10.0

Administering IBM MQ

Note

Before using this information and the product it supports, read the information in “Notices” on page 569.

This edition applies to version 9 release 4 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2007, 2026.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Administering....7
- Ways of administering IBM MQ queue managers and associated resources.... 8
- Administering IBM MQ for Multiplatforms using control commands....10
- Administering IBM MQ using MQSC commands....12
  - MQSC command syntax.... 12
  - MQSC input file syntax.... 15
  - Running MQSC commands interactively under **runmqsc**....17
  - Running MQSC commands from text files under **runmqsc**.... 22
  - Automatic configuration from an MQSC script at startup.... 23
- Automating IBM MQ administration using PCF commands.... 24
  - Introduction to IBM MQ Programmable Command Formats....25
  - Using the MQAI to simplify the use of PCFs.... 37
- Administration using the REST API.... 72
  - Getting started with the administrative REST API.... 73
  - Remote administration using the REST API.... 78
  - REST API time stamps.... 82
  - REST API error handling....82
  - REST API discovery.... 85
  - REST API national language support....86
  - REST API versions.... 88
- Administration using the IBM MQ Console.... 89
  - Getting started with the IBM MQ Console.... 89
  - Quick tour of the IBM MQ Console....91
  - IBM MQ Console settings....125
- Administration using the IBM MQ Explorer.... 125
  - What you can do with the IBM MQ Explorer....125
  - Setting up the IBM MQ Explorer.... 127
- Using the IBM MQ Taskbar application (Windows only)....133
  - The IBM MQ alert monitor application (Windows only).... 133
- Working with local IBM MQ objects.... 133
  - Working with queue managers.... 134
  - Stopping MQI channels....144
  - Working with local queues....144
  - Working with remote queues....154
  - Working with alias queues.... 157
  - Working with model queues.... 159
  - Working with dead-letter queues.... 160
  - Working with administrative topics.... 178
  - Working with subscriptions....182
  - Working with services.... 186
  - Managing objects for triggering.... 193
  - **CAPEXPRY**: Limit message expiry time....195
  - Using the **dmpmqmsg** utility between two systems.... 197
- Working with remote IBM MQ objects.... 200
  - Configuring queue managers for remote administration....201
  - Managing the command server for remote administration.... 205
  - Issuing MQSC commands on a remote queue manager....206
  - Data conversion between coded character sets 207
- Administering Managed File Transfer.... 211
  - Starting an MFT agent.... 213
  - Listing MFT agents.... 218
  - Stopping an MFT agent.... 218

Starting a new file transfer....220
Creating a scheduled file transfer....223
Working with pending file transfers....224
Triggering a file transfer....225
Monitoring file transfers that are in progress....226
Viewing the status of file transfers in the Transfer Log....228
Monitoring MFT resources....230
Working with file transfer templates....260
Transferring data from files to messages....263
Transferring data from messages to files....278
The protocol bridge....288
The Connect:Direct bridge....311
Working with MFT from IBM Integration Bus....326
MFT recovery and restart....327
Setting a timeout for recovery of stalled transfers....327
Administering IBM MQ Telemetry....333
Configuring a queue manager for telemetry on Linux and AIX....333
Configuring a queue manager for telemetry on Windows....335
Configuring distributed queuing to send messages to MQTT clients....337
MQTT client identification, authorization, and authentication....339
IBM MQ Telemetry channel authentication using TLS....345
Publication privacy on telemetry channels....350
TLS configuration of MQTT Java clients and telemetry channels....350
IBM MQ Telemetry channel JAAS configuration....356
Administering an AMQP client....358
AMQP Service does not start automatically on queue manager startup....358
Viewing IBM MQ objects in use by AMQP clients....359
AMQP client identification, authorization, and authentication....360
Publication privacy on channels....362
Configuring AMQP clients with TLS....363
Disconnecting AMQP clients from the queue manager....365
Administering multicast....365
Getting started with multicast....365
IBM MQ Multicast topic topology....366
Controlling the size of multicast messages....367
Enabling data conversion for Multicast messaging....369
Multicast application monitoring....369
Multicast message reliability....370
Advanced multicast tasks....371
Administering IBM MQ for IBM i....374
Managing IBM MQ for IBM i using CL commands....374
Alternative ways of administering IBM MQ for IBM i....387
Work management for IBM i....393
Availability, backup, recovery, and restart on IBM i....399
Quiescing IBM MQ for IBM i....440
Administering IBM MQ for z/OS....444
Issuing queue manager commands on z/OS....444
Using the operations and control panels on z/OS....458
Using the IBM MQ for z/OS utilities....467
Using the Command Facility on z/OS....469
Working with IBM MQ objects on z/OS....470
Implementing the system using multiple cluster transmission queues....473
Writing programs to administer IBM MQ for z/OS....475
Managing IBM MQ resources on z/OS....487
Recovery and restart on z/OS....525
IBM MQ and IMS....546
Operating Advanced Message Security on z/OS....558
Administering IBM MQ Internet Pass-Thru....559

Starting and stopping MQIPT....559
Administering MQIPT by using the command line....561
Making backups....567
Performance tuning....567

Notices....569
Programming interface information....570
Trademarks....570

Administering IBM MQ

To administer your IBM MQ queue managers and associated resources, choose your preferred method from a set of tasks you can use to activate and manage those resources.

About this task

You can administer IBM MQ objects locally or remotely:

Local administration

Local administration means carrying out administration tasks on any queue managers you have defined on your local system. You can access other systems, for example through the TCP/IP terminal emulation program **telnet**, and carry out administration there. In IBM MQ, you can consider this as local administration because no channels are involved, that is, the communication is managed by the operating system.

For more information, see "Working with local IBM MQ objects" on page 133.

Remote administration

IBM MQ supports administration from a single point of contact through remote administration. Remote administration allows you to issue commands from your local system that are processed on another system and applies also to the IBM MQ Explorer. For example, you can issue a remote command to change a queue definition on a remote queue manager. You do not have to log on to that system, although you do need to have the appropriate channels defined. The queue manager and command server on the target system must be running.

Some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log on to the remote system and issue the commands from there or create a process that can issue the commands for you. This restriction applies also to the IBM MQ Explorer.

For more information, see "Working with remote IBM MQ objects" on page 200.

There are a number of different methods that you can use to create and administer your queue managers and their related resources in IBM MQ. These methods include command-line interfaces, graphical user interfaces, and an administration API.

There are different sets of commands that you can use to administer IBM MQ depending on your platform:

- "IBM MQ control commands" on page 8
- "IBM MQ Script (MQSC) commands" on page 8
- "Programmable Command Formats (PCFs)" on page 8
- The administrative REST API
- IBM i "IBM i Control Language (CL)" on page 9

There are also the other following options for creating and managing IBM MQ objects:

- Windows Linux "The IBM MQ Explorer" on page 9
- "The IBM MQ Console" on page 9
- Windows "The Microsoft Cluster Service (MSCS)" on page 10

z/OS For information about the administration interfaces and options on IBM MQ for z/OS®, see "Administering IBM MQ for z/OS" on page 444.

You can automate some administration and monitoring tasks for both local and remote queue managers by using PCF commands. These commands can also be simplified by using the IBM MQ Administration Interface (MQAI) on some platforms. For more information about automating administration tasks, see "Automating IBM MQ administration using PCF commands" on page 24.

© Copyright IBM Corp. 2007, 2026

Related concepts
IBM MQ technical overview
Related tasks
Planning
Configuring
Related reference
Command sets comparison

Ways of administering IBM MQ queue managers and associated resources

You can administer IBM MQ queue managers and associated resources by using IBM MQ control commands, IBM MQ Script Commands (MQSC), Programmable Command Formats (PCFs), the administrative REST API, the IBM MQ Console, and the IBM MQ Explorer. For IBM i you can also use IBM i Control Language, and for Windows you can also use the Microsoft Cluster Service (MSCS).

IBM MQ control commands

Multi

The control commands provide a way to perform a number of IBM MQ administration tasks. For AIX®, Linux®, and Windows, you issue these commands at the system command line. For IBM i, you issue these commands within a Qshell. See “Administering IBM MQ for Multiplatforms using control commands” on page 10.

IBM MQ Script (MQSC) commands

Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

ALW On AIX, Linux, and Windows, you open a **runmqsc** command prompt, then issue MQSC commands to a local or remote queue manager from that prompt. You can do this interactively, or you can run a sequence of commands from an ASCII text file. For more information, see “Running MQSC commands interactively under runmqsc” on page 17 and “Running MQSC commands from text files under runmqsc” on page 22.

IBM i On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. For more information, see “Administering using MQSC commands on IBM i” on page 388.

z/OS On z/OS, MQSC commands can be issued from a number of sources, depending on the command. For more information, see “Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS” on page 444.

Programmable Command Formats (PCFs)

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. You can use PCF commands in a systems management application program for administration of IBM MQ objects: authentication information objects, channels, channel listeners, namelists, process definitions, queue managers, queues, services, and storage classes. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local, or remote, using the local queue manager.

For more information about PCFs, see “Introduction to IBM MQ Programmable Command Formats” on page 25.

For definition of PCFs and structures for the commands and responses, see Programmable command formats reference.

The administrative REST API

The administrative REST API provides a RESTful interface that you can use to administer IBM MQ. When you use the administrative REST API, you invoke an HTTP method on a URL that represents an IBM MQ object. For example, you can request information about IBM MQ installations by using the HTTP method GET on the following URL:

```
https://localhost:9443/ibmmq/rest/v1/admin/installation
```

You can use the administrative REST API with the HTTP/REST implementation of a programming language, or by using tools such as cURL, or a REST client browser add-on.

For more information, see The administrative REST API

The IBM MQ Console

You can use the IBM MQ Console to administer IBM MQ from a web browser.

For more information, see “Administration using the IBM MQ Console” on page 89.

The IBM MQ Explorer

Windows Linux

Using the IBM MQ Explorer, you can perform the following actions:

- Define and control various resources, such as queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and clusters.
- Start or stop a local queue manager and its associated processes.
- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.
- Check to see which applications, users, or channels have a particular queue open, from the queue status.

On Windows and Linux for x86-64 systems, you can start IBM MQ Explorer by using the system menu, or the `MQExplorer` executable file.

Linux On Linux, to start the IBM MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

For more information, see “Administration using the IBM MQ Explorer” on page 125.

You can use IBM MQ Explorer to administer remote queue managers on other platforms including z/OS.

From IBM MQ 9.3.0, IBM MQ Explorer has been removed from the IBM MQ installation package. It remains available as a separate download, and can be installed from the stand-alone IBM MQ Explorer download available from Fix Central. For more information, see Installing and uninstalling IBM MQ Explorer as a stand-alone application on Linux and Windows.

IBM i Control Language (CL)

IBM i

This is the preferred way to issue administration commands to IBM MQ for IBM i. The commands can be issued either at the command line or by writing a CL program. These commands perform similar functions to PCF commands, but the format is different. CL commands are designed exclusively for servers and CL responses are human-readable, whereas PCF commands are platform independent and both command and response formats are intended for program use.

For full details of the IBM i Control Language (CL), see “Managing IBM MQ for IBM i using CL commands” on page 374 and IBM MQ for IBM i CL commands.

The Microsoft Cluster Service (MSCS)

Windows

Microsoft Cluster Service (MSCS) enables you to connect servers into a *cluster*, giving higher availability of data and applications, and making it easier to manage the system. MSCS can automatically detect and recover from server or application failures.

It is important not to confuse clusters in the MSCS sense with IBM MQ clusters. The distinction is as follows:

IBM MQ clusters
: These are groups of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues to be shared among them for load balancing and redundancy.

MSCS clusters
: These are groups of computers, which are connected together and configured in such a way that, if one fails, MSCS performs a *failover*, transferring the state data of applications from the failing computer to another computer in the cluster and re-initiating their operation there.

Supporting the Microsoft Cluster Service (MSCS) provides detailed information about how to configure your IBM MQ for Windows system to use MSCS.

Related tasks

“Administering IBM MQ using MQSC commands” on page 12
You can use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects. MQSC commands are available on all platforms.

Related reference

Administration reference

Multi Administering IBM MQ for Multiplatforms using control commands

The control commands provide a way to perform a number of IBM MQ administration tasks. For AIX, Linux, and Windows, you issue these commands at the system command line. For IBM i, you issue these commands within a Qshell.

Before you begin

When using control commands that operate on a queue manager, you must use the command from the installation associated with the queue manager you are working with.

When using control commands that operate on a queue manager configured to use connection authentication with CHCKLOCL(REQUIRED), and a failure to connect is observed, either

- Supply a user ID and password if the control command allows this.
- Use MQSC equivalents of the control commands where those exist.
- Start the queue manager using the -ns option, while control commands that cannot connect need to be run.

Note: Different platforms can accept command arguments entered in a different order. Particularly, this means that commands that work on Linux might not work on other platforms. For this reason, you should always enter arguments as specified in the syntax diagrams.

For a full list of the control commands, see the IBM MQ control commands reference.

Procedure

- Linux AIX

  Use control commands on AIX and Linux systems.

  In IBM MQ for AIX or Linux systems, you enter control commands in a shell window.

  If you want to issue control commands, your user ID must be a member of the mqm group for most control commands. For more information about this, see Authority to administer IBM MQ on AIX, Linux, and Windows. In addition, note the environment-specific information. for the platform, or platforms, your enterprise uses.

  In UNIX and Linux environments, control commands, including the command name itself, the flags, and any arguments, are case-sensitive. For example, in the command:

  ```
  crtmqm -u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
  ```

  - The command name must be `crtmqm`, not `CRTMQM`.
  - The flag must be `-u`, not `-U`.
  - The dead-letter queue is called `SYSTEM.DEAD.LETTER.QUEUE`.
  - The argument is specified as `jupiter.queue.manager`, which is different from `JUPITER.queue.manager`.

  Take care to type the commands exactly as you see them in the examples.

- Windows

  Use control commands on Windows systems.

  In IBM MQ for Windows, you enter control commands at a command prompt.

  If you want to issue control commands, your user ID must be a member of the mqm group for most control commands. For more information about this, see Authority to administer IBM MQ on AIX, Linux, and Windows. In addition, note the environment-specific information. for the platform, or platforms, your enterprise uses.

  The control commands and their flags are not case-sensitive, but arguments to those commands, such as queue names and queue manager names, are case-sensitive.

  For example, in the command:

  ```
  crtmqm /u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
  ```

  - The command name can be entered in uppercase or lowercase, or a mixture of the two. These are all valid: `crtmqm`, `CRTMQM`, and `CRTmqm`.
  - The flag can be entered as `-u`, `-U`, `/u`, or `/U`.
  - `SYSTEM.DEAD.LETTER.QUEUE` and `jupiter.queue.manager` must be entered exactly as shown.

- IBM i

  Use control commands on IBM i systems.

  On IBM MQ for IBM i, you run the control commands from a Qshell environment. To use the Qshell, you type STRQSH at the IBM i command line. You can exit and return to the command line at any time by pressing F3.

  A small number of control commands are not supported on IBM i. For example, multi-installation commands are not supported because you cannot have more than one copy of IBM MQ on an IBM i system. Commands that are not supported on IBM i are flagged ALW in the IBM MQ control commands reference.

Related reference

IBM MQ control commands reference

Administering IBM MQ using MQSC commands

You can use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects. MQSC commands are available on all platforms.

About this task

Available MQSC commands are detailed in the MQSC commands reference.

The way that you issue MQSC commands depends on your platform:

- ALW On AIX, Linux, and Windows, you issue MQSC commands to a queue manager from the **runmqsc** command prompt. You can use this command prompt in a number of ways:
  - Interactively, issuing MQSC commands from a keyboard. See “Running MQSC commands interactively under runmqsc” on page 17.
  - Issuing MQSC commands from an ASCII text file. See “Running MQSC commands from text files under runmqsc” on page 22.
  - Issuing MQSC commands on a remote queue manager. See “Issuing MQSC commands on a remote queue manager” on page 206.
- IBM i On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. For more information, see “Administering using MQSC commands on IBM i” on page 388.
- z/OS On z/OS, MQSC commands can be issued from a number of sources, depending on the command. For more information, see “Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS” on page 444.

Procedure

- “MQSC command syntax” on page 12
- “MQSC: Special characters and generic values” on page 14
- “Running MQSC commands interactively under runmqsc” on page 17
- “Running MQSC commands from text files under runmqsc” on page 22
- “Automatic configuration from an MQSC script at startup” on page 23

Related tasks

Resolving problems with MQSC commands

Related reference

runmqsc (run MQSC commands)

MQSC command syntax

You can use MQSC commands to manage queue manager objects. MQSC commands are available on all platforms. Some elements of command syntax are platform-specific.

Sequence of parameters

Each command starts with a primary parameter (a verb), and this is followed by a secondary parameter (a noun). This is then followed by the name or generic name of the object (in parentheses) if there is one, which there is on most commands. Following that, parameters can usually occur in any order; if a parameter has a corresponding value, the value must occur directly after the parameter to which it relates.

Note: z/OS On z/OS, the secondary parameter does not have to be second.

Blanks and commas

Keywords, parentheses, and values can be separated by any number of blanks and commas. A comma shown in the syntax diagrams can always be replaced by one or more blanks. There must be at least one blank immediately preceding each parameter (after the primary parameter) except on z/OS .

Any number of blanks can occur at the beginning or end of the command, and between parameters, punctuation, and values. For example, the following command is valid:

```
ALTER QLOCAL ('Account' )      TRIGDPTH ( 1)
```

Blanks within a pair of quotation marks are significant.

Additional commas can appear anywhere where blanks are allowed and are treated as if they were blanks (unless, of course, they are inside strings enclosed by quotation marks).

Repeated parameters

Repeated parameters are not allowed. Repeating a parameter with its "NO" version, as in REPLACE NOREPLACE, is also not allowed.

Strings and single quotation marks

Strings that contain blanks, lowercase characters or special characters must be enclosed in single quotation marks, unless one of the following is true:

- The special characters are one or more of the following characters:
  - Period (.)
  - Forward slash (/)
  - Underscore (_)
  - Percent sign (%)
- z/OS The command is issued from the IBM MQ for z/OS operations and control panels.
- The string is a generic value ending with an asterisk. (on IBM i these must be enclosed in single quotation marks)
- The string is a single asterisk, for example, TRACE(*) (on IBM i these must be enclosed in single quotation marks)
- The string is a range specification containing a colon, for example, CLASS(01:03)

If the string itself contains a single quotation mark, the single quotation mark is represented by two single quotation marks.

Multi On Multiplatforms, a string containing no characters (that is, two single quotation marks with no space in between) is interpreted as a blank space enclosed in single quotation marks, that is, interpreted in the same way as (' '). The exception to this is if the attribute being used is one of the following attributes, when two single quotation marks with no space are interpreted as a zero-length string:

- TOPICSTR
- SUB
- USERDATA
- SELECTOR

z/OS On z/OS, if you want a blank space enclosed in single quotation marks, you must enter it as such (' '). A string containing no characters ('') is the same as entering ().

Any trailing blanks in string attributes that are based on MQCHARV types, such as SELECTOR, sub user data, are treated as significant, which means that `'abc  '` does not equal `'abc'`.

Empty parentheses

An opening parenthesis followed by a closing parenthesis, with no significant information in between, is not valid except where specifically noted. For example, the following string is not valid:

```
NAME ( )
```

Lowercase and uppercase

Keywords are not case sensitive: AltER, alter, and ALTER are all acceptable.

Anything that is not contained within quotation marks is folded to uppercase.

Synonyms

Synonyms are defined for some parameters. For example, DEF is always a synonym for DEFINE, so DEF QLOCAL is valid. Synonyms are not, however, just minimum strings; DEFI is not a valid synonym for DEFINE.

Note: There is no synonym for the DELETE parameter. This is to avoid accidental deletion of objects when using DEF, the synonym for DEFINE.

Special characters

MQSC commands use certain special characters to have certain meanings. For more information about these special characters and how to use them, see "MQSC: Special characters and generic values" on page 14.

Related tasks

Resolving problems with MQSC commands

Related reference

runmqsc (run MQSC commands)

MQSC: Special characters and generic values

Some characters, for example, backslash (\) and double quote (") characters have special meanings when used with MQSC commands. Some special characters that can be used with parameters can have generic values but must be specified correctly.

Precede backslash (\) and double quote (") characters with a \, that is, enter \\ or \" if you want \ or " in your text.

Wherever a parameter can have a generic value, it is entered ending with an asterisk (*), for example ABC*. A generic value means all values beginning with; so ABC* means all values beginning with ABC. If characters that require quotation marks are used in the value, the asterisk must be placed inside the quotation marks, thus 'abc*'. The asterisk must be the last or only character in the value.

