

SC34-5390-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

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

Readers' Comments

MQSeries®

Administration Interface
Programming Guide and Reference

SC34-5390-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

MQSeries®

MQSeries Administration Interface Programming Guide and Reference SC34-5390-00

You can send your comments POST FREE on this form from any one of these countries:

Australia
Belgium
Bermuda
Cyprus
Denmark
Finland
France
Germany
Greece
Hong Kong
Iceland
Israel
Italy
Luxembourg
Monaco
Netherlands
New Zealand
Norway
Portugal
Republic of Ireland
Singapore
Spain
Sweden
Switzerland
United Arab Emirates
United States of America

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name ______________________
Company or Organization ______________________
Address ______________________
____
EMAIL ______________________
Telephone ______________________

1 Cut along this line

4 Fasten here with adhesive tape

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5390-00