The question mark (?) and colon (:) are not allowed in generic values.

When you need to use any of these special characters in a field (for example as part of a description), you must enclose the whole string in single quotation marks.

Table 1. Descriptions of characters that have special meanings

Character	Description
	Blanks are used as separators. Multiple blanks are equivalent to a single blank, except in strings that are enclosed in apostrophes ('). Any trailing blanks in those string attributes which are based on MQCHARV types are treated as significant.

Table 1. Descriptions of characters that have special meanings (continued)	
Character	**Description**
,	Commas are used as separators. Multiple commas are equivalent to a single comma, except in strings that are enclosed in apostrophes (').
'	An apostrophe indicates the beginning or end of a string. IBM MQ leaves all characters that are enclosed in quotation marks exactly as they are entered. The containing apostrophes are not included when calculating the length of the string.
''	Single quotation marks inside a string are treated by IBM MQ as one character when calculating the length of the string and the string is not terminated.
=	z/OS On z/OS, an equals sign indicates the start of a parameter value which is ended by a comma or blank.
(	An open parenthesis indicates the beginning of a parameter value or list of values.
)	A close parenthesis indicates the end of a parameter value or list of values.
:	A colon indicates an inclusive range. For example (1:5) means (1,2,3,4,5). This notation can be used only in **TRACE** commands.
*	An asterisk means all. For example, **DISPLAY TRACE (*)** means display all traces, and **DISPLAY QUEUE (PAY*)** means display all queues with names that begin with PAY.

MQSC input file syntax

If you have long commands, or are using a particular sequence of commands repeatedly, you can use an input file to issue MQSC commands. The contents of the input file must follow the syntax described in this topic.

Overview

MQSC commands are input through the *standard input device*, also referred to as `stdin`. Typically this is the keyboard, but you can specify that input is to come from an input file.

You can use this input file with any of the following platform-specific tools:

- ALW The **runmqsc** command on AIX, Linux, and Windows. See “Running MQSC commands from text files under runmqsc” on page 22
- IBM i The **STRMQM** command on IBM i. See “Administering using MQSC commands on IBM i” on page 388
- z/OS The CSQINP1, CSQINP2, and CSQINPX initialization data sets or the CSQUTIL batch utility on z/OS. See “Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS” on page 444

Syntax

MQSC input file syntax:

- For portability among IBM MQ environments, limit the line length in MQSC command files to 72 characters.
- Each command must start on a new line.
- A line starting with an asterisk (*) in the first position is ignored. This can be used to insert comments into the file.
- Blank lines are ignored.

- A plus sign (+) indicates that the command is continued from the first non-blank character in the next line. If you use + to continue a command, remember to leave at least one blank before the next parameter (except on z/OS where this is not necessary). Any comments or blank lines are discarded when the command is reassembled into a single string.
- A minus sign (-), this indicates that the command is to be continued from the start of the next line. Any comments or blank lines are discarded when the command is reassembled into a single string.
- MQSC commands that are contained within an Escape PCF (Programmable Command Format) command cannot be continued with the plus sign, or the minus sign. The entire command must be contained within a single Escape command. For information about the PCF commands, see “Introduction to IBM MQ Programmable Command Formats” on page 25.
- On Multiplatforms, and on z/OS for commands issued from the CSQUTIL batch utility program, you can use a semicolon character (;) to terminate a command, even if you have entered a plus sign (+) at the end of the previous line.
- A line must not end in a keyboard control character (for example, a tab).
- If you run the **runmqsc** command in client mode by redirecting `stdin` from a text file, and you supply the **-u** flag to provide credentials, the **runmqsc** command does not prompt for a password and instead the password is read from `stdin`. You should ensure that the first line of data provided through `stdin` is the password. This can be done by using command line tools such as "echo" or "cat" and passing the password followed by the MQSC script into the **runmqsc** command `stdin`.
- Windows On Windows, if certain special characters such as the pound sign (£) and the logical NOT (¬) are used in a command script (for example, as part of an object description), they are displayed differently in the output from a command such as **DISPLAY QLOCAL**.

See also “MQSC command syntax” on page 12.

Examples

The following example is an extract from an MQSC command file that shows command **DEFINE QLOCAL**.

```
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE +
DESCR(' ') +
PUT(ENABLED) +
DEFPRTY(0) +
DEFPSIST(NO) +
GET(ENABLED) +
MAXDEPTH(5000) +
MAXMSGL(1024) +
DEFSOPT(SHARED) +
NOHARDENBO +
USAGE(NORMAL) +
NOTRIGGER;
```

Figure 1. Extract from an MQSC command file

When the **runmqsc** command completes, a report is returned. The following example is an extract from a report:

```
Starting MQSC for queue manager jupiter.queue.manager.
.
.
12:   DEFINE QLOCAL('ORANGE.LOCAL.QUEUE') REPLACE +
:      DESCR(' ') +
:      PUT(ENABLED) +
:      DEFPRTY(0) +
:      DEFPSIST(NO) +
:      GET(ENABLED) +
:      MAXDEPTH(5000) +
:      MAXMSGL(1024) +
:      DEFSOPT(SHARED) +
:      NOHARDENBO +
:      USAGE(NORMAL) +
:      NOTRIGGER;
AMQ8006: IBM MQ queue created.
:
.
.
```

Figure 2. Extract from an MQSC command report file

You can also use the example MQSC command files to help you create your text file:

amqscos0.tst
Definitions of objects used by sample programs.

amqscic0.tst
Definitions of queues for CICS® transactions.

Linux AIX On AIX and Linux, these files are located in the directory *MQ_INSTALLATION_PATH*/samp. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

Windows On Windows, these files are located in the directory *MQ_INSTALLATION_PATH*\tools\mqsc\samples. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

ALW Running MQSC commands interactively under `runmqsc`

On AIX, Linux, and Windows, you can use the **runmqsc** command prompt to issue MQSC commands to a queue manager interactively. Interactive running is particularly suitable for quick tests.

Before you begin

You must use the **runmqsc** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with by using the `dspmq -o installation` command.

You can make it easier to see that you are in an MQSC environment and see some details of the current environment by setting a prompt of your choice using the **MQPROMPT** environment variable. For more information, see “Setting the MQSC command prompt” on page 20.

Linux AIX When you run MQSC commands interactively on AIX and Linux platforms, the **runmqsc** command prompt also supports additional command line editor functions. See “Enabling command recall and completion, and Emacs command keys, for runmqsc” on page 21.

About this task

The **runmqsc** command is used to open a command prompt from which you can issue MQSC commands. These commands and their syntax are described in the MQSC commands reference.

When you start the **runmqsc** command prompt as described in this task, you set the prompt to run in one of three modes, depending on the flags set on the command:

- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not run.
- *Direct mode*, where the MQSC commands are run on a local queue manager.
- *Indirect mode*, where the MQSC commands are run on a remote queue manager.

The procedure below sets the prompt to run in direct mode. Other options are illustrated in the examples that follow the main steps.

Procedure

1. Open a command window or shell and enter the following command:

   ```
   runmqsc QMgrName
   ```

   Where *QMgrName* specifies the name of the queue manager that you want to process the MQSC commands. You can leave *QMgrName* blank to process MQSC commands on the default queue manager.
2. Type in any MQSC commands, as required. For example, to create a local queue called ORANGE.LOCAL.QUEUE enter the following command:

   ```
   DEFINE QLOCAL (ORANGE.LOCAL.QUEUE)
   ```

   For commands that have too many parameters to fit on one line, use continuation characters to indicate that a command is continued on the following line:

   - A minus sign (-) indicates that the command is to be continued from the start of the following line.
   - A plus sign (+) indicates that the command is to be continued from the first non-blank character on the following line.

   Command input terminates with the final character of a non-blank line that is not a continuation character. You can also terminate command input explicitly by entering a semicolon (;).
3. Stop working with MQSC commands by entering the following command:

   ```
   end
   ```

   Alternatively, you can use the **exit** command, the **quit** command, or the EOF character for your operating system.

Results

When you issue MQSC commands, the queue manager returns operator messages that confirm your actions or tell you about the errors you have made. For example, the following message confirms a queue is created:

```
AMQ8006: IBM MQ queue created.
```

The following message indicates that you have made a syntax error:

```
AMQ8405: Syntax error detected at or near end of command segment below:-
AMQ8426: Valid MQSC commands are:

ALTER
CLEAR
DEFINE
DELETE
DISPLAY
END
PING
REFRESH
RESET
RESOLVE
RESUME
```

```
START
STOP
SUSPEND
4 : end
```

These messages are sent to the standard output device, which by default is the display. If you have not entered the command correctly, refer to the reference information for the command to find the correct syntax. See MQSC commands reference.

Example

The following are variants of the `runmqsc` *QMgrName* command that is used in the previous steps. These variants create different configurations of the **runmqsc** command prompt.

- The following command uses command filtering to pass a single MQSC command to the MQSC interpreter.

  Windows On Windows:

  ```
  echo display chstatus(*) | runmqsc QMname
  ```

  Linux On Linux:

  ```
  echo "display chstatus(*)" | runmqsc QMname
  ```

- The following command does not specify a queue manager name, so the MQSC commands are processed on the default queue manager.

  ```
  runmqsc
  ```

- This command submits commands to the QMREMOTE queue manager, using QMLOCAL to submit the commands.

  ```
  runmqsc -w 30 -m QMLOCAL QMREMOTE
  ```

- This command verifies that the command syntax is correct on a local queue manager without running the commands. Note that the commands to be verified are read from an input file `myprog.in`.

  ```
  runmqsc -f myprog.in -v QmgrName
  ```

  For more information about working with input and output files, see “Running MQSC commands from text files under runmqsc” on page 22.

What to do next

For full details of the **runmqsc** command syntax, optional parameters and return codes, see runmqsc (run MQSC commands).

Related tasks

“Running MQSC commands from text files under runmqsc” on page 22
If you have long commands, or are using a particular sequence of commands repeatedly, you can use a text file to issue MQSC commands. You can redirect `stdin` from a text file. You can also redirect the output to a file.

Related reference

MQSC commands reference

ALW Setting the MQSC command prompt

On AIX, Linux, and Windows, use the **MQPROMPT** environment variable to set the prompt that is displayed when you run the **runmqsc** command. This makes it easier to see that you are in an MQSC environment, and to see some details of the current environment.

About this task

You can set the prompt that is displayed when you run the **runmqsc** command. The prompt is inserted both when the **runmqsc** command is run interactively, and when input is redirected into **runmqsc** from a file or from the standard input device (stdin).

You can include plain text in the command prompt, and you can also insert environment variables by using the +VARNAME+ notation in the same manner as IBM MQ service object definitions. For more information, see "Using replaceable inserts on service definitions" on page 190.

There are a number of other additional replaceable inserts IBM MQ provides, described in the following table.

Replaceable insert	Description
MQ_HOST_NAME	Host name of the system
MQ_FILE_SEP	Platform specific file separator: • Linux AIX On AIX and Linux systems, the **MQ_FILE_SEP** is /. • Windows On Windows systems, the location of the **MQ_FILE_SEP** is \
MQ_PATH_SEP	Platform specific path separator: • Linux AIX On AIX and Linux systems, the **MQ_PATH_SEP** is :. • Windows On Windows systems, the location of the **MQ_PATH_SEP** is ;
MQ_DATE_TIME	Local system date and time in a fixed YYYY-MM-DD hh:mm:ss.SSS format, for example: `2020-12-25 17:41:37.408`

Notes:

- The MQ replaceable inserts values relate to the IBM MQ installation and host system the **runmqsc** command is associated with.
- **MQPROMPT** is limited to a maximum of 256 characters when inserts are expanded. **MQPROMPT** expansions over this value result in the entire **MQPROMPT** string being truncated without the expansions.

Example

The following example sets the prompt to MQSC:

- Linux AIX

```
export MQPROMPT="MQSC"
```

- Windows

```
set "MQPROMPT=MQSC"
```

AIX The following example sets the **MQPROMPT** variable on an AIX system. The prompt is set to display a username (taken from the associated system environment variable), the queue manager name, and the IBM MQ host name (taken from the IBM MQ replaceable inserts):

```
sh> export MQPROMPT="+USER+ @ +QMNAME+ @ +MQ_HOST_NAME+> "
sh> runmqsc MY.QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager MY.QMGR.

myuser @ MY.QMGR @ aix1> DISPLAY QMSTATUS
```

```
C:\ > set "MQPROMPT=+USERNAME+ @ +QMNAME+ @ +MQ_HOST_NAME+> "
C:\ > runmqsc MY.QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager MY.QMGR.

myuser @ MY.QMGR @ WIN1> DISPLAY QMSTATUS
```

The following example adds a timestamp to the **MQPROMPT** examples above, taken from the MQ replaceable inserts:

```
sh> export MQPROMPT="+MQ_DATE_TIME+ +USER+ @ +QMNAME+ @ +MQ_HOST_NAME+> "
sh> runmqsc MY.QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager MY.QMGR.

2020-11-24 18:10:00.404 myuser @ MY.QMGR @ aix1> DISPLAY QMSTATUS
```

```
C:\ > set "MQPROMPT=+MQ_DATE_TIME+ +USERNAME+ @ +QMNAME+ @ +MQ_HOST_NAME+> "
C:\ > runmqsc MY.QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager MY.QMGR.

2020-11-24 18:10:01.007 myuser @ MY.QMGR @ WIN1> DISPLAY QMSTATUS
```

Linux AIX Enabling command recall and completion, and Emacs command keys, for `runmqsc`

Use the **runmqsc** command prompt on AIX and Linux to enable command recall, command completion, and Emacs command keys.

About this task

On AIX and Linux systems, you can make the following additional command line editor functions available from the **runmqsc** command prompt:

- Recall of previously entered commands by using the up arrow key and the down arrow key
- Automatic completion for the next keyword of a command by using the tab key and the space bar
- Emacs command keys, or similar command key functions

To use these functions, the curses library must be installed. If the curses library is not installed on your system, the **runmqsc** prompt does not have the command line editor functions and a message is shown when the **runmqsc** command prompt is started. The name of the curses library to install depends on the UNIX platform:

- AIX On AIX, install `curses`.
- Linux On Linux, install `ncurses`.

Procedure

- Install `ncurses` or `curses`.

  Note: The following example uses instructions for Linux

  Run the following command to find the existing `ncurses` packages:

```
rpm -qa | grep -i ncurses
```

  The required `ncurses` packages are as follows:

```
ncurses-term-6.1-7.20180224.el8.noarch
ncurses-6.1-7.20180224.el8.x86_64
ncurses-base-6.1-7.20180224.el8.noarch
ncurses-c++-libs-6.1-7.20180224.el8.x86_64
ncurses-libs-6.1-7.20180224.el8.x86_64
ncurses-compat-libs-6.1-7.20180224.el8.x86_64
ncurses-devel-6.1-7.20180224.el8.x86_64
```

  You can install all of the required `ncurses` packages listed in the preceding text by running the following command:

```
dnf install ncurses*
```

- Customize the Emacs key bindings.

  You can customize the keys that are bound to the commands. For example, you can bind the keys to vi bindings instead of the default Emacs key bindings.

  The keys are customized by editing the `.editrc` file that is stored in the home directory. For more information, see editrc in the FreeBSD man pages.

- Disable command recall, command completion, and Emacs command keys.

  To do this, set the environment variable **MQ_OVERRIDE_LIBEDIT_LOAD** to TRUE.

  This environment variable can be used as a workaround when the **runmqsc** command prompt displays the following informational message:

```
AMQ8521I: Command completion and history unavailable
```

ALW Running MQSC commands from text files under `runmqsc`

If you have long commands, or are using a particular sequence of commands repeatedly, you can use a text file to issue MQSC commands. You can redirect `stdin` from a text file. You can also redirect the output to a file.

Before you begin

This task assumes that you have created a text file that contains the MQSC commands that you want to run. For detailed syntax and examples of these files, see “MQSC input file syntax” on page 15.

You can set the MQSC command prompt to a prompt of your choice by using the **MQPROMPT** environment variable. For more information, see “Setting the MQSC command prompt” on page 20.

About this task

Input for the **runmqsc** command is taken from the *standard input device*, also referred to as `stdin`. Typically this is the keyboard, but you can specify that input is to come from a serial port or a file.

Output for the **runmqsc** command is output to the *standard output device*, also referred to as `stdout`. Typically this is a display, but you can redirect output to a serial port or a file.

Procedure

1. On a local queue manager, verify that the command syntax in the file is correct without running the commands.

   Use the **-v** flag on the **runmqsc** command, along with one of the following options:

   - Use the **-f** option to identify the input text filename. For example:

     ```
     runmqsc -f myprog.in -v localQmgrName
     ```

   You cannot specify a remote queue manager when verifying commands. That is, you cannot specify the **-w** flag.

   The returned report is similar to that shown in Figure 2 on page 17.
2. When the command syntax is correct, remove the **-v** flag then rerun the **runmqsc** command.

   Note that you can now specify a remote queue manager.

   - Run (for example) the following command:

     ```
     runmqsc -f myprog.in QmgrName
     ```

   Figure 1 on page 16 shows an extract from a command file such as `myprog.in` and Figure 2 on page 17 shows the corresponding extract of the output from a report file such as `results.out`.

What to do next

For full details of the **runmqsc** command syntax, optional parameters and return codes, see runmqsc (run MQSC commands).

Related tasks

"Setting the MQSC command prompt" on page 20
On AIX, Linux, and Windows, use the **MQPROMPT** environment variable to set the prompt that is displayed when you run the **runmqsc** command. This makes it easier to see that you are in an MQSC environment, and to see some details of the current environment.

"Running MQSC commands interactively under runmqsc" on page 17
On AIX, Linux, and Windows, you can use the **runmqsc** command prompt to issue MQSC commands to a queue manager interactively. Interactive running is particularly suitable for quick tests.

Related reference

MQSC commands reference

Multi Automatic configuration from an MQSC script at startup

You can configure your queue manager to automatically apply the contents of an MQSC script, or set of MQSC scripts, on every queue manager start.

You can use this functionality to have a configuration which can be modified, and automatically replayed at the next queue manager restart. As an example, if the script or scripts are located on a mounted drive, it is possible to have a centralized configuration where the latest version gets applied to every queue manager as they start.

A particular scenario in which this can be useful, is to ensure a uniform cluster contains the same definitions on all queue managers in the cluster, by having a single set of configuration which they all apply. For an example of this, see Creating a new uniform cluster.

Before you begin

You can use:

1. A single script, and create a text file using MQSC commands.

2. A set of MQSC scripts:
   - To identify a directory where the configurations will exist, and
   - In that directory, create files, each with the extension `.mqsc`, for example `queues.mqsc`.

Given that this script is reapplied on every queue manager start, it is important that commands can be replayed. For example, a **DEFINE** command must include the **REPLACE** string, otherwise the command appears as a failure on the second queue manager start, as the object already exists.

Note that in an MQSC script, any line prefixed with * is treated as a comment.

Enabling automatic configuration of MQSC scripts

Important: You must not issue commands for channels of type MQTT, as they are not supported for automatic configuration during startup.

You can configure a new queue manager by using the **-ic** flag to the **crtmqm** command, and pointing either at a specific file, or directory. The supplied value is stored in the `qm.ini` file under the AutoConfig stanza, as attribute **MQSCConfig**.

You can configure an existing queue manager to enable automatic MQSC configuration, by adding the AutoConfig stanza attribute **MQSCConfig**, pointing to a valid file or directory. For example:

```
AutoConfig:
   MQSCConfig=C:\mq_configuration\uniclus.mqsc
```

How does automatic configuration work?

During queue manager startup, the configuration identified by the **AutoConfig** stanza attribute **MQSCConfig** is passed through **runmqsc** validation, to ensure valid syntax, and then stored in the queue manager data tree into the `autocfg` directory as a single file `cached.mqsc`.

When multiple files from a directory are processed in, they are processed in alphabetical order, and if it contains an MQSC end or quit command, the rest of the contents of that file are skipped.

During the first ever start of the queue manager, an inability to read the file or directory, or a file with MQSC syntax that is not valid, prevents the queue manager from starting, with an appropriate error message both to the console and to the queue manager error log.

On subsequent restarts, if the file or directory pointed to is unreadable or contains invalid MQSC syntax, the previously cached file is used and a message written to the error log of the queue manager highlights this.

At the point the contents of the `cached.mqsc` are applied to the queue manager, when all of the MQSC commands have been applied, the queue manager is enabled for applications to connect. The **runmqsc** log of the configuration being applied is stored in the errors directory of the queue manager, as a file called `autocfgmqsc.LOG`.

In addition, any MQSC command that does not complete successfully, is logged to the queue manager error log, identifying why the command fails.

Automating IBM MQ administration using PCF commands

You might decide that it would be beneficial to your installation to automate some administration and monitoring tasks. You can automate administration tasks for both local and remote queue managers using programmable command format (PCF) commands. This section assumes that you have experience of administering IBM MQ objects.

PCF commands

IBM MQ programmable command format (PCF) commands can be used to program administration tasks into an administration program. In this way, from a program you can manipulate queue manager objects

(queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and even manipulate the queue managers themselves.

PCF commands cover the same range of functions provided by MQSC commands. You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an IBM MQ message. Each command is sent to the target queue manager using the MQI function MQPUT in the same way as any other message. Providing the command server is running on the queue manager receiving the message, the command server interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned in another data structure. The application can then process the reply and act accordingly.

Note: Unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

Briefly, these are some of the things needed to create a PCF command message:

Message descriptor
This is a standard IBM MQ message descriptor, in which:

- Message type (*MsqType*) is MQMT_REQUEST.
- Message format (*Format*) is MQFMT_ADMIN.

Application data
Contains the PCF message including the PCF header, in which:

- The PCF message type (*Type*) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

For a complete description of the PCF data structures and how to implement them, see "Introduction to IBM MQ Programmable Command Formats" on page 25.

PCF object attributes

Object attributes in PCF are not limited to eight characters as they are for MQSC commands. They are shown in this guide in italics. For example, the PCF equivalent of RQMNAME is *RemoteQMgrName*.

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about escape PCFs, see Escape.

Introduction to IBM MQ Programmable Command Formats

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration. They can be used to solve the problem of complex administration of distributed networks especially as networks grow in size and complexity.

Programmable Command Formats are supported on all IBM MQ platforms.

The problem PCF commands solve

The administration of distributed networks can become complex. The problems of administration continue to grow as networks increase in size and complexity.

Examples of administration specific to messaging and queuing include:

- Resource management.

For example, queue creation and deletion.

- Performance monitoring.

  For example, maximum queue depth or message rate.

- Control.

  For example, tuning queue parameters such as maximum queue depth, maximum message length, and enabling and disabling queues.

- Message routing.

  Definition of alternative routes through a network.

IBM MQ PCF commands can be used to simplify queue manager administration and other network administration. PCF commands allow you to use a single application to perform network administration from a single queue manager within the network.

What are PCFs?

PCFs define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. You can use PCF commands in a systems management application program for administration of IBM MQ objects: authentication information objects, channels, channel listeners, namelists, process definitions, queue managers, queues, services, and storage classes. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local, or remote, using the local queue manager.

Each queue manager has an administration queue with a standard queue name and your application can send PCF command messages to that queue. Each queue manager also has a command server to service the command messages from the administration queue. PCF command messages can therefore be processed by any queue manager in the network and the reply data can be returned to your application, using your specified reply queue. PCF commands and reply messages are sent and received using the normal Message Queue Interface (MQI).

For a list of the available PCF commands, including their parameters, see Definitions of the Programmable Command Formats.

Using IBM MQ Programmable Command Formats

You can use PCFs in a systems management program for IBM MQ remote administration.

This section includes:

- “PCF command messages” on page 26
- “PCF responses in IBM MQ” on page 29
- z/OS “Extended responses” on page 31
- Rules for naming IBM MQ objects
- “Authority checking for PCF commands in IBM MQ” on page 32

PCF command messages

PCF command messages consist of a PCF header, parameters identified in that header and also user-defined message data. The messages are issued using Message Queue interface calls.

Each command and its parameters are sent as a separate command message containing a PCF header followed by a number of parameter structures; for details of the PCF header, see MQCFH - PCF header, and for an example of a parameter structure, see MQCFST - PCF string parameter. The PCF header identifies the command and the number of parameter structures that follow in the same message. Each parameter structure provides a parameter to the command.

Replies to the commands, generated by the command server, have a similar structure. There is a PCF header, followed by a number of parameter structures. Replies can consist of more than one message but commands always consist of one message only.

Multi On Multiplatforms, the queue to which the PCF commands are sent is always called the SYSTEM.ADMIN.COMMAND.QUEUE.

z/OS On z/OS, commands are sent to SYSTEM.COMMAND.INPUT, although SYSTEM.ADMIN.COMMAND.QUEUE can be an alias for it. The command server servicing this queue sends the replies to the queue defined by the *ReplyToQ* and *ReplyToQMgr* fields in the message descriptor of the command message.

How to issue PCF command messages

Use the normal Message Queue Interface (MQI) calls, MQPUT, MQGET, and so on, to put and retrieve PCF command and response messages to and from their queues.

Note:

Ensure that the command server is running on the target queue manager for the PCF command to process on that queue manager.

For a list of supplied header files, see IBM MQ COPY, header, include and module files.

Message descriptor for a PCF command

The IBM MQ message descriptor is fully documented in MQMD - Message descriptor.

A PCF command message contains the following fields in the message descriptor:

Report
Any valid value, as required.

MsgType
This field must be MQMT_REQUEST to indicate a message requiring a response.

Expiry
Any valid value, as required.

Feedback
Set to MQFB_NONE

Multi **Encoding**
If you are sending to an IBM MQ for Multiplatforms system, set this field to the encoding used for the message data. Conversion is performed if necessary.

Multi **CodedCharSetId**
If you are sending to an IBM MQ for Multiplatforms system, set this field to the coded character-set identifier used for the message data. Conversion is performed if necessary.

Format
Set to MQFMT_ADMIN.

Priority
Any valid value, as required.

Persistence
Any valid value, as required.

MsgId
The sending application can specify any value, or MQMI_NONE can be specified to request the queue manager to generate a unique message identifier.

CorrelId
The sending application can specify any value, or MQCI_NONE can be specified to indicate no correlation identifier.

ReplyToQ
The name of the queue to receive the response.

ReplyToQMgr
The name of the queue manager for the response (or blank).

Message context fields
These fields can be set to any valid values, as required. Normally the Put message option MQPMO_DEFAULT_CONTEXT is used to set the message context fields to the default values.

If you are using a version-2 MQMD structure, you must set the following additional fields:

GroupId
Set to MQGI_NONE

MsgSeqNumber
Set to 1

Offset
Set to 0

MsgFlags
Set to MQMF_NONE

OriginalLength
Set to MQOL_UNDEFINED

Sending user data

The PCF structures can also be used to send user-defined message data. In this case the message descriptor *Format* field must be set to MQFMT_PCF.

Sending and receiving PCF messages in a specified queue

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 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.
- Put Message Options using the MQPMO structure. For more information about the MQPMO structure, see MQPMO - Put-message options.
- 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.

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 MQGMO - Get-message options.
- Get Message Options using the MQGMO structure. For more information about the MQGMO structure, see MQMD - Message Descriptor.
- The handle of the bag to contain the converted message.

For a full description of the mqGetBag call, see mqGetBag.

PCF responses in IBM MQ

In response to each command, the command server generates one or more response messages. A response message has a similar format to a command message.

The PCF header has the same command identifier value as the command to which it is a response (see MQCFH - PCF header for details). The message identifier and correlation identifier are set according to the report options of the request.

If the PCF header type of the command message is MQCFT_COMMAND, standard responses only are generated. Such commands are supported on Multiplatforms only. Older applications do not support PCF on z/OS ; the IBM MQ Windows Explorer is one such application (however, the IBM WebSphere® MQ 6.0 or later IBM MQ Explorer does support PCF on z/OS).

If the PCF header type of the command message is MQCFT_COMMAND_XR, either extended or standard responses are generated. Such commands are supported on z/OS and some Multiplatforms. Commands issued on z/OS generate only extended responses.

If a single command specifies a generic object name, a separate response is returned in its own message for each matching object. For response generation, a single command with a generic name is treated as multiple individual commands (except for the control field MQCFC_LAST or MQCFC_NOT_LAST). Otherwise, one command message generates one response message.

Certain PCF responses might return a structure even when it is not requested. This structure is shown in the definition of the response (Definitions of the Programmable Command Formats) as *always returned*. The reason that, for these responses, it is necessary to name the objects in the response to identify which object the data applies.

Message descriptor for a response

A response message has the following fields in the message descriptor:

MsgType
This field is MQMT_REPLY.

MsgId
This field is generated by the queue manager.

CorrelId
This field is generated according to the report options of the command message.

Format
This field is MQFMT_ADMIN.

Encoding
Set to MQENC_NATIVE.

CodedCharSetId
Set to MQCCSI_Q_MGR.

Persistence
The same as in the command message.

Priority
The same as in the command message.

The response is generated with MQPMO_PASS_IDENTITY_CONTEXT.

Multi *Standard responses*

Command messages with a header type of MQCFT_COMMAND, standard responses are generated. Such commands are supported only on Multiplatforms.

There are three types of standard response:

- OK response
- Error response
- Data response

OK response

This response consists of a message starting with a command format header, with a *CompCode* field of MQCC_OK or MQCC_WARNING.

For MQCC_OK, the *Reason* is MQRC_NONE.

For MQCC_WARNING, the *Reason* identifies the nature of the warning. In this case the command format header might be followed by one or more warning parameter structures appropriate to this reason code.

In either case, for an inquire command further parameter structures might follow as described in the following sections.

Error response

If the command has an error, one or more error response messages are sent (more than one might be sent even for a command that would normally have only a single response message). These error response messages have MQCFC_LAST or MQCFC_NOT_LAST set as appropriate.

Each such message starts with a response format header, with a *CompCode* value of MQCC_FAILED and a *Reason* field that identifies the particular error. In general, each message describes a different error. In addition, each message has either zero or one (never more than one) error parameter structures following the header. This parameter structure, if there is one, is an MQCFIN structure, with a *Parameter* field containing one of the following:

- MQIACF_PARAMETER_ID

  The *Value* field in the structure is the parameter identifier of the parameter that was in error (for example, MQCA_Q_NAME).
- MQIACF_ERROR_ID

  This value is used with a *Reason* value (in the command format header) of MQRC_UNEXPECTED_ERROR. The *Value* field in the MQCFIN structure is the unexpected reason code received by the command server.
- MQIACF_SELECTOR

  This value occurs if a list structure (MQCFIL) sent with the command contains a duplicate selector or one that is not valid. The *Reason* field in the command format header identifies the error, and the *Value* field in the MQCFIN structure is the parameter value in the MQCFIL structure of the command that was in error.
- MQIACF_ERROR_OFFSET

  This value occurs when there is a data compare error on the Ping Channel command. The *Value* field in the structure is the offset of the Ping Channel compare error.
- MQIA_CODED_CHAR_SET_ID

This value occurs when the coded character-set identifier in the message descriptor of the incoming PCF command message does not match that of the target queue manager. The *Value* field in the structure is the coded character-set identifier of the queue manager.

The last (or only) error response message is a summary response, with a *CompCode* field of MQCC_FAILED, and a *Reason* field of MQRCCF_COMMAND_FAILED. This message has no parameter structure following the header.

Data response

This response consists of an OK response (as described earlier) to an inquire command. The OK response is followed by additional structures containing the requested data as described in Definitions of the Programmable Command Formats.

Applications must not depend upon these additional parameter structures being returned in any particular order.

z/OS *Extended responses*

Commands issued on z/OS generate extended responses.

There are three types of extended response:

- Message response, with type MQCFT_XR_MSG
- Item response, with type MQCFT_XR_ITEM
- Summary response, with type MQCFT_XR_SUMMARY

Each command can generate one, or more, sets of responses. Each set of responses comprises one or more messages, numbered sequentially from 1 in the *MsgSeqNumber* field of the PCF header. The *Control* field of the last (or only) response in each set has the value MQCFC_LAST. For all other responses in the set, this value is MQCFC_NOT_LAST.

Any response can include one, or more, optional MQCFBS structures in which the *Parameter* field is set to MQBACF_RESPONSE_SET, the value being a response set identifier. Identifiers are unique and identify the set of responses which contain the response. For every set of responses, there is an MQCFBS structure that identifies it.

Extended responses have at least two parameter structures:

- An MQCFBS structure with the *Parameter* field set to MQBACF_RESPONSE_ID. The value in this field is the identifier of the set of responses to which the response belongs. The identifier in the first set is arbitrary. In subsequent sets, the identifier is one previously notified in an MQBACF_RESPONSE_SET structure.
- An MQCFST structure with the *Parameter* field set to MQCACF_RESPONSE_Q_MGR_NAME, the value being the name of the queue manager from which the set of responses come.

Many responses have additional parameter structures, and these structures are described in the following sections.

You cannot determine in advance how many responses there are in a set other than by getting responses until one with MQCFC_LAST is found. Neither can you determine in advance how many sets of responses there are as any set might include MQBACF_RESPONSE_SET structures to indicate that additional sets are generated.

Extended responses to Inquire commands

Inquire commands normally generate an item response (type MQCFT_XR_ITEM) for each item found that matches the specified search criteria. The item response has a *CompCode* field in the header with a value of MQCC_OK, and a *Reason* field with a value of MQRC_NONE. It also includes other parameter structures describing the item and its requested attributes, as described in Definitions of the Programmable Command Formats.

If an item is in error, the *CompCode* field in the header has a value of MQCC_FAILED and the *Reason* field identifies the particular error. Additional parameter structures are included to identify the item.

Certain Inquire commands might return general (not name-specific) message responses in addition to the item responses. These responses are informational, or error, responses of the type MQCFT_XR_MSG.

If the Inquire command succeeds, there might, optionally, be a summary response (type MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_OK, and a *Reason* field value of MQRC_NONE.

If the Inquire command fails, item responses might be returned, and there might optionally be a summary response (type MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_FAILED, and a *Reason* field value of MQRCCF_COMMAND_FAILED.

Extended responses to commands other than Inquire

Successful commands generate message responses in which the *CompCode* field in the header has a value of MQCC_OK, and the *Reason* field has a value of MQRC_NONE. There is always at least one message; it might be informational (MQCFT_XR_MSG) or a summary (MQCFT_XR_SUMMARY). There might optionally be additional informational (type MQCFT_XR_MSG) messages. Each informational message might include a number of additional parameter structures with information about the command; see the individual command descriptions for the structures that can occur.

Commands that fail generate error message responses (type MQCFT_XR_MSG), in which the *CompCode* field in the header has a value of MQCC_FAILED and the *Reason* field identifies the particular error. Each message might include a number of additional parameter structures with information about the error: see the individual error descriptions for the structures that can occur. Informational message responses might be generated. There might, optionally, be a summary response (MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_FAILED, and a *Reason* field value of MQRCCF_COMMAND_FAILED.

Extended responses to commands using CommandScope

If a command uses the **CommandScope** parameter, or causes a command using the **CommandScope** parameter to be generated, there is an initial response set from the queue manager where the command was received. Then a separate set, or sets, of responses is generated for each queue manager to which the command is directed (as if multiple individual commands were issued). Finally, there is a response set from the receiving queue manager which includes an overall summary response (type MQCFT_XR_SUMMARY). The MQCACF_RESPONSE_Q_MGR_NAME parameter structure identifies the queue manager that generates each set.

The initial response set has the following additional parameter structures:

- MQIACF_COMMAND_INFO (MQCFIN). Possible values in this structure are MQCMDI_CMDSCOPE_ACCEPTED or MQCMDI_CMDSCOPE_GENERATED.
- MQIACF_CMDSCOPE_Q_MGR_COUNT (MQCFIN). This structure indicates the number of queue managers to which the command is sent.

Authority checking for PCF commands in IBM MQ

When a PCF command is processed, the *UserIdentifier* from the message descriptor in the command message is used for the required IBM MQ object authority checks. Authority checking is implemented differently on each platform as described in this topic.

The checks are performed on the system on which the command is being processed; therefore this user ID must exist on the target system and have the required authorities to process the command. If the message has come from a remote system, one way of achieving the ID existing on the target system is to have a matching user ID on both the local and remote systems.

Note: z/OS For information about authority checking on z/OS, see Task 1: Identify the z/OS system parameters.

IBM MQ for IBM i

IBM i

In order to process any PCF command, the user ID must have *dsp* authority for the IBM MQ object on the target system.

In addition, IBM MQ object authority checks are performed for certain PCF commands, as shown in Table 2 on page 34.

In most cases these checks are the same checks as those checks performed by the equivalent IBM MQ CL commands issued on a local system. See the Setting up security on IBM i , for more information about the mapping from IBM MQ authorities to IBM i system authorities, and the authority requirements for the IBM MQ CL commands. Details of security concerning exits are given in the Link level security using a security exit documentation.

To process any of the following commands the user ID must be a member of the group profile QMQMADM:

- Ping Channel
- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Reset Channel
- Resolve Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener

IBM MQ for UNIX, Linux, and Windows

ALW

In order to process any PCF command, the user ID must have *dsp* authority for the queue manager object on the target system. In addition, IBM MQ object authority checks are performed for certain PCF commands, as shown in Table 2 on page 34.

To process any of the following commands the user ID must belong to group *mqm*.

Note: For Windows **only**, the user ID can belong to group *Administrators* or group *mqm*.

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel
- Reset Cluster

- Refresh Cluster
- Suspend Queue Manager
- Resume Queue Manager

IBM MQ Object authorities for Multiplatforms

Multi

Table 2. Object authorities		
Command	**IBM MQ object authority**	**Class authority (for object type)**
Change Authentication Information	dsp and chg	n/a
Change Channel	dsp and chg	n/a
Change Channel Listener	dsp and chg	n/a
Change Client Connection Channel	dsp and chg	n/a
Change Namelist	dsp and chg	n/a
Change Process	dsp and chg	n/a
Change Queue	dsp and chg	n/a
Change Queue Manager	chg *see Note 3 and Note 5*	n/a
Change Service	dsp and chg	n/a
Clear Queue	clr	n/a
Copy Authentication Information	dsp	crt
Copy Authentication Information (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Channel	dsp	crt
Copy Channel (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Channel Listener	dsp	crt
Copy Channel Listener (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Client Connection Channel	dsp	crt
Copy Client Connection Channel (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Namelist	dsp	crt
Copy Namelist (Replace) *see Note 1*	*from:* dsp *to:* dsp and chg	crt
Copy Process	dsp	crt
Copy Process (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Queue	dsp	crt

Table 2. Object authorities (continued)		
Command	**IBM MQ object authority**	**Class authority (for object type)**
Copy Queue (Replace) *see Note 1*	*from:* dsp *to:* dsp and chg	crt
Create Authentication Information	*(system default authentication information)* dsp	crt
Create Authentication Information (Replace) *see Note 1*	*(system default authentication information)* dsp *to:* chg	crt
Create Channel	*(system default channel)* dsp	crt
Create Channel (Replace) *see Note 1*	*(system default channel)* dsp *to:* chg	crt
Create Channel Listener	*(system default listener)* dsp	crt
Create Channel Listener (Replace) *see Note 1*	*(system default listener)* dsp *to:* chg	crt
Create Client Connection Channel	*(system default channel)* dsp	crt
Create Client Connection Channel (Replace) *see Note 1*	*(system default channel)* dsp *to:* chg	crt
Create Namelist	*(system default namelist)* dsp	crt
Create Namelist (Replace) *see Note 1*	*(system default namelist)* dsp *to:* dsp and chg	crt
Create Process	*(system default process)* dsp	crt
Create Process (Replace) *see Note 1*	*(system default process)* dsp *to:* chg	crt
Create Queue	*(system default queue)* dsp	crt
Create Queue (Replace) *see Note 1*	*(system default queue)* dsp *to:* dsp and chg	crt
Create Service	*(system default queue)* dsp	crt
Create Service (Replace) *see Note 1*	*(system default queue)* dsp *to:* chg	crt
Delete Authentication Information	dsp and dlt	n/a
Delete Authority Record	*(queue manager object)* chg *see Note 4*	*see Note 4*
Delete Channel	dsp and dlt	n/a
Delete Channel Listener	dsp and dlt	n/a
Delete Client Connection Channel	dsp and dlt	n/a
Delete Namelist	dsp and dlt	n/a
Delete Process	dsp and dlt	n/a
Delete Queue	dsp and dlt	n/a
Delete Service	dsp and dlt	n/a
Inquire Authentication Information	dsp	n/a

Table 2. Object authorities (continued)		
Command	**IBM MQ object authority**	**Class authority (for object type)**
Inquire Authority Records	*see Note 4*	*see Note 4*
Inquire Channel	dsp	n/a
Inquire Channel Listener	dsp	n/a
Inquire Channel Status (for **ChannelType** MQCHT_CLSSDR)	inq	n/a
Inquire Client Connection Channel	dsp	n/a
Inquire Namelist	dsp	n/a
Inquire Process	dsp	n/a
Inquire Queue	dsp	n/a
Inquire Queue Manager	*see note 3*	n/a
Inquire Queue Status	dsp	n/a
Inquire Service	dsp	n/a
Ping Channel	ctrl	n/a
Ping Queue Manager	*see note 3*	n/a
Refresh Queue Manager	(queue manager object) chg	n/a
Refresh Security (for **SecurityType** MQSECTYPE_SSL)	(queue manager object) chg	n/a
Reset Channel	ctrlx	n/a
Reset Queue Manager	(queue manager object) chg	n/a
Reset Queue Statistics	dsp and chg	n/a
Resolve Channel	ctrlx	n/a
Set Authority Record	*(queue manager object)* chg *see Note 4*	*see Note 4*
Start Channel	ctrl	n/a
Stop Channel	ctrl	n/a
Stop Connection	(queue manager object) chg	n/a
Start Listener	ctrl	n/a
Stop Listener	ctrl	n/a
Start Service	ctrl	n/a
Stop Service	ctrl	n/a
Escape	*see Note 2*	*see Note 2*

Notes:

1. This command applies if the object to be replaced does exist, otherwise the authority check is as for Create, or Copy without Replace.

2. The required authority is determined by the MQSC command defined by the escape text, and it is equivalent to one of the previous commands.
3. In order to process any PCF command, the user ID must have dsp authority for the queue manager object on the target system.
4. This PCF command is authorized unless the command server has been started with the -a parameter. By default the command server starts when the queue manager is started, and without the -a parameter. For more information, see Programmable command formats reference.
5. Granting a user ID `chg` authority for a queue manager gives the ability to set authority records for all groups and users. Do not grant this authority to ordinary users or applications.

IBM MQ also supplies some channel security exit points so that you can supply your own user exit programs for security checking. For more information, see Displaying a channel.

Multi Using the MQAI to simplify the use of PCFs

The IBM MQ Administration Interface (MQAI) is a programming interface to IBM MQ that is available on AIX, IBM i, Linux, and Windows. It performs administration tasks on an IBM MQ queue manager using data bags to handle properties (or parameters) of objects in a way that is easier than using Programmable Command Formats (PCFs).

The MQAI performs administration tasks on a queue manager through the use of *data bags*. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using PCFs.

The advantages of using the MQAI are as follows:

Simplify the use of PCF messages
The MQAI is an easier way to administer IBM MQ. If you use the MQAI, you do not have to write your own PCF messages. This avoids the problems associated with complex data structures.

To pass parameters in programs written using MQI calls, the PCF message must contain the command, and details of the string or integer data. To create this configuration manually, you have to add several statements in your program for every structure, and you have to allocate memory space. This task can be long and laborious.

Programs written using the MQAI pass parameters into the appropriate data bag, and you need only one statement for each structure. The use of the MQAI data bags removes the need for you to handle arrays and allocate storage, and provides some degree of isolation from the details of the PCF.

Handle error conditions more easily
It is difficult to get return codes back from PCF commands. The MQAI makes it easier for the program to handle error conditions.

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 IBM MQ built-in data conversion for PCF data. This avoids the need to write data-conversion exits.

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager, using the `mqExecute` call. This call waits for any response messages. The `mqExecute` call handles the exchange with the command server, and returns responses in a *response bag*.

Examples of using the MQAI

The following sample programs demonstrate the use of MQAI to perform the various tasks:

- `amqsaicq.c`: create a local queue.
- `amqsaiem.c`: display events on the screen using a simple event monitor.
- `amqsailq.c`: print a list of all local queues and their current depths.
- `amqsaicl.c`: print a list of all channels and their types.

Building your MQAI application

To build your application using the MQAI, you link to the same libraries as you do for IBM MQ. For information on how to build your IBM MQ applications, see Building a procedural application.

Hints and tips for configuring IBM MQ using MQAI

The MQAI uses PCF messages to send administration commands to the command server rather than dealing directly with the command server itself. Tips for configuring IBM MQ using the MQAI can be found in “Hints and tips for using MQAI to configure IBM MQ” on page 38.

Related reference

IBM MQ Administration Interface reference

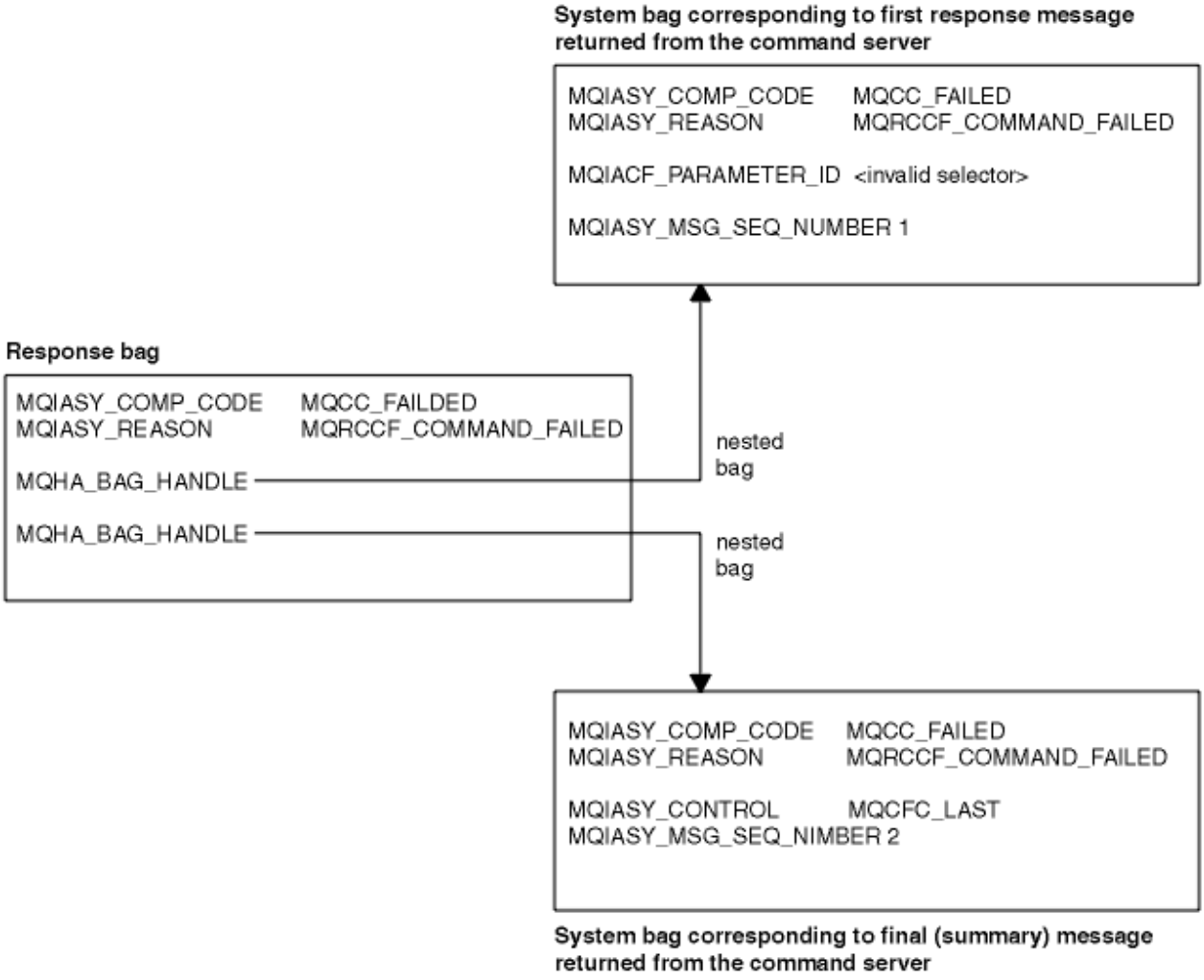
Multi Hints and tips for using MQAI to configure IBM MQ

The IBM MQ Administration Interface (MQAI) 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 IBM MQ using the MQAI.

- Character strings in IBM MQ are blank padded to a fixed length. Using C, null-terminated strings can normally be supplied as input parameters to IBM MQ programming interfaces.
- To clear the value of a string attribute, set it to a single blank character rather than an empty string.
- Consider in advance the attributes that you want to change and inquire on just those attributes.
- Certain attributes cannot be changed, for example a queue name or a channel type. Ensure that you attempt to change only those attributes that can be modified. Refer to the list of required and optional parameters for the specific PCF change object. See Definitions of the Programmable Command Formats.
- 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. 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 by a bag handle.

  For more information on MQExecute, see “Sending administration commands to the qm command server using the mqExecute call” on page 71

  The following diagram shows this scenario:



Multi Advanced MQAI topics

Information on indexing, data conversion and use of message descriptor

Indexing
Indexes are used when replacing or removing existing data items from a bag to preserve insertion order.

Data conversion
The strings contained in an MQAI data bag can be in a variety of coded character sets and these can be converted using the mqSetInteger call.

Use of the message descriptor
The MQAI generates a message descriptor which is set to an initial value when the data bag is created.

Multi *Indexing in the MQAI*

Indexes are used when replacing or removing existing data items from a bag. There are three types of indexing, which allows data items to be retrieved easily.

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 (user or system) to which the item belongs.
- The index relative to all the data items in the bag (user and system).

This allows indexing by user selectors, system selectors, or both as shown in Figure 3 on page 40.

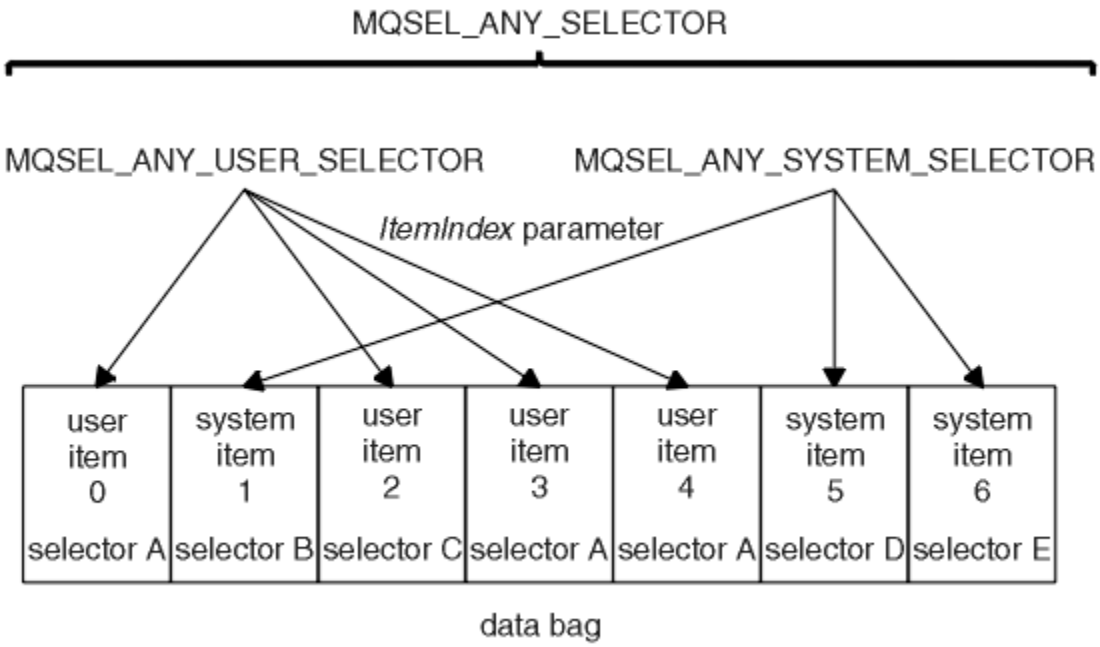


Figure 3. Indexing

In Figure 3 on page 40, user item 3 (selector A) can be referred to by the following index pairs:

- selector A (ItemIndex 1)
- MQSEL_ANY_USER_SELECTOR (ItemIndex 2)
- MQSEL_ANY_SELECTOR (itemIndex 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 68 and “Deleting data items” on page 70.

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.

Multi *Data conversion processing in the MQAI*

The strings contained in an MQAI data bag can be in a variety of coded character sets. These strings can be converted using the mqSetInteger call.

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 3 on page 41 shows the rules applied when converting data bags into messages and vice versa:

Table 3. CCSID processing			
MQAI call	**CCSID**	**Input to call**	**Output to call**
mqBagToBuffer	Bag CCSID (1)	Ignored	Unchanged
mqBagToBuffer	String CCSIDs in bag	Used	Unchanged
mqBagToBuffer	String CCSIDs in buffer	Not applicable	Copied from string CCSIDs in bag
mqBufferToBag	Bag CCSID (1)	Ignored	Unchanged
mqBufferToBag	String CCSIDs in buffer	Used	Unchanged
mqBufferToBagmqBufferToBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in buffer
mqPutBag	MQMD CCSID	Used	Unchanged (2)
mqPutBag	Bag CCSID (1)	Ignored	Unchanged
mqPutBag	String CCSIDs in bag	Used	Unchanged
mqPutBag	String CCSIDs in message sent	Not applicable	Copied from string CCSIDs in bag
mqGetBag	MQMD CCSID	Used for data conversion of message	Set to CCSID of data returned (3)
mqGetBag	Bag CCSID (1)	Ignored	Unchanged
mqGetBag	String CCSIDs in message	Used	Unchanged
mqGetBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in message
mqExecute	Request-bag CCSID	Used for MQMD of request message (4)	Unchanged
mqExecute	Reply-bag CCSID	Used for data conversion of reply message (4)	Set to CCSID of data returned (3)
mqExecute	String CCSIDs in request bag	Used for request message	Unchanged
mqExecute	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.

Related concepts

“Data conversion between coded character sets ” on page 207

Message data in IBM MQ defined formats (also known as built-in formats) can be converted by the queue manager from one coded character set to another, provided that both character sets relate to a single language or a group of similar languages.

"The ccsid_part2.tbl file" on page 209
The `ccsid_part2.tbl` file is used to supply additional CCSID information. The `ccsid_part2.tbl` file replaces the `ccsid.tbl` file that was used before IBM MQ 9.0.

Multi *Use of the message descriptor in the MQAI*

The message descriptor that the MQAI generates is set to an initial value when the data bag is created.

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 4. 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 4 on page 42.

Table 5. Format and MsgType parameters of the MQMD

PCF command type	**Format**	**MsgType**
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_REPORT	MQFMT_ADMIN	MQMT_REPORT
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_TRACE_ROUTE	MQFMT_ADMIN	MQMT_DATAGRAM
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_*	MQFMT_PCF	MQMT_DATAGRAM

Table 5 on page 42 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 6 on page 42.

Table 6. Message descriptor values

Parameter	**Value**
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	see Table 5 on page 42
Expiry	30 seconds (note "1" on page 43)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE

Table 6. Message descriptor values (continued)	
Parameter	**Value**
CodedCharSetId	depends on the bag CCSID (note “2” on page 43)
Format	see Table 5 on page 42
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	see note “3” on page 43
ReplyToQMgr	blank

Notes:

1. This value can be overridden on the mqExecute call by using the **OptionsBag** parameter. For information about this, see mqExecute.
2. See “Data conversion processing in the MQAI” on page 40.
3. Name of the user-specified reply queue or MQAI-generated temporary dynamic queue for messages of type MQMT_REQUEST. Blank otherwise.

Multi Sample C program for creating a local queue (amqsaicq.c)

The sample C program `amqsaicq.c` creates a local queue using the MQAI.

```
/*****************************************************************************/
/*                                                                           */
/* Program name: AMQSAICQ.C                                                  */
/*                                                                           */
/* Description:  Sample C program to create a local queue using the          */
/*               IBM MQ 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, 2026                          */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* 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)    */
/*                                                                  */
/********************************************************************/
/********************************************************************/
/* 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 IBM 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;

}
/********************************************************************/
/*                                                                  */
/* 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);

   /**************************************************************************/
   /* 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,                   /* IBM 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);
      }
   }
   /********************************************************************/
   /* 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);

}
```

Multi Sample C program for displaying events using an event monitor (amqsaiem.c)

The sample C program `amqsaiem.c` demonstrates a basic event monitor using the MQAI.

```
*****************************************************************************/
/*                                                                           */
/* Program name: AMQSAIEM.C                                                  */
/*                                                                           */
/* Description:  Sample C program to demonstrate a basic event monitor       */
/*               using the IBM MQ Admin Interface (MQAI).                    */
/* Licensed Materials - Property of IBM                                      */
/*                                                                           */
/* 63H9336                                                                   */
/* (c) Copyright IBM Corp. 1999, 2026. All Rights Reserved.                  */
/*                                                                           */
/* US Government Users Restricted Rights - Use, duplication or               */
/* disclosure restricted by GSA ADP Schedule Contract with                   */
/* IBM Corp.                                                                 */
/*****************************************************************************/
/*                                                                           */
/* 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                  */
/*            SYSTEM.ADMIN.LOGGER.EVENT      Logger events                   */
/*                                                                           */
/*    To monitor the queue manager event queue or the performance event queue,*/
/*    the attributes of the queue manager need to be changed to enable        */
/*    these events. For more information about this, see Part 1 of the        */
/*    Programmable System Management book. The queue manager attributes can   */
/*    be changed using either MQSC commands or the MQAI interface.            */
/*    Channel events are enabled by default.                                  */
/*                                                                           */
/* Program logic                                                             */
/*    Connect to the Queue Manager.                                          */
/*    Open the requested event queue with a wait interval of 30 seconds.      */
/*    Wait for a message, and when it arrives get the message from the queue  */
/*    and format it into an MQAI bag using 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 the program      */
/*    prints out the contents of the formatted bag.                          */
/*    Loop around to wait for another message until either there is an error  */
/*    or the wait interval of 30 seconds is reached.                         */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* 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                              */

/*****************************************************************************/
/* Macros                                                                    */
/*****************************************************************************/
#if MQAT_DEFAULT == MQAT_WINDOWS_NT
  #define Int64 "I64"
#elif defined(MQ_64_BIT)
  #define Int64 "l"
#else
  #define Int64 "ll"
#endif

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

/*****************************************************************************/
/* Function: main                                                            */
/*****************************************************************************/
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 (times out after 30 secs)\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 messages  */
   /* 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;

}

/*****************************************************************************/
/*                                                                           */
/* 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 for 30 seconds for an event message and then */
/*          terminates if no more messages are available.                  */
/*                                                                         */
/*-------------------------------------------------------------------------*/
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.                   */
   /* 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;

   /*----------------------------------------------------------------------*/
   /* 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 = 30000;               /* 30 second wait for message   */
   gmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING + MQGMO_CONVERT;
   gmo.Version = MQGMO_VERSION_2;          /* Avoid need to reset Message ID */
   gmo.MatchOptions = MQMO_NONE;           /* and Correlation ID after every */
                              /* mqGetBag
   /*----------------------------------------------------------------------*/
   /* If open fails, 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);

     /*******************************************************************/
     /* If get fails, we cannot access the queue and must stop the monitor.   */
     /*******************************************************************/
     if (compCode != MQCC_OK)
     {
        bQueueOK = 0;

        /****************************************************************/
        /* If get fails because no message available then we have timed out, */
        /* so report this, otherwise report an error.                   */
        /****************************************************************/
        if (reason == MQRC_NO_MSG_AVAILABLE)
        {
           printf("No more messages\n");
        }
        else
        {
           CheckCallResult("Get bag", compCode, reason);
        }
     }

     /*******************************************************************/
     /* Event message read - Print the contents of the event bag        */
     /*******************************************************************/
     else
     {
       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);
   }

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


   /*************************************************************************/
   /* Variables                                                             */
   /*************************************************************************/
   MQLONG  itemCount;                     /* Number of items in the bag     */
   MQLONG  itemType;                      /* Type of the item               */
   int     i;                             /* Index of item in the bag       */
   MQCHAR  stringVal[LENGTH+1];           /* Value if item is a string      */
   MQBYTE  byteStringVal[LENGTH];         /* Value if item is a byte string */
   MQLONG  stringLength;                  /* Length of string value         */
   MQLONG  ccsid;                         /* CCSID of string value          */
   MQINT32 iValue;                        /* Value if item is an integer    */
   MQINT64 i64Value;                      /* Value if item is a 64-bit      */
                                          /* 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("
      printf("
      printf("
   }

   /*************************************************************************/
   /* If no errors found, 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++;

       switch(itemType)
       {
       case MQITEM_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 MQITEM_INTEGER64:
            /*********************************************************/
            /* Item is a 64-bit integer. Find its value and display its    */
            /* index, selector and value.                            */
            /*********************************************************/
            mqInquireInteger64(dataBag,       /* Bag handle             */
                               MQSEL_ANY_SELECTOR, /* Allow any selector   */
                               i,             /* Index position in the bag */
                               &i64Value,     /* Returned integer value    */
                               &compCode,     /* Completion code           */
                               &reason);      /* Reason Code               */

            if (compCode != MQCC_OK)
               errors++;
            else
               printf("%.*s  %-2d     %-4d     (%"Int64"d)\n",
                       indent, blanks, i, selector, i64Value);
            break;


      case MQITEM_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  */
                            &stringLength,    /* Actual length of string   */
                            &ccsid,           /* Coded character set ID    */
                            &compCode,        /* Completion code           */
                            &reason);         /* Reason Code               */

            /*********************************************************/
            /* The call can return a warning if the string is too long for */
            /* the output buffer and has been truncated, so only check     */
```

```
            /* explicitly for call failure.                               */
            /************************************************************/
            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;

         case MQITEM_BYTE_STRING:
            /************************************************************/
            /* Item is a byte string. Obtain the byte string in a buffer,  */
            /* prepare the byte string for displaying and display the      */
            /* index, selector and string.                                 */
            /************************************************************/
            mqInquireByteString(dataBag,      /* Bag handle               */
                                MQSEL_ANY_SELECTOR, /* Allow any selector  */
                                i,            /* Index position in the bag */
                                LENGTH,       /* Maximum length of buffer  */
                                byteStringVal, /* Buffer to receive string */
                                &stringLength, /* Actual length of string  */
                                &compCode,    /* Completion code           */
                                &reason);     /* Reason Code

            /************************************************************/
            /* The call can return a warning if the string is too long for */
            /* the output buffer and has been truncated, so only check     */
            /* explicitly for call failure.                               */
            /************************************************************/
            if (compCode == MQCC_FAILED)
                errors++;
            else
            {
               printf("%.*s  %-2d     %-4d     X'",
                      indent, blanks, i, selector);

               for (i = 0 ; i < stringLength ; i++)
                  printf("

               printf("'\n");
            }
            break;

         case MQITEM_BAG:
            /************************************************************/
            /* Item is an embedded bag handle, so call the PrintBagContents*/
            /* 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);
               if (selector == MQHA_BAG_HANDLE)
                  printf("
               else
                  printf("
               PrintBagContents(bagHandle, indent+INDENT);
            }
            break;

         default:
```

```
                printf("
            }
        }
    }
    return errors;
}
```

Multi Sample C program for inquiring about channel objects (amqsaicl.c)

The sample C program amqsaicl.c inquires channel objects using the MQAI.

```
/********************************************************************************/
/*                                                                              */
/* Program name: AMQSAICL.C                                                     */
/*                                                                              */
/* Description:  Sample C program to inquire channel objects                    */
/*               using the IBM MQ Administration Interface (MQAI)               */
/*                                                                              */
/* <N_OCO_COPYRIGHT>                                                            */
/* Licensed Materials - Property of IBM                                         */
/*                                                                              */
/* 63H9336                                                                      */
/* (c) Copyright IBM Corp. 2008, 2026. All Rights Reserved.                     */
/*                                                                              */
/* US Government Users Restricted Rights - Use, duplication or                  */
/* disclosure restricted by GSA ADP Schedule Contract with                      */
/* IBM Corp.                                                                    */
/* <NOC_COPYRIGHT>                                                              */
/********************************************************************************/
/*                                                                              */
/* Function:                                                                    */
/*    AMQSAICL is a sample C program that demonstrates how to inquire           */
/*    attributes of the local queue manager using the MQAI interface. In        */
/*    particular, it inquires all channels and their types.                     */
/*                                                                              */
/*     - A PCF command is built from items placed into an MQAI administration   */
/*       bag.                                                                   */
/*       These are:-                                                            */
/*            - The generic channel name "*"                                    */
/*            - The attributes to be inquired. In this sample we just want      */
/*              name and type attributes                                        */
/*                                                                              */
/*     - The mqExecute MQCMD_INQUIRE_CHANNEL call is executed.                  */
/*       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_CHANNEL 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 has been          */
/*       embedded in the response bag to the mqExecute call.                    */
/*                                                                              */
/* Note: The command server must be running.                                    */
/*                                                                              */
/********************************************************************************/
/*                                                                              */
/* AMQSAICL has 2 parameter - the queue manager name (optional)                 */
/*                          - output file (optional) default varies             */
/********************************************************************************/

/********************************************************************************/
/* Includes                                                                     */
/********************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#if (MQAT_DEFAULT == MQAT_OS400)
#include <recio.h>
#endif

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

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

/*****************************************************************************/
/* DataTypes                                                                 */
/*****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
typedef _RFILE OUTFILEHDL;
#else
typedef FILE OUTFILEHDL;
#endif

/*****************************************************************************/
/* Constants                                                                 */
/*****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "*SDR     ",    /* MQCHT_SENDER    */
  "*SVR     ",    /* MQCHT_SERVER    */
  "*RCVR    ",    /* MQCHT_RECEIVER  */
  "*RQSTR   ",    /* MQCHT_REQUESTER */
  "*ALL     ",    /* MQCHT_ALL       */
  "*CLTCN   ",    /* MQCHT_CLNTCONN  */
  "*SVRCONN ",    /* MQCHT_SVRCONN   */
  "*CLUSRCVR",    /* MQCHT_CLUSRCVR  */
  "*CLUSSDR "     /* MQCHT_CLUSSDR   */
};
#else
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "sdr      ",    /* MQCHT_SENDER    */
  "svr      ",    /* MQCHT_SERVER    */
  "rcvr     ",    /* MQCHT_RECEIVER  */
  "rqstr    ",    /* MQCHT_REQUESTER */
  "all      ",    /* MQCHT_ALL       */
  "cltconn  ",    /* MQCHT_CLNTCONN  */
  "svrcn    ",    /* MQCHT_SVRCONN   */
  "clusrcvr ",    /* MQCHT_CLUSRCVR  */
  "clussdr  "     /* MQCHT_CLUSSDR   */
};
#endif

/*****************************************************************************/
/* Macros                                                                    */
/*****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
  #define OUTFILE "QTEMP/AMQSAICL(AMQSAICL)"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = _Ropen((fname),"wr, rtncode=Y");
  #define CLOSEOUTFILE(hdl) \
    _Rclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    _Rwrite((hdl),(buf),(buflen));

#elif (MQAT_DEFAULT == MQAT_UNIX)
  #define OUTFILE "/tmp/amqsaicl.txt"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));

#else
  #define OUTFILE "amqsaicl.txt"
  #define OPENOUTFILE(fname) \
    fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));
```

```
#endif

#define ChlType2String(t) ChlTypeMap[(t)-1].name

/********************************************************************************/
/* 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 cAttrsBag;                       /* bag containing chl attributes   */
   MQHBAG errorBag;                        /* bag containing cmd server error */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */
   MQLONG chlNameLength;                   /* Actual length of chl name       */
   MQLONG chlType;                         /* Channel type                    */
   MQLONG i;                               /* loop counter                    */
   MQLONG numberOfBags;                    /* number of bags in response bag  */
   MQCHAR chlName[MQ_OBJECT_NAME_LENGTH+1];/* name of chl extracted from bag  */
   MQCHAR OutputBuffer[100];               /* output data buffer              */
   OUTFILEHDL *outfp = NULL;               /* output file handle              */

   /*****************************************************************************/
   /* Connect to the queue manager                                              */
   /*****************************************************************************/
   if (argc &gt; 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);
   }

   /*****************************************************************************/
   /* Open the output file                                                      */
   /*****************************************************************************/
   if (argc &gt; 2)
   {
     OPENOUTFILE(outfp, argv[2]);
   }
   else
   {
     OPENOUTFILE(outfp, OUTFILE);
   }

   if(outfp == NULL)
   {
     printf("Could not open output file.\n");
     goto MOD_EXIT;
   }
   /*****************************************************************************/
   /* Create an admin bag for the mqExecute call                                */
   /*****************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &adminBag;, &compCode;, &reason;);
   CheckCallResult("Create admin bag", compCode, reason);

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

   /*****************************************************************************/
   /* Put the generic channel name into the admin bag                           */
   /*****************************************************************************/
   mqAddString(adminBag, MQCACH_CHANNEL_NAME, MQBL_NULL_TERMINATED, "*",
               &compCode;, &reason;);
   CheckCallResult("Add channel name", compCode, reason);
```

```
/*****************************************************************/
/* Put the channel type into the admin bag                       */
/*****************************************************************/
mqAddInteger(adminBag, MQIACH_CHANNEL_TYPE, MQCHT_ALL, &compCode;, &reason;);
CheckCallResult("Add channel type", compCode, reason);

/*****************************************************************/
/* Add an inquiry for various attributes                         */
/*****************************************************************/
mqAddInquiry(adminBag, MQIACH_CHANNEL_TYPE, &compCode;, &reason;);
CheckCallResult("Add inquiry", compCode, reason);

/*****************************************************************/
/* Send the command to find all the channel names and channel types.  */
/* 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_CHANNEL,  /* 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");
   goto MOD_EXIT;
}

/*****************************************************************/
/* Check the result from mqExecute call. If successful find the channel    */
/* types for all the channels. 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 channel are in separate bags. */
  /***************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags;,
               &compCode;, &reason;);
  CheckCallResult("Count number of bag handles", compCode, reason);

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

    /*************************************************************/
    /* Get the channel name out of the channel attributes bag              */
    /*************************************************************/
    mqInquireString(cAttrsBag, MQCACH_CHANNEL_NAME, 0, MQ_OBJECT_NAME_LENGTH,
                    chlName, &chlNameLength, NULL, &compCode, &reason);
    CheckCallResult("Get channel name", compCode, reason);

    /*************************************************************/
    /* Get the channel type out of the channel attributes bag              */
    /*************************************************************/

 mqInquireInteger(cAttrsBag, MQIACH_CHANNEL_TYPE, MQIND_NONE, &chlType,
                  &compCode, &reason);
    CheckCallResult("Get type", compCode, reason);

    /*************************************************************/
    /* Use mqTrim to prepare the channel name for printing.              */
    /* Print the result.                                         */
    /*************************************************************/
```

```
        mqTrim(MQ_CHANNEL_NAME_LENGTH, chlName, chlName, &compCode, &reason);
        sprintf(OutputBuffer, "%-20s%-9s", chlName, ChlType2String(chlType));
        WRITEOUTFILE(outfp,OutputBuffer,29)
      }
   }

   else                                              /* Failed mqExecute     */
   {
     printf("Call to get channel attributes failed: Cc = %ld : Rc = %ld\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: Cc = %ld : Rc = %ld\n",
                mqExecuteCC, mqExecuteRC);
     }
   }

MOD_EXIT:
   /*******************************************************************/
   /* 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);
   }

   /*******************************************************************/
   /* Close the output file if open                                   */
   /*******************************************************************/
   if(outfp != NULL)
      CLOSEOUTFILE(outfp);

   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 = %ld : Reason = %ld\n", callText,
                cc, rc);
}
```

Multi Sample C program for inquiring about queues and printing information (amqsailq.c)

The sample C program `amqsailq.c` inquires the current depth of the local queues using the MQAI.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Program name: AMQSAILQ.C                                                */
/*                                                                         */
/* Description:  Sample C program to inquire the current depth of the local*/
/*               queues using the IBM MQ 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, 2026                        */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* 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)               */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/* 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 IBM 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);
   /*************************************************************************/
   /* 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,                    /* IBM 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);

  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);
    }
  }

  /********************************************************************/
  /* 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);
}
```

Multi Data bags and the MQAI

A data bag is a means of handling properties or parameters of objects using the IBM MQ Administration Interface (MQAI).

Data Bags

- The data bag contains zero or more *data items*. These data items 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 64-bit integer, an integer filter, a string, a string filter, a byte string, a byte string filter, or a handle of another bag. Data items are described in details in "Types of data item available in the MQAI" on page 65

  There are two types of selector; *user selectors* and *system selectors*. These are described in MQAI Selectors. 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 in the MQAI" on page 39.

  A hierarchy of these concepts is shown in Figure 1.

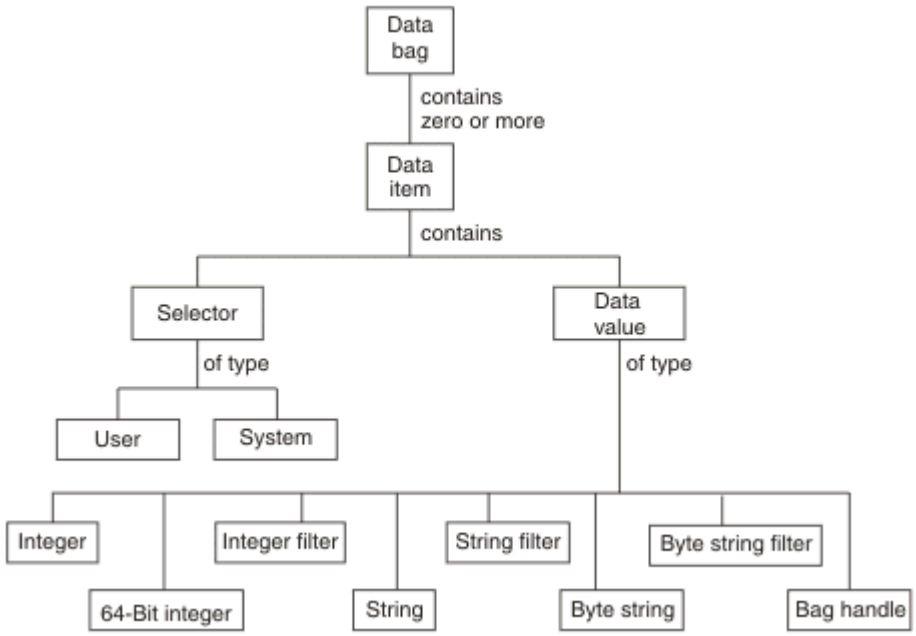


Figure 4. Hierarchy of MQAI concepts

The hierarchy has been explained in a previous paragraph.

Types of data bag

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

user bag
A simple bag used for user data.

administration bag
A bag created for data used to administer IBM MQ objects by sending administration messages to a command server. The administration bag automatically implies certain options as described in “Creating and deleting data bags” on page 64.

command bag
A bag also created for commands for administering IBM MQ objects. However, unlike the administration bag, the command bag does not automatically imply certain options although these options are available. For more information about options, see “Creating and deleting data bags” on page 64.

group bag
A bag used to hold a set of grouped data items. Group bags cannot be used for administering IBM MQ objects.

In addition, the **system bag** is 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.

Using Data Bags The different ways of using data bags are listed in this topic:

Using Data Bags

The different ways of using data bags are shown in the following list:

- You can create and delete data bags “Creating and deleting data bags” on page 64.
- You can send data between applications using data bags “Putting and receiving data bags using the MQAI” on page 65.
- You can add data items to data bags “Adding data items to bags with the MQAI” on page 66.
- You can add an inquiry command within a data bag “Adding an inquiry command to a bag” on page 67.
- You can inquire within data bags “Inquiring within data bags” on page 67.
- You can count data items within a data bag “Counting data items” on page 70.
- You can change information within a data bag “Changing information within a bag” on page 68.
- You can clear a data bag “Clearing a bag using the mqClearBag call” on page 69.
- You can truncate a data bag “Truncating a bag using the mqTruncateBag call” on page 69.
- You can convert bags and buffers “Converting bags and buffers” on page 69.

Creating and deleting data bags

Creating data bags

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

The **Options** parameter of the MQCreateBag call lets you choose whether to create a user bag, a command bag, a group bag, or an administration bag.

To create a user bag, a command bag, or a group bag, you can choose one or more further options 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. For more information on data items, see “Types of data item available in the MQAI” on page 65.
- 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.

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 qm command server using the mqExecute call” on page 71. User code cannot delete a system bag.

For a full description of the mqDeleteBag call, see mqDeleteBag.

Multi *Putting and receiving data bags using the MQAI*

Data can also be sent between applications by putting and getting data bags using the mqPutBag and mqGetBag calls. This lets the IBM MQ Administration Interface (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.

For more information on sending and receiving PCF messages in a specific queue, see “Sending and receiving PCF messages in a specified queue” on page 28

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.

Multi *Types of data item available in the MQAI*

Data items are used by the IBM MQ Administration Interface (MQAI) to populate data bags when they are created. These data items can be user or system items.

These 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 and the MQAI” on page 66.

Types of Data Items

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.

The data item can either be integer or character-string items. Here are the types of data item available within the MQAI:

- Integer
- 64-bit integer
- Integer filter
- Character-string
- String filter
- Byte string
- Byte string filter
- Bag handle

Using Data Items

These are the following ways of using data items:

- “Counting data items” on page 70.
- “Deleting data items” on page 70.
- “Adding data items to bags with the MQAI” on page 66.
- “Filtering and querying data items” on page 67.

Multi *System items and the MQAI*

System items can be used by the IBM MQ Administration Interface (MQAI) 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 MQAI Selectors.

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 68), but the bag-creation options cannot be changed by the user. You cannot delete system items. (See “Deleting data items” on page 70.)

Multi *Adding data items to bags with the MQAI*

When a data bag is created using the IBM MQ Administration Interface (MQAI), you can populate it with data items. These data items can be user or system items.

For more information about data items, see “Types of data item available in the MQAI” on page 65.

The MQAI lets you add integer items, 64-bit integer items, integer filter items, character-string items, string filter, byte string items, and byte string filter items to bags and this is shown in Figure 5 on page 66. 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.

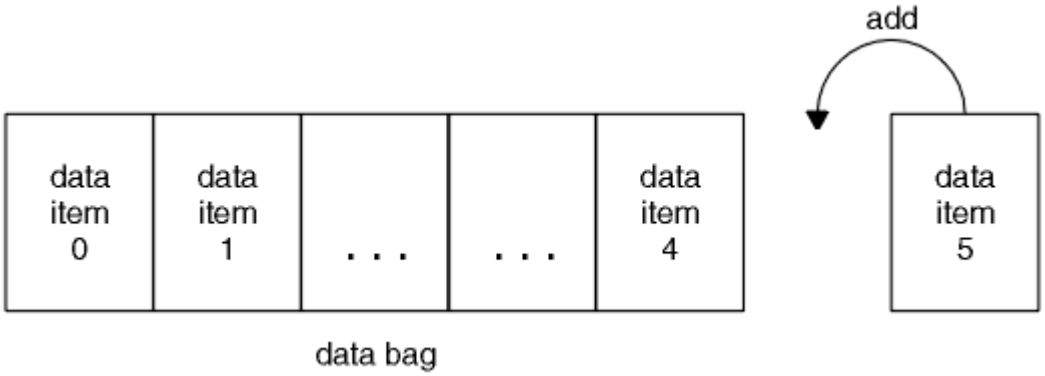


Figure 5. Adding data items

Add data items to a bag using the mqAdd* calls:

- To add integer items, use the mqAddInteger call as described in mqAddInteger
- To add 64-bit integer items, use the mqAddInteger64 call as described in mqAddInteger64
- To add integer filter items, use the mqAddIntegerFilter call as described in mqAddIntegerFilter
- To add character-string items, use the mqAddString call as described in mqAddString

- To add string filter items, use the mqAddStringFilter call as described in mqAddStringFilter
- To add byte string items, use the mqAddByteString call as described in mqAddByteString
- To add byte string filter items, use the mqAddByteStringFilter call as described in mqAddByteStringFilter

For more information on adding data items to a bag, see "System items and the MQAI" on page 66.

Multi *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 on which you want to inquire from IBM MQ.

For a full description of the mqAddInquiry call, see mqAddInquiry.

Multi *Filtering and querying data items*

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

- 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) about which you are inquiring.

  Other attributes that can be filtered correspond to the PCF Inquire* commands that can be found in "Introduction to IBM MQ Programmable Command Formats" on page 25. For example, to inquire about the attributes of a channel, see the Inquire Channel command in this product documentation. The "Required parameters" and "Optional parameters" of the Inquire Channel command identify the selectors that you can use for filtering.
- You can ***query*** particular attributes of an object using the mqAddInquiry call. This specifies the selector in which you are interested. 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, ...)
```

Multi *Inquiring within data bags*

You can inquire about:

- The value of an integer item using the mqInquireInteger call. See mqInquireInteger.
- The value of a 64-bit integer item using the mqInquireInteger64 call. See mqInquireInteger64.
- The value of an integer filter item using the mqInquireIntegerFilter call. See mqInquireIntegerFilter.
- The value of a character-string item using the mqInquireString call. See mqInquireString.
- The value of a string filter item using the mqInquireStringFilter call. See mqInquireStringFilter.
- The value of a byte string item using the mqInquireByteString call. See mqInquireByteString.

- The value of a byte string filter item using the mqInquireByteStringFilter call. See mqInquireByteStringFilter.
- The value of a bag handle using the mqInquireBag call. See mqInquireBag.

You can also inquire about the type (integer, 64-bit integer, integer filter, character string, string filter, byte string, byte string filter or bag handle) of a specific item using the mqInquireItemInfo call. See mqInquireItemInfo.

Multi *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 6 on page 68).

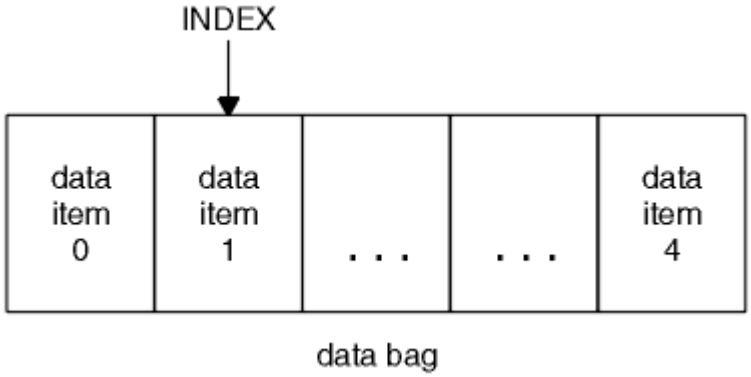


Figure 6. Modifying a single data item

2. Delete all existing occurrences of the specified selector and add a new occurrence to the end of the bag. (See Figure 7 on page 68.) A special index value allows ***all*** instances of a parameter to be replaced.

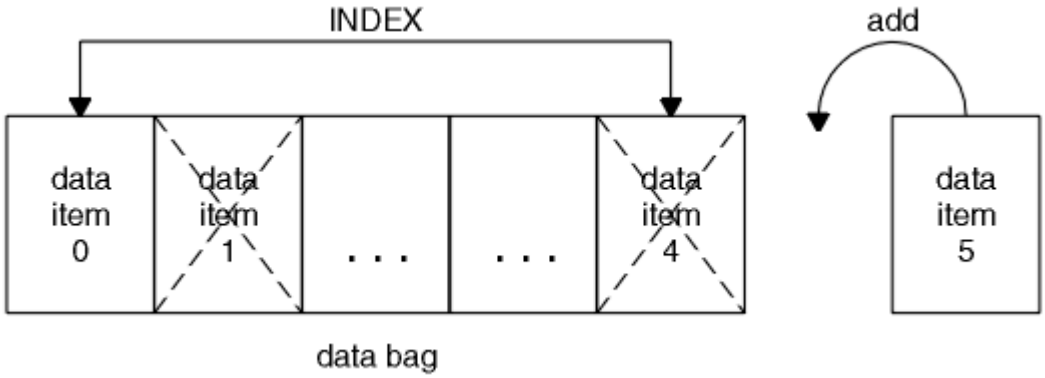


Figure 7. Modifying all data items

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. The mqSetInteger64 call lets you modify 64-bit integer items. The mqSetIntegerFilter call lets you modify integer filter items. The mqSetString call lets you modify character-string items. The mqSetStringFilter call lets you modify string filter items. The mqSetByteString call lets you modify byte string items. The mqSetByteStringFilter call lets you modify byte string filter 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

- mqSetInteger64
- mqSetIntegerFilter
- mqSetString
- mqSetStringFilter
- mqSetByteString
- mqSetByteStringFilter

Multi *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.

Multi *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.

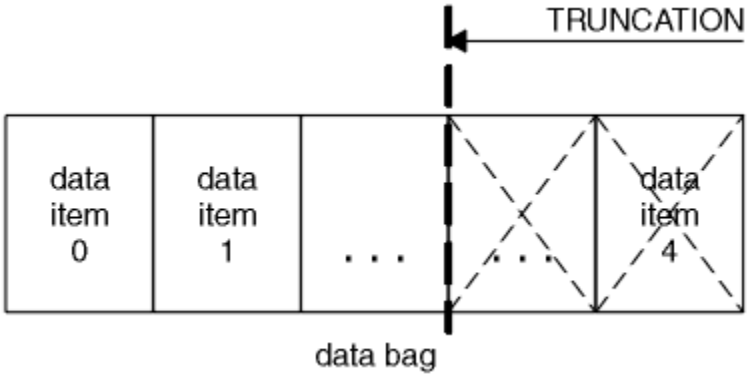


Figure 8. Truncating a bag

For a full description of the mqTruncateBag call, see mqTruncateBag.

Multi *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 Figure 9 on page 69. For a full description of the mqBagToBuffer call, see mqBagToBuffer.

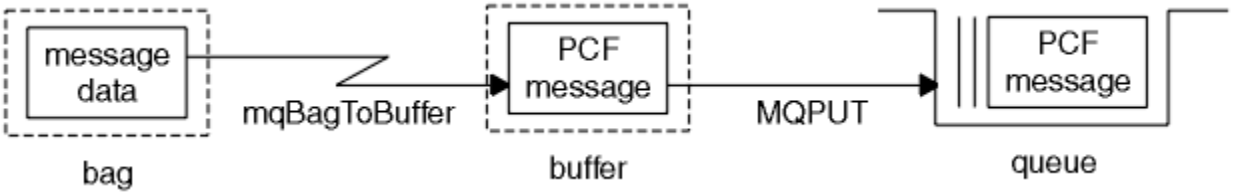


Figure 9. Converting bags to PCF messages

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 Figure 10 on page 70. For a full description of the mqBufferToBag call, see mqBufferToBag.

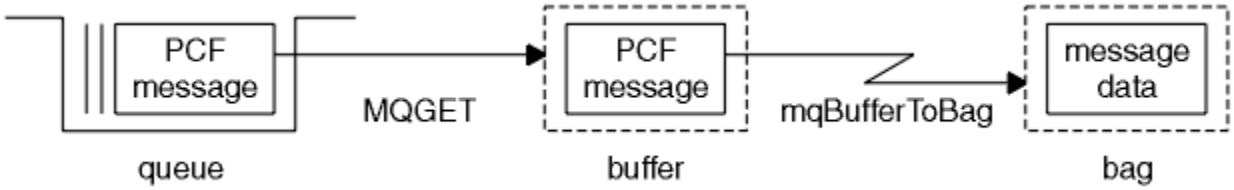


Figure 10. Converting PCF messages to bag form

Multi *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 might be fewer unique selectors in the bag than data items.

For a full description of the mqCountItems call, see mqCountItems.

Multi *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. For detailed information, see "Deleting data items from a bag using the mqDeleteItem call" on page 70.
- Delete all user items from a bag, that is, clear a bag. For detailed information see "Clearing a bag using the mqClearBag call" on page 69.
- Delete user items from the end of a bag, that is, truncate a bag. For detailed information, see "Truncating a bag using the mqTruncateBag call" on page 69.

Multi *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 11 on page 70.)

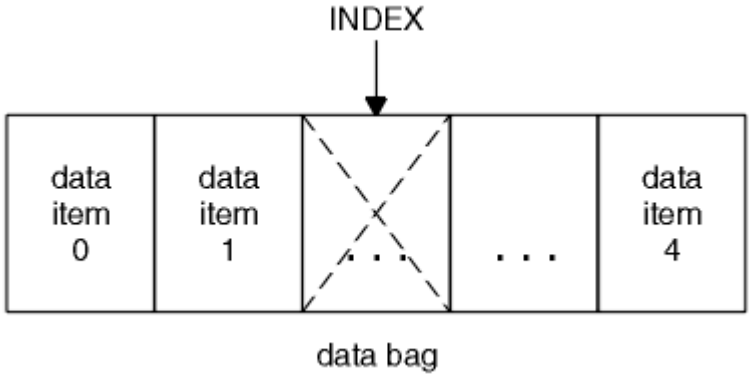


Figure 11. Deleting a single data item

or

2. All occurrences of the specified selector. (See Figure 12 on page 71.)

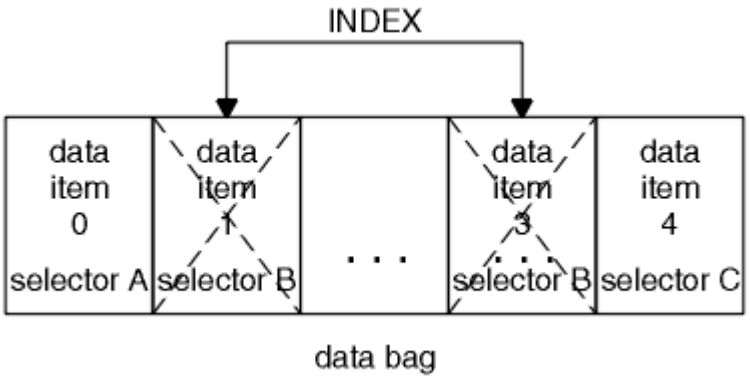


Figure 12. Deleting all data items

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.

Multi Sending administration commands to the qm command server using the mqExecute call

When a data bag has been created and populated, an administrative command message can be sent to the command server of a queue manager using the mqExecute call. This handles the exchange with the command server and returns responses in a bag.

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager. The easiest way to do this is by using the mqExecute call. 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 IBM MQ 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 13 on page 72.

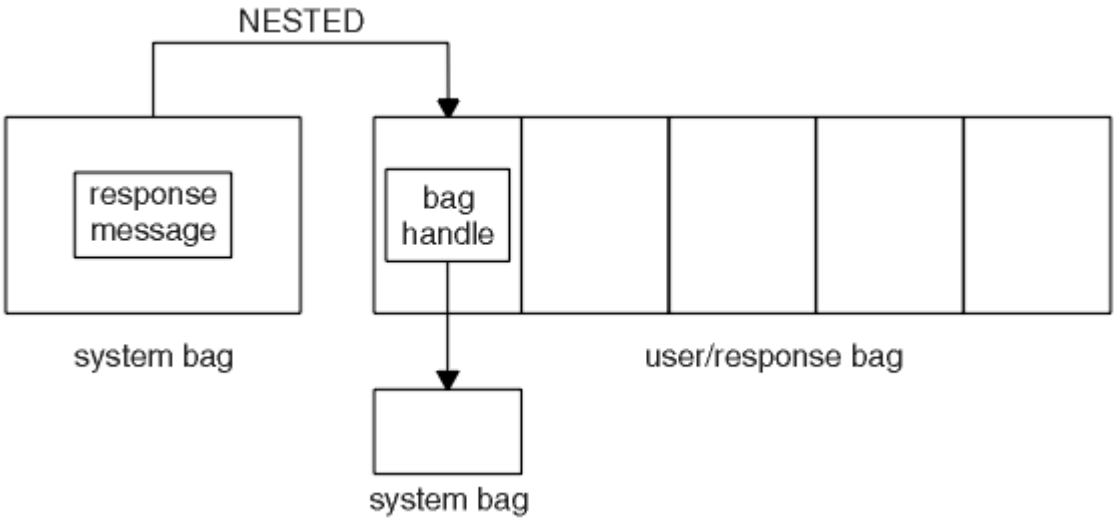


Figure 13. Nesting

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 handles 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.

For examples uses of the mqExecute call, see Example code

Administration using the REST API

You can use the administrative REST API to administer IBM MQ objects, such as queue managers and queues, and Managed File Transfer agents and transfers. Information is sent to, and received from,

the administrative REST API in JSON format. These RESTful APIs can help you to embed IBM MQ administration into popular DevOps and automation tooling.

Before you begin

Note: You cannot use the administrative REST API with the stand-alone IBM MQ Web Server. For more information about the installation options for the IBM MQ component that runs the administrative REST API, see The IBM MQ Console and REST API.

For reference information about the available REST resources, see The administrative REST API reference.

Procedure

- "Getting started with the administrative REST API" on page 73
- "Using the administrative REST API" on page 76
- "Remote administration using the REST API" on page 78
- "REST API time stamps" on page 82
- "REST API error handling" on page 82
- "REST API discovery" on page 85
- "REST API national language support" on page 86

Getting started with the administrative REST API

Get started quickly with the administrative REST API and try out a few example requests by using cURL to create, update, view, and delete a queue.

Before you begin

To get you started with using the administrative REST API, the examples in this task have the following requirements:

- The examples use cURL to make REST requests to display information about queue managers on the system, and to create a queue, update, view, and delete a queue. Therefore, to complete this task you need cURL installed on your system.
- To complete this task, you must be a user with certain privileges so that you can use the **dspmqweb** command:
  - z/OS On z/OS, you must have authority to run the **dspmqweb** command, and write access to the `mqwebuser.xml` file.
  - Multi On all other operating systems, you must be a privileged user.

IBM i On IBM i, the commands should be running in QSHELL.

Procedure

1. Ensure that you configured the mqweb server for use by the administrative REST API, the administrative REST API for MFT, the messaging REST API, or IBM MQ Console.

   For more information about configuring the mqweb server with a basic registry, see Basic configuration for the mqweb server.

2. z/OS

   On z/OS, set the WLP_USER_DIR environment variable so that you can use the **dspmqweb** command. Set the variable to point to your mqweb server configuration by entering the following command:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

where *WLP_user_directory* is the name of the directory that is passed to `crtmqweb`. For example:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

For more information, see Creating the mqweb server.

3. Determine the REST API URL by entering the following command:

```
dspmqweb status
```

The examples in the following steps assume that your REST API URL is the default URL `https://localhost:9443/ibmmq/rest/v3/`. If your URL is different than the default, substitute your URL in the following steps.

4. Try out a GET request on the `qmgr` resource by using basic authentication with the `mqadmin` user:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/qmgr -X GET -u mqadmin:mqadmin
```

5. Create, display, alter, and delete a queue by using the `mqsc` resource:

This example uses a queue manager `QM1`. Either create a queue manager with the same name, or substitute an existing queue manager on your system.

a) Make a POST request on the `mqsc` resource to create the local queue:

In the body of the request, the name of the new queue is set to `Q1`. Basic authentication is used, and an `ibm-mq-rest-csrf-token` HTTP header with an arbitrary value is set in the cURL REST request. This additional header is required for POST, PATCH, and DELETE requests:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
data "{\"type\": \"runCommandJSON\", \"command\": \"define\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\"}"
```

b) Make a POST request on the `mqsc` resource to view the local queue created in step “5.a” on page 74:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
data "{\"type\": \"runCommandJSON\", \"command\": \"display\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\"}"
```

c) Make a POST request on the `mqsc` resource to resource to update the description of the queue:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
data "{\"type\": \"runCommandJSON\", \"command\": \"alter\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\", \"parameters\": {\"descr\": \"new description\" }}"
```

d) Make a POST request on the `mqsc` resource to view the new queue description. Specify the **`responseParameters`** attribute in the request body so that the response includes the description field:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
data "{\"type\": \"runCommandJSON\", \"command\": \"display\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\", \"responseParameters\" : [\"descr\"]}"
```

e) Make a POST request on the `mqsc` resource to delete the queue:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
data "{\"type\": \"runCommandJSON\", \"command\": \"delete\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\"}"
```

f) Make a POST request on the `mqsc` resource to prove that the queue is deleted:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM1/mqsc -X POST -u
mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
```

```
data "{\"type\": \"runCommandJSON\", \"command\": \"display\", \"qualifier\": \"qlocal\",
\"name\": \"Q1\"}"
```

What to do next

- The examples use basic authentication to secure the request. You can use token-based authentication or client-based authentication instead. For more information, see Using client certificate authentication with the REST API and IBM MQ Console, and Using token-based authentication with the REST API.
- Learn more about using the administrative REST API and constructing URLs with query parameters: “Using the administrative REST API” on page 76.
- Browse the reference information for the available administrative REST API resources and all the available optional query parameters: administrative REST API reference.
- Learn how to use the administrative REST API to administer IBM MQ objects on remote systems: “Remote administration using the REST API” on page 78.
- Learn how to use the administrative REST API with MFT:“Getting started with the REST API for MFT” on page 75.
- Discover the messaging REST API, a RESTful interface for IBM MQ messaging: Messaging using the REST API.
- Discover the IBM MQ Console, a browser-based GUI: “Administration using the IBM MQ Console” on page 89.

Getting started with the REST API for MFT

Get started quickly with the administrative REST API for Managed File Transfer and try out a few example requests to view the MFT agent status, and to view a list of transfers.

Before you begin

- The examples use cURL to send REST requests to view a list of transfers and view MFT agent status. Therefore, to complete this task you need cURL installed on your system.
- To complete this task, you must be a user with certain privileges so that you can use the **dspmqweb** command:
  - z/OS On z/OS, you must have authority to run the **dspmqweb** command, and write access to the `mqwebuser.xml` file.
  - Multi On all other operating systems, you must be a privileged user.

Procedure

1. Ensure that the mqweb server is configured for the administrative REST API for MFT:
   - Ensure that you configured the mqweb server for use by the administrative REST API, the administrative REST API for MFT, the messaging REST API, or IBM MQ Console. For more information about configuring the mqweb server with a basic registry, see Basic configuration for the mqweb server.
   - If the mqweb server is configured, ensure that step 8 of Basic configuration for the mqweb server was completed to enable the administrative REST API for MFT.
2. z/OS
   On z/OS, set the WLP_USER_DIR environment variable so that you can use the **dspmqweb** command. Set the variable to point to your mqweb server configuration by entering the following command:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

where *WLP_user_directory* is the name of the directory that is passed to `crtmqweb`. For example:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

For more information, see Creating the mqweb server.

3. Determine the REST API URL by entering the following command:

```
dspmqweb status
```

The examples in the following steps assume that your REST API URL is the default URL `https://localhost:9443/ibmmq/rest/v3/`. If your URL is different than the default, substitute your URL in the following steps.

4. Make a GET request on the `agent` resource to return basic details about all agents, including the name, type and state:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/mft/agent/ -X GET -u mftadmin:mftadmin
```

5. Create some transfers to display by using the **fteCreateTransfer** command.

The mqweb server caches information about transfers and returns this information when a request is made. This cache is reset when the mqweb server is restarted. You can see whether the server was restarted by viewing the `console.log` and `messages.log` files, or on z/OS, looking at the output from the started task.

6. Make a GET request on the `transfer` resource to return details of up to four transfers that were made since the mqweb server was started:

```
curl -k https://localhost:9443/ibmmq/rest/v3/admin/mft/transfer?limit=4 -X GET -u
mftadmin:mftadmin
```

What to do next

- The examples use basic authentication to secure the request. You can use token-based authentication or client-based authentication instead. For more information, see Using token-based authentication with the REST API, and Using client certificate authentication with the REST API and IBM MQ Console.
- Learn more about using the administrative REST API and constructing URLs with query parameters: “Using the administrative REST API” on page 76.
- Browse the reference information for the available administrative REST API for MFT resources and all the available optional query parameters: administrative REST API reference.
- Discover the messaging REST API, a RESTful interface for IBM MQ messaging: Messaging using the REST API.
- Discover the IBM MQ Console, a browser-based GUI: “Administration using the IBM MQ Console” on page 89.

Using the administrative REST API

When you use the administrative REST API, you invoke HTTP methods on URLs that represent the various IBM MQ objects, such as queue managers or queues. The HTTP method, for example POST, represents the type of action to be performed on the object that is represented by the URL. Further information about the action might be provided in JSON as part of the payload of the HTTP method, or encoded in query parameters. Information about the result of performing the action might be returned as the body of the HTTP response.

Before you begin

Consider these things before you use the administrative REST API:

- You must authenticate with the mqweb server in order to use the administrative REST API. You can authenticate by using HTTP basic authentication, client certificate authentication, or token based

authentication. For more information about how to use these authentication methods, see IBM MQ Console and REST API security.

- The REST API is case-sensitive. For example, an HTTP GET on the following URL does not display information if the queue manager is called `qmgr1`.

```
/ibmmq/rest/v3/admin/qmgr/QMGR1
```

- Not all of the characters that can be used in IBM MQ object names can be directly encoded in a URL. To encode these characters correctly, you must use the appropriate URL encoding:
  - A forward slash, /, must be encoded as %2F.
  - A percent sign, %, must be encoded as %25.
- Due to the behavior of some browsers, do not name objects by using only period or forward slash characters.

About this task

When you use the REST API to perform an action on an object, you first need to construct a URL to represent that object. Each URL starts with a prefix, which describes which host name and port to send the request to. The rest of the URL describes a particular object, or set of objects, known as a resource.

The action that is to be performed on the resource defines whether the URL needs query parameters or not. It also defines the HTTP method that is used, and whether additional information is sent to the URL, or returned from it, in JSON form. The additional information might form part of the HTTP request, or be returned as part of the HTTP response.

After you construct the URL, and create an optional JSON payload for sending in the HTTP request, you can send the HTTP request to IBM MQ. You can send the request by using the HTTP implementation that is built into the programming language of your choice. You can also send the requests by using command line tools such as cURL, or a web browser, or web browser add-on.

Important: You must, as a minimum, carry out steps “1.a” on page 77 and “1.b” on page 77.

Procedure

1. Construct the URL:
   a) Determine the prefix URL by entering the following command:

   ```
   dspmqweb status
   ```

   The URL that you want to use includes the `/ibmmq/rest/` phrase.

   b) Add the resource to the URL path.

   The following IBM MQ resources are available:

   - /admin/installation
   - /admin/qmgr
   - /admin/queue
   - /admin/subscription
   - /admin/channel
   - /action/qmgr/{qmgrname}/mqsc

   The following Managed File Transfer resources are available:

   - /admin/agent
   - /admin/transfer
   - /admin/monitor

For example, to interact with queue managers, add `/qmgr` to the prefix URL to create the following URL:

```
https://localhost:9443/ibmmq/rest/v3/admin/qmgr
```

c) Optional: Add any additional optional path segments to the URL.

In the reference information for each object type, the optional segments can be identified in the URL by the braces that surround it { }.

For example, add the queue manager name `QM1` to the URL to create the following URL:

```
https://localhost:9443/ibmmq/rest/v3/admin/qmgr/QM1
```

d) Optional: Add an optional query parameter to the URL.

Add a question mark, ?, variable name, equal sign =, and a value or list of values to the URL.

For example, to request all attributes of queue manager QM1, create the following URL:

```
https://localhost:9443/ibmmq/rest/v3/admin/qmgr/QM1?attributes=*
```

e) Add further optional query parameters to the URL.

Add an ampersand, &, to the URL, and then repeat step d.

2. Invoke the relevant HTTP method on the URL. Specify any optional JSON payload, and provide the appropriate security credentials to authenticate. For example:
   - Use the HTTP/REST implementation of your chosen programming language.
   - Use a tool such as a REST client browser add-on or cURL.

Remote administration using the REST API

You can use the REST API to administer remote queue managers, and the IBM MQ objects that are associated with those queue managers. This remote administration includes queue managers that are on the same system, but not in the same IBM MQ installation as the mqweb server. Therefore, you can use the REST API to administer your entire IBM MQ network with only one installation that runs the mqweb server. To administer remote queue managers, you must configure the administrative REST API gateway so that at least one queue manager in the same installation as the mqweb server acts as a gateway queue manager. Then, you can specify the remote queue manager in the REST API resource URL to perform the specified administrative action.

Before you begin

You can prevent remote administration by disabling the administrative REST API gateway. For more information, see Configuring the administrative REST API gateway.

To use the administrative REST API gateway, the following conditions must be met:

- The mqweb server must be configured and started. For more information about configuring and starting the mqweb server, see “Getting started with the administrative REST API” on page 73.
- The queue manager that you want to configure as the gateway queue manager must be in the same installation as the mqweb server.
- The remote queue manager that you want to administer must be IBM MQ 8.0 or later.
- You must ensure that any attributes that are specified in your request are valid for the system that you are sending the request to. For example, if the gateway queue manager is on Windows and the remote queue manager is on z/OS, you cannot request that the `dataCollection.statistics` attribute is returned for an HTTP GET request on the queue resource.
- You must ensure that any attributes that are specified in your request are valid for the level of IBM MQ that you are sending the request to. For example, if the remote queue manager is running IBM MQ 8.0,

you cannot request that the `extended.enableMediaImageOperations` attribute is returned for an HTTP GET request on the queue resource.

- You must use one of these supported REST resources:
  - `/queue`
  - `/subscription`
  - `/channel`
  - `/mqsc`
  - `/qmgr`

    The `/qmgr` resource returns only a subset of the attributes when you query a remote queue manager: `name`, `status.started`, `status.channelInitiatorState`, `status.ldapConnectionState`, `status.connectionCount`, and `status.publishSubscribeState`.

About this task

To use the administrative REST API gateway to administer remote queue managers, you must prepare the queue managers for remote administration. That is, you must configure transmission queues, listeners, and sender and receiver channels between the gateway queue manager and the remote queue manager. You can then send a REST request to the remote queue manager by specifying the queue manager in the resource URL. The gateway queue manager is specified by either using the **setmqweb** command to set the `mqRestGatewayQmgr` attribute to the name of the gateway queue manager, or sending the name of the gateway queue manager in a header that is sent with the request. The request is sent through the gateway queue manager to the remote queue manager. The response is returned with a header that indicates the queue manager that was used as the gateway queue manager.

Procedure

1. Configure communications between the gateway queue manager and the remote queue managers that you want to administer. These configuration steps are the same steps that are required to configure remote administration by runmqsc and PCF.

   For more information about these steps, see "Configuring queue managers for remote administration" on page 201.
2. Configure security on the remote queue managers:

   a) Ensure that the relevant user IDs exist on the system that the remote queue manager runs on. The user ID that must exist on the remote system depends on the role of the REST API user:

   - If the REST API user is in the MQWebAdmin or the MQWebAdminRO group, the user ID that started the mqweb server must exist on the remote system. On the IBM MQ Appliance, the user that starts the mqweb server is `mqsystem`.
   - If the REST API user is in the MQWebUser group, that REST API user ID must exist on the remote system.

   b) Ensure that the relevant user IDs are granted the necessary levels of authority to access the appropriate REST API resources on the remote queue manager:

   - Authority to put messages to the `SYSTEM.ADMIN.COMMAND.QUEUE`.
   - Authority to put messages to the `SYSTEM.REST.REPLY.QUEUE`.
   - Authority to access the transmissions queues that are defined for remote administration.
   - Authority to display queue manager attributes.
   - Authority to perform the REST requests. For more information, see the Security requirements section of the REST API resources reference topics.
3. Configure which local queue manager is used as the gateway. You can configure a default gateway queue manager, specify the gateway queue manager in an HTTP header, or use a combination of both approaches:

- Configure a default gateway queue manager by using the **setmqweb** command:

  ```
  setmqweb properties -k mqRestGatewayQmgr -v qmgrName
  ```

  where *qmgrName* is the name of the gateway queue manager.

  This gateway queue manager is used when both the following statements are true:

  – A queue manager is not specified in the `ibm-mq-rest-gateway-qmgr` header of a REST request.
  – The queue manager that is specified in the REST API resource URL is not a local queue manager.
- Configure the gateway queue manager on every REST request by setting the HTTP header `ibm-mq-rest-gateway-qmgr` to the name of the gateway queue manager.

4. Include the name of the remote queue manager that you want to administer in the resource URL.

   For example, to get information about the remote queue manager `remoteQM`, use the following URL:

   ```
   https://localhost:9443/ibmmq/rest/v3/admin/qmgr/remoteQM
   ```

Results

An `ibm-mq-rest-gateway-qmgr` header is returned with the REST response. This header specifies which queue manager was used as the gateway queue manager.

If you have difficulty with using the administrative REST API to administer remote queue managers:

- Check that the remote queue manager is running.
- Check that the command server is running on the remote system.
- Check that the channel disconnect interval has not expired. For example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.

Example

In the following example, there are three IBM MQ installations on two machines. On `Machine 1`, there is an `Installation 1` and an `Installation 2`. On `Machine 2`, there is an `Installation 3`. An mqweb server is configured for `Installation 1`. There is a single queue manager in each installation, and these queue managers are configured for remote administration. That is, the following listeners, channels and queues are configured and started:

- On queue manager QM1, in `Installation 1`, on `Machine 1`:
  – Sender channel QM1.to.QM2
  – Receiver channel QM2.to.QM1
  – Sender channel QM1.to.QM3
  – Receiver channel QM3.to.QM1
  – Transmission queue QM2
  – Transmission queue QM3
  – A listener configured on port 1414
- On queue manager QM2, in `Installation 2`, on `Machine 1`:
  – Sender channel QM2.to.QM1
  – Receiver channel QM1.to.QM2
  – Transmission queue QM1
  – A listener configured on port 1415
- On queue manager QM3, in `Installation 3`, on `Machine 2`:
  – Sender channel QM3.to.QM1

- Receiver channel QM1.to.QM3
- Transmission queue QM1
- The default listener

A queue, `Qon2` is defined on QM2, and a queue `Qon3` is defined on QM3.

The user `mquser` is defined on both machines, is granted the MQWebAdmin role in the REST API, and is granted authority to access the appropriate queues on each queue manager.

The setmqweb command is used to configure queue manager QM1 as the default gateway queue manager.

The following diagram shows this configuration:

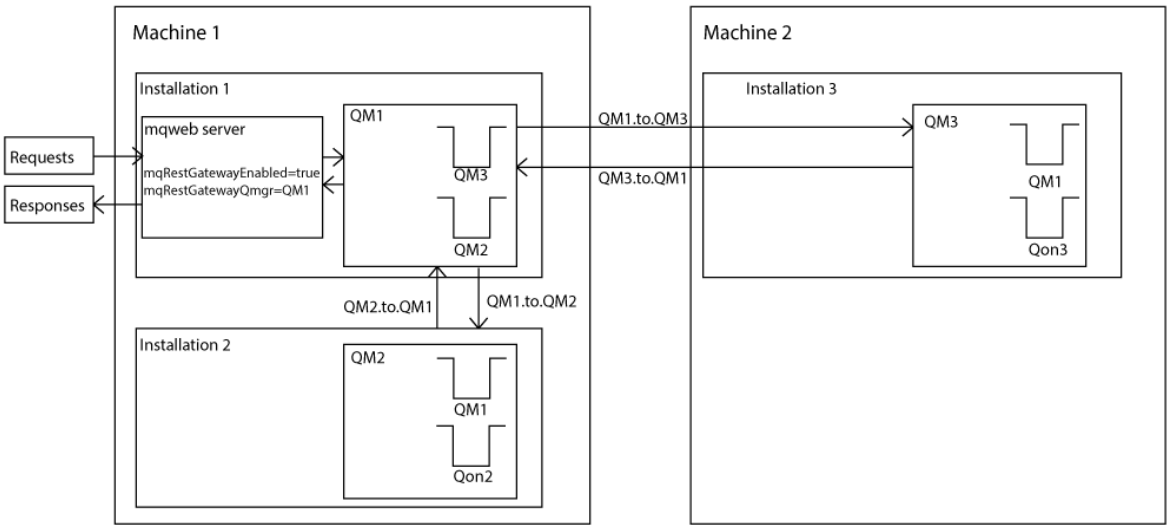


Figure 14. Diagram of example configuration for remote administration by using the REST API.

The following REST request is sent to the mqweb server:

```
POST https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM2/mqsc
```

The REST request includes the following JSON:

```
{
  "type": "runCommandJSON",
  "command": "display",
  "qualifier": "qlocal",
  "name": "Qon2",
  "responseParameters": ["type", "descr"]
}
```

The following response is received:

```
{
   "commandResponse": [   {
      "completionCode": 0,
      "reasonCode": 0,
      "parameters":       {
         "descr": "Queue on QM2, Installation 2",
         "type": "QLOCAL",
         "queue": "Qon2"
      }
   }],
   "overallReasonCode": 0,
   "overallCompletionCode": 0
}
```

The following REST request is sent to the mqweb server:

```
POST https://localhost:9443/ibmmq/rest/v3/admin/action/qmgr/QM3/mqsc
```

The REST request includes the following JSON:

```
{
  "type": "runCommandJSON",
  "command": "display",
  "qualifier": "qlocal",
  "name": "Qon3",
  "responseParameters": ["type", "descr"]
}
```

The following response is received:

```
{
   "commandResponse": [   {
      "completionCode": 0,
      "reasonCode": 0,
      "parameters":       {
         "descr": "Queue on QM3, Installation 3",
         "type": "QLOCAL",
         "queue": "Qon3"
      }
   }],
   "overallReasonCode": 0,
   "overallCompletionCode": 0
}
```

REST API time stamps

When date and time information is returned by the administrative REST API, it is returned in Coordinated Universal Time (UTC), and in a set format.

The date and time is returned in the following time stamp format:

```
YYYY-MM-DDTHH:mm:ss:sssZ
```

For example, `2012-04-23T18:25:43.000Z`, where the Z indicates that the time zone is Coordinated Universal Time (UTC).

The accuracy of this time stamp is not guaranteed. For example, if the mqweb server is not started in the same time zone as the queue manager that is specified in the resource URL, the time stamp might not be accurate. Further, if Daylight Saving Time adjustments are necessary, the time stamp might not be accurate.

REST API error handling

The REST API reports errors by returning an appropriate HTTP response code, for example 404 (Not Found), and a JSON response. Any HTTP response code that is not in the range 200 - 299 is considered an error.

The error response format

The response is in JSON format in UTF-8 encoding. It contains nested JSON objects:

- An outer JSON object that contains a single JSON array called `error`.
- Each element in the array is a JSON object that represents information about an error. Each JSON object contains the following properties:

  type
  String.

  The type of error.

messageId
String.

A unique identifier for the message of the form MQWBnnnnX. This identifier has the following elements:

MQWB
A prefix that shows that the message originated in the IBM MQ Rest API.

nnnn
A unique number that identifies the message.

X
A single letter that denotes the severity of the message:

- I if a message is purely informational.
- W if a message is warning of an issue.
- E if a message indicates that an error occurred.
- S if a message indicates that a severe error occurred.

message
String.

A description of the error.

explanation
String.

An explanation of the error.

action
String.

A description of steps that can be taken to resolve the error.

qmgrName

z/OS This field is only available for z/OS where the queue manager is a member of the queue sharing group. You must have specified the **commandScope** optional query parameter, or the **queueSharingGroupDisposition** attribute.

String.

The name of the queue manager that experienced the error.

This field is not applicable for the messaging REST API.

completionCode
This field is only available when **type** is pcf, java, or rest.

Number.

The MQ completion code associated with the failure.

reasonCode
This field is only available when **type** is pcf, java, or rest.

Number.

The MQ reason code associated with the failure.

exceptions
This field is only available when **type** is java.

Array.

An array of chain Java or JMS exceptions. Each element of the exceptions array contains a **stackTrace** string array.

The **stackTrace** string array contains the details of each exception split into lines.

Errors with queue sharing groups

z/OS

In a queue sharing group, it is possible to specify an optional query parameter of **commandScope** for certain commands. This parameter allows the command to be propagated to other queue managers in the queue sharing group. Any one of these commands can fail independently, resulting in some commands succeeding and some commands failing for the queue sharing group.

In cases where a command partially fails, an HTTP error code of 500 is returned. For each queue manager that generated a failure, information on that failure is returned as an element in the `error` JSON array. For each queue manager that successfully ran the command, the name of the queue manager is returned as an element in a `success` JSON array.

Examples

- The following example shows the error response to an attempt to get information about a queue manager that does not exist:

```
"error": [
   {
      "type": "rest",
      "messageId": "MQWB0009E",
      "message": "MQWB0009E: Could not query the queue manager 'QM1'",
      "explanation": "The MQ REST API was invoked specifying a queue manager name which
cannot be located.",
      "action": "Resubmit the request with a valid queue manager name or no queue manager
name, to retrieve a list of queue managers. "
   }
]
```

- z/OS The following example shows the error response to an attempt to delete a queue in a queue sharing group that does not exist for some queue managers:

```
"error" : [
  {
    "type": "rest",
    "messageId": "MQWB0037E",
    "message": "MQWB0037E: Could not find the queue 'missingQueue' - the queue manager reason
code is 3312 : 'MQRCCF_UNKNOWN_OBJECT_NAME'",
    "explanation": "The MQ REST API was invoked specifying a queue name which cannot be
located.",
    "action": "Resubmit the request with the name of an existing queue, or with no queue name
to retrieve a list of queues.",
    "qmgrName": "QM1"
  },
  {
    "type": "rest",
    "messageId": "MQWB0037E",
    "message": "MQWB0037E: Could not find the queue 'missingQueue' - the queue manager reason
code is 3312 : 'MQRCCF_UNKNOWN_OBJECT_NAME'",
    "explanation": "The MQ REST API was invoked specifying a queue name which cannot be
located.",
    "action": "Resubmit the request with the name of an existing queue, or with no queue name
to retrieve a list of queues.",
    "qmgrName": "QM2"
  }
],
"success" : [{"qmgrName": "QM3"}, {"qmgrName": "QM4"}]
```

Errors with MFT requests

If MFT REST API services are not enabled, and you invoke the MFT REST API, you receive the following exception:

```
{"error": [{
  "action": "Enable the Managed File Transfer REST API and resubmit the request.",
  "completionCode": 0,
  "explanation": "Managed File Transfer REST calls are not permitted as the service is
disabled.",
```

```
  "message": "MQWB0400E: Managed File Transfer REST API is not enabled.",
  "msgId": "MQWB0400E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

If MFT REST API services are enabled and the coordination queue manager is not set in the `mqwebuser.xml` file, you receive the following exception:

```
{"error": [{
  "action": "Set the coordination queue manager name and restart the mqweb server.",
  "completionCode": 0,
  "explanation": "Coordination queue manager name must be set before using Managed File
Transfer REST services.",
  "message": "MQWB0402E: Coordination queue manager name is not set.",
  "msgId": "MQWB0402E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

REST API discovery

Documentation for the REST API is available within the IBM Documentation and in Swagger format. Swagger is a commonly used approach for documenting REST APIs. The Swagger documentation for the REST API can be viewed by enabling the API Discovery feature (apiDiscovery) on the mqweb server.

Before you begin

Stabilized

Important: The apiDiscovery feature has been stabilized. You can still use this feature. At present, IBM MQ does not support the use of the `mpOpenAPI` feature.

You must enable security for the mqweb server to view the Swagger documentation by using API Discovery. For more information about the steps that are required to enable security, see IBM MQ Console and REST API security.

Procedure

1. Locate the `mqwebuser.xml` file in one of the following directories:
   - ALW *MQ_DATA_PATH*/web/installations/*installationName*/servers/mqweb
   - z/OS *WLP_user_directory*/servers/mqweb

     Where *WLP_user_directory* is the directory that was specified when the **crtmqweb** script ran to create the mqweb server definition.
2. Add the appropriate XML to the `mqwebuser.xml` file:
   - If the <featureManager> tags exist in your `mqwebuser.xml` file, add the following XML within the <featureManager> tags:

     ```
     <feature>apiDiscovery-1.0</feature>
     ```
   - If the <featureManager> tags do not exist in your `mqwebuser.xml` file, add the following XML within the <server> tags:

     ```
     <featureManager>
          <feature>apiDiscovery-1.0</feature>
     </featureManager>
     ```
3. View the Swagger documentation by using one of the following methods:
   - Display a web page that you can browse and try out the REST API by entering the following URL in a browser:

     `https://host:port/ibm/api/explorer`

In addition to authenticating each request, you must include a `ibm-mq-rest-csrf-token` header for each POST, PATCH, or DELETE request. The contents of this header can be any string, including blank.

This request header is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

- Retrieve a single Swagger 2 document that describes the whole REST API by issuing an HTTP GET to the following URL:

  `https://`*`host`*`:`*`port`*`/ibm/api/docs`

  This document can be used for applications where you want to programmatically navigate the available APIs.

host
Specifies the host name or IP address that the REST API is available on.

The default value is `localhost`.

port
Specifies the HTTPS port number that the administrative REST API uses.

The default value is 9443.

If the host name or port number is changed from the default, you can determine the correct values from the REST API URL. Use the **`dspmqweb status`** command to view the URL.

Related information
dspmqweb status (display mqweb server status)

REST API national language support

The REST API supports, with certain qualifications, the ability to specify national languages as part of an HTTP request.

Background

HTTP headers allow particular behavior to be specified on requests and additional information to be provided in responses.

Included in the HTTP headers is the ability to request that information is returned in a national language. The REST API honors this header where possible.

Specifying a national language

In the ACCEPT-LANGUAGE HTTP header, one or more language tags can be supplied. You can optionally associate a rank with the tags, allowing for the specification of a list ordered by preference. This page has a helpful discussion of the principle.

The REST API honors this header, selecting a language from the ACCEPT-LANGUAGE header and returning messages in that language. When the ACCEPT-LANGUAGE header contains no language that the REST API can support, messages are returned in a default language. This default language corresponds to the default locale of the REST API web server.

The “What data is translated?” on page 87 section explains what data is translated.

Indicating the applicable language on responses

The CONTENT-LANGUAGE HTTP header on responses from the REST API indicates the language in which the messages are returned.

What data is translated?

Error and informational messages are translated, other text is not.

- Data that is returned from a queue manager is not translated - for example in the case of executing an MQSC command via the REST API, the queue manager's responses are in the locale of the queue manager.
- The generated (Swagger) documentation for the REST API, as exposed via the `apiDiscovery` feature, is in English.

What languages are supported?

In addition to English, REST API error and informational messages are translated into the following languages.

Chinese(Simplified)
Denoted by the language tag `zh_CN`

Chinese(Traditional)
Denoted by the language tag `zh_TW`

Czech
Denoted by the language tag `cs`

French
Denoted by the language tag `fr`

Hungarian
Denoted by the language tag `hu`

Italian
Denoted by the language tag `it`

Japanese
Denoted by the language tag `ja`

Korean
Denoted by the language tag `ko`

Polish
Denoted by the language tag `pl`

(Brazilian) Portuguese
Denoted by the language tag `pt_BR`

Spanish
Denoted by the language tag `es`

Examples

In the examples, the web server has an English default locale.

Specifying a single supported language
In the request headers, `ACCEPT-LANGUAGE` is set to `fr`. This setting specifies that French is the preferred language for translatable text.

In the response headers, `CONTENT-LANGUAGE` is set to `fr`. This setting indicates that error and informational messages in the response are in French.

Specifying a list of languages
In the request headers, `ACCEPT-LANGUAGE` is set to `am, fr`. This setting specifies that Amharic and French are acceptable languages for translatable text and that Amharic is the preferred language for translatable text.

In the response headers, `CONTENT-LANGUAGE` is set to `fr`. This setting indicates that error and informational messages in the response are in French, as the REST API does not support Amharic.

Specifying a single unsupported language

In the request headers, `ACCEPT-LANGUAGE` is set to `am`. This setting specifies that Amharic is the preferred language for translatable text.

In the response headers, `CONTENT-LANGUAGE` is set to `en`. This setting indicates that error and informational messages in the response are in English, as the REST API does not support Amharic.

REST API versions

The REST API version number forms part of the base URL for REST requests. For example, `https://localhost:9443/ibmmq/rest/`**`v3`**`/admin/installation`. The version number is used to isolate clients from changes to the REST API that might be introduced in future releases.

IBM MQ 9.3.0 introduces version 3 of the REST API. This version increase applies to the administrative REST API, messaging REST API, and MFT REST API. This version increase changes the resource URL that is used for the REST API. The URL prefix for the resource URLs at version 3 is the following URL:

```
https://host:port/ibmmq/rest/v3/
```

Stabilized Some changes that are introduced to the REST API might change existing REST API function such that clients that use the REST API might need to be updated. To prevent such changes from forcing clients to be updated, the REST API version number is increased, and existing function stabilized at the previous number. The new function that might change the existing function is added to the REST API at the new version number. Therefore, clients can continue to use the REST API at the previous version without being updated.

The REST API changes that might result in requiring a client update includes the following changes:

- Removal of support for an existing attribute in the JSON that is sent to, or returned from, the REST API.
- Removal of a URL, HTTP verb, or header. For example, if a URL or header is renamed, or if a different verb is used.
- Addition of a new mandatory JSON attribute to data that is sent to an existing URL.
- Addition of a new mandatory HTTP header to data that is sent to an existing URL.
- Addition of a new mandatory query parameter to an existing URL.

When this type of change is introduced to REST API function that existed in a Long Term Support (LTS) release, the version number of the REST API is increased for the first of these changes. Any subsequent changes that are made within a Continuous Delivery (CD) release that might require changes to clients that use the REST API use the new version number.

This version number remains the same throughout subsequent CD releases until the next LTS release. Therefore, the version number increases at most once between LTS releases.

Stabilized When the version number is increased, the existing REST API function is stabilized at the old version number. That is, the existing REST API function that was available at the LTS release remains available at the old version number, but no further changes are made to that version. Any new function that is added to the REST API is added to the new REST API version. However, any additions that are made to the REST API in CD releases before the version increase are not guaranteed to be included in the older version of the REST API.

Deprecated Existing clients can continue to use the REST API at the old version number without requiring any changes. Older versions of the REST API might be deprecated, and eventually removed.

Some changes do not require changes to clients that use the REST API. These changes do not result in an increase of version number. Therefore, ensure that any client that uses the REST API does not need to be updated when these types of changes are introduced. These changes to the REST API might include the following changes:

- Addition of a new JSON attribute to existing data that is returned from the REST API.
- Addition of a new URL.

- Addition of a new HTTP verb to an existing URL.
- Addition of a new status code to an existing URL.
- Addition of new optional JSON attributes to data that is sent to an existing URL.
- Addition of new query parameters on an existing URL.
- Addition of new headers to data that is sent to an existing URL.
- Return of new headers from the REST API.

Changes to new Continuous Delivery REST API function

For new REST API function that is added in a CD release, any changes that are made to this new function that might then require changes to REST API clients do not increase the version number. That is, the new function can change before the next LTS release without increasing the version number. When the function is included in an LTS release, any subsequent changes that might require changes to REST API clients do increase the version number.

Example

1. At LTS release X, the REST API is at version 1.
2. At CD release X.0.1, support for a new URL is added. This change does not require changes to clients that use the REST API. Therefore, the REST API remains at version 1.
3. At CD X.0.2, support for a new URL is added. This change does not require changes to clients that use the REST API. Therefore, the REST API remains at version 1.
4. At LTS release Y, the REST API is at version 1.
5. At CD release Y.0.1, an existing URL is renamed. This change might require changes to clients that use the REST API. Therefore, a new version of the REST API is created as version 2. The renamed URL is included in version 2 of the REST API, along with all the existing function. Any new function that is added to the REST API is added to version 2. Version 1 remains stabilized at the level in LTS release Y.
6. At CD release Y.0.2, another existing URL is renamed. As the version is already increased in CD release Y, the REST API remains at version 2. Version 1 remains stabilized at the level in LTS release Y.
7. At LTS release Z, the REST API remains at version 2. Version 1 remains stabilized at the level in LTS release Y.

Administration using the IBM MQ Console

You can perform basic administration tasks by using the IBM MQ Console.

Note: Do not disable the command server on any of your queue managers when you use the IBM MQ Console. If the command server is disabled for a queue manager:

- The IBM MQ Console becomes unresponsive, with long delays to the processing of commands
- Any commands that are issued to the queue manager time out.

Related tasks

Tracing the IBM MQ Console

Getting started with the IBM MQ Console

To get started with the IBM MQ Console, you determine the URI for the console, connect to it and log in.

Before you begin

Before you can use the IBM MQ Console the mqweb server must be installed and configured. For more details, see Basic configuration for the mqweb server.

The mqweb server must be started, see strmqweb (start mqweb server).

To complete this task, you must be a user with certain privileges so that you can use the **dspmqweb** command:

- z/OS On z/OS, you must have authority to run the **dspmqweb** command, and write access to the `mqwebuser.xml` file.
- Multi On all other operating systems, you must be a privileged user.

IBM i On IBM i, the commands should be running in QSHELL.

About this task

Be aware of the following restrictions of the IBM MQ Console:

- z/OS
  - Queue managers on z/OS cannot be created, deleted, started or stopped.
  - Channel initiators on z/OS cannot be started or stopped, and the channel initiator status is not displayed.
  - Listeners cannot be administered.
  - Start, ping, resolve, and reset channel commands can only be issued with CHLDISP(DEFAULT).
  - Objects defined with QSGDISP(GROUP) cannot be displayed or managed.
  - Queue manager security cannot be managed.
  - System resource usage cannot be monitored.

Procedure

1. If the mqweb server is not already configured for use by the IBM MQ Console, configure the mqweb server.

   For more information about configuring the mqweb server with a basic registry, see Basic configuration for the mqweb server.
2. z/OS

   On z/OS, set the WLP_USER_DIR environment variable so that you can use the **dspmqweb** command. Set the variable to point to your mqweb server configuration by entering the following command:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   where *WLP_user_directory* is the name of the directory that is passed to `crtmqweb`. For example:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   For more information, see Creating the mqweb server.
3. Determine the URI for the IBM MQ Console by entering the following command:

   ```
   dspmqweb status
   ```

   The command generates an output similar to the following:

   ```
   MQWB1124I: Server 'mqweb' is running.
      URLS:
      https://localhost:9443/ibmmq/rest/v1/
      https://localhost:9443/ibmmq/console/
   ```

   The URI for the IBM MQ Console ends with the suffix `console/`.
4. Connect to the IBM MQ Console by entering the URL from the previous step in a browser.

   A security exception might be produced by the browser because the default certificate that is provided with the mqweb server is not a trusted certificate. Choose to proceed to the IBM MQ Console.

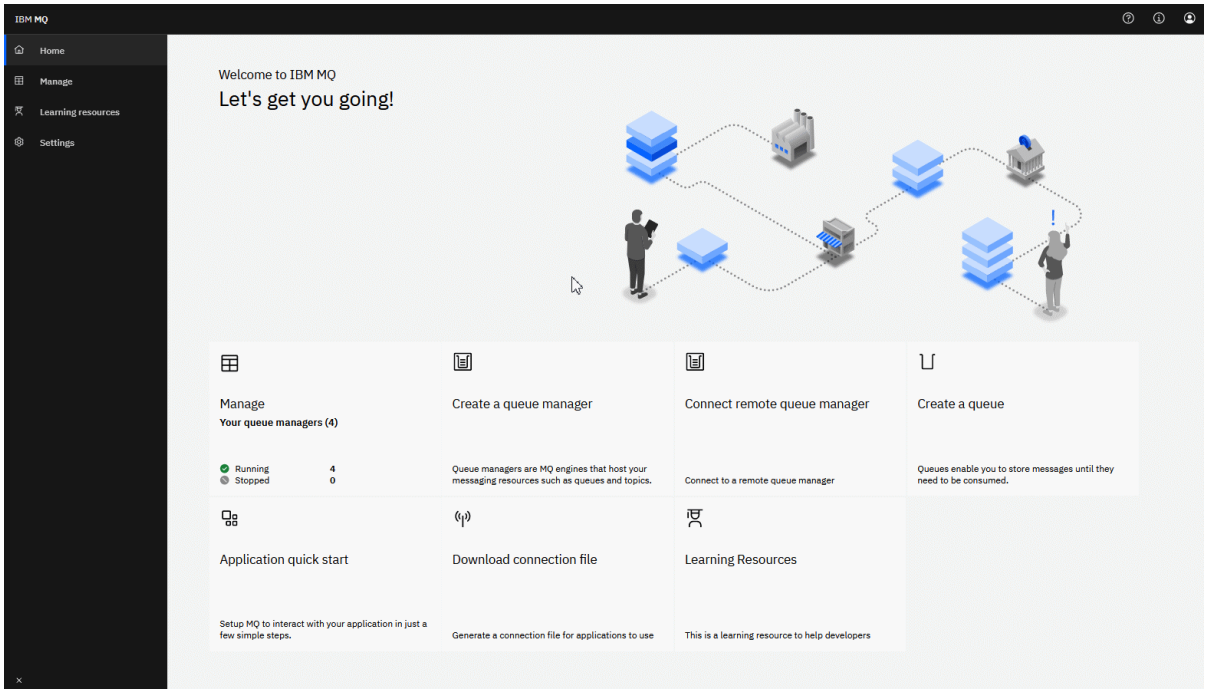
5. Log in to the IBM MQ Console. If you have used the default settings provided in the basic registry sample file when configuring the mqweb server, you can use the user name `mqadmin` and the password `mqadmin` to log in.

What to do next

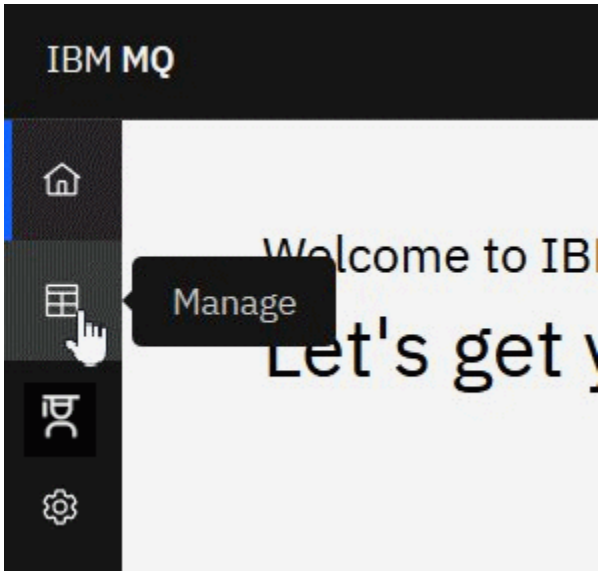
By default, users log in to the IBM MQ Console with a user ID and password, and an LTPA token is used by the browser to authenticate further requests for the duration of the session. You can also use client certificate authentication. For more information, see Using client certificate authentication with the REST API and IBM MQ Console.

Quick tour of the IBM MQ Console

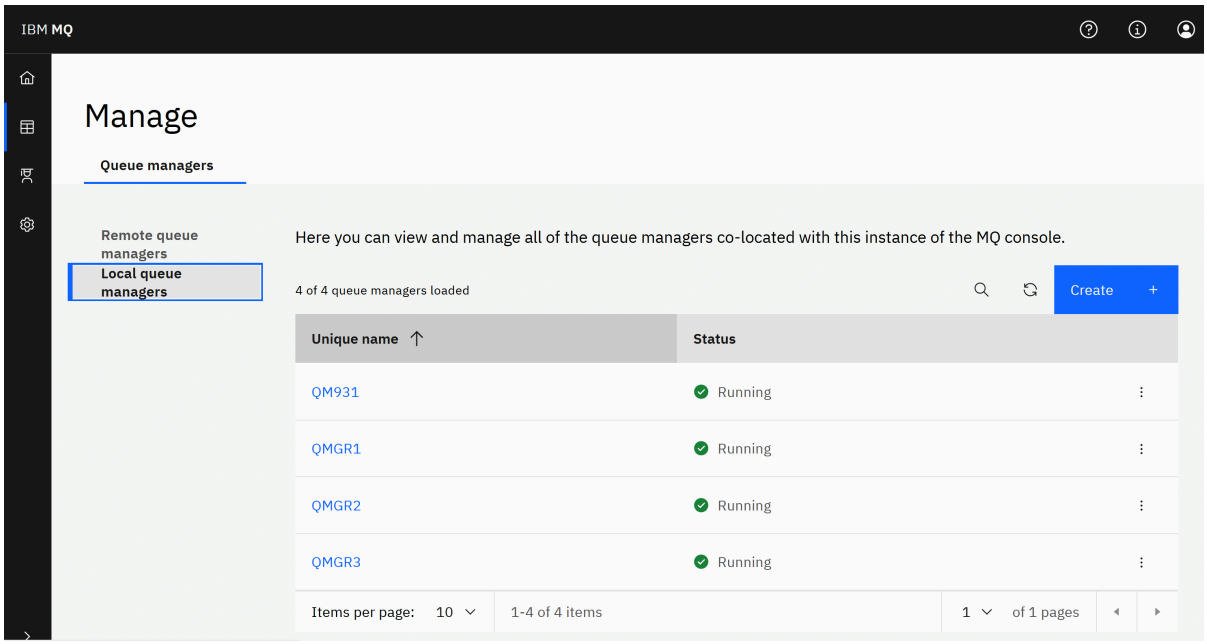
When you first log in to the IBM MQ Console you are taken to the landing page. From here you can choose to manage existing queue managers, create a queue manager or a queue, navigate to some education topics, or open the IBM MQ product information in IBM Documentation. You can also launch the application quick start, which guides you through the process of quickly and easily setting up messaging between new or existing queue managers and applications.



Alternatively, you can just click the manage icon to start managing IBM MQ objects straightaway.



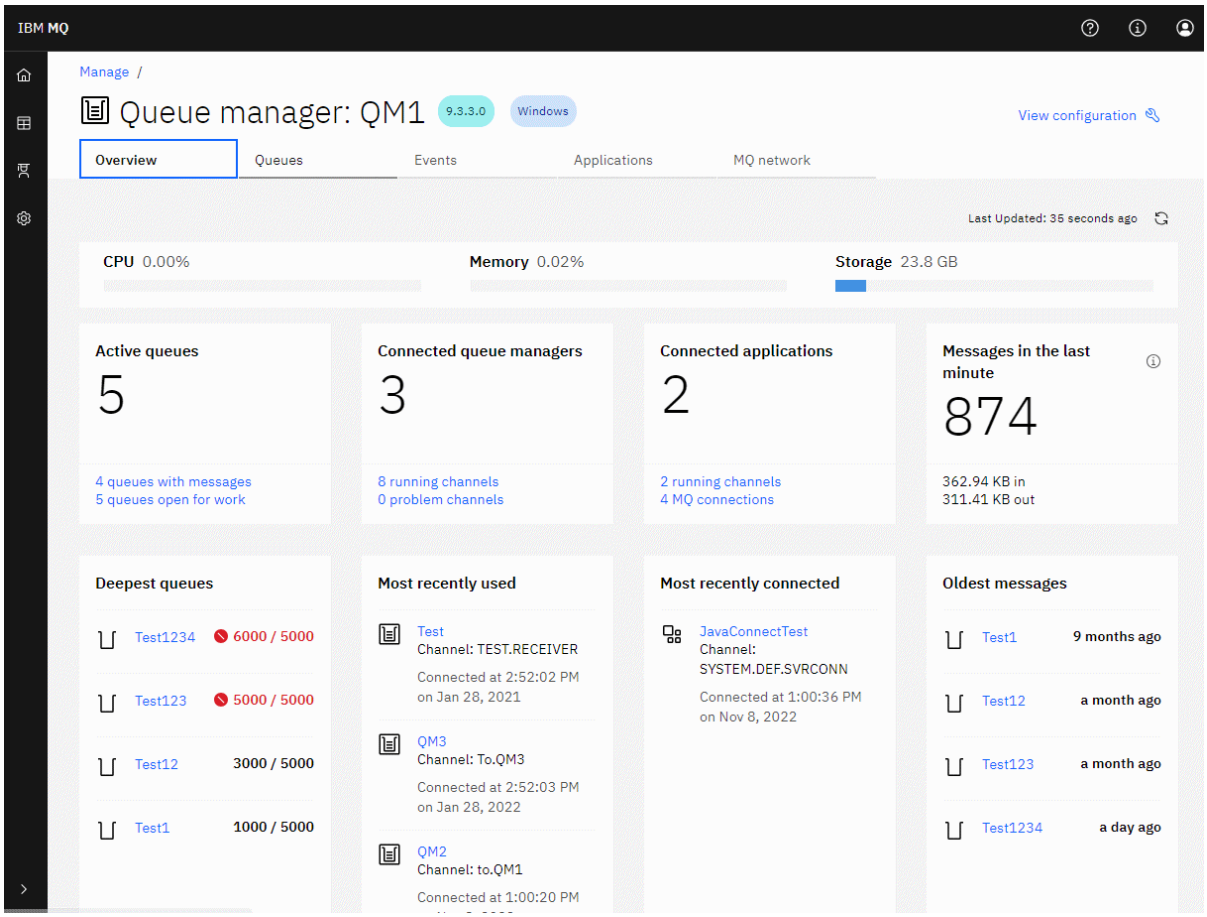
The manage view initially shows queue managers and their current state. You can also create new queue managers, and connect remote queue managers.



V 10.0.0 Each queue manager has a menu that lets you stop, configure, or view the status of a running queue manager, or start or delete a stopped queue manager.

Authority Records, Authentication Information objects, and Channel Authentication records for the queue manager can be found on the **Security** tab of the queue manager's **Configuration** page, where you can create and add new ones.

Click on the name of a running queue manager to open its dashboard.



From the queue manager dashboard you can complete the following actions:

On the **Overview** tab, view the following information:

CPU
Percentage estimate of CPU usage by the queue manager. (Not applicable on z/OS.)

Memory
Percentage estimate of memory usage by the queue manager. (Not applicable on z/OS or Windows.)

Storage
Percentage estimate of the free space of the disk on which the queue manager resides. (Not applicable on z/OS.)

Active queues
Count of queues that either have messages, or are open for input or output.

Connected queue managers
Count of currently connected queue managers as derived from active channels.

Connected applications
Count of currently connected applications.

Messages in the last minute
Displays a summary of the PUT/GET system topics that show message throughput every 10 seconds. (Not applicable on z/OS.)

Subscriptions
Displays a count of subscriptions. Only visible on z/OS and on other platforms where monitoring of system topics is inhibited (see setmqweb properties).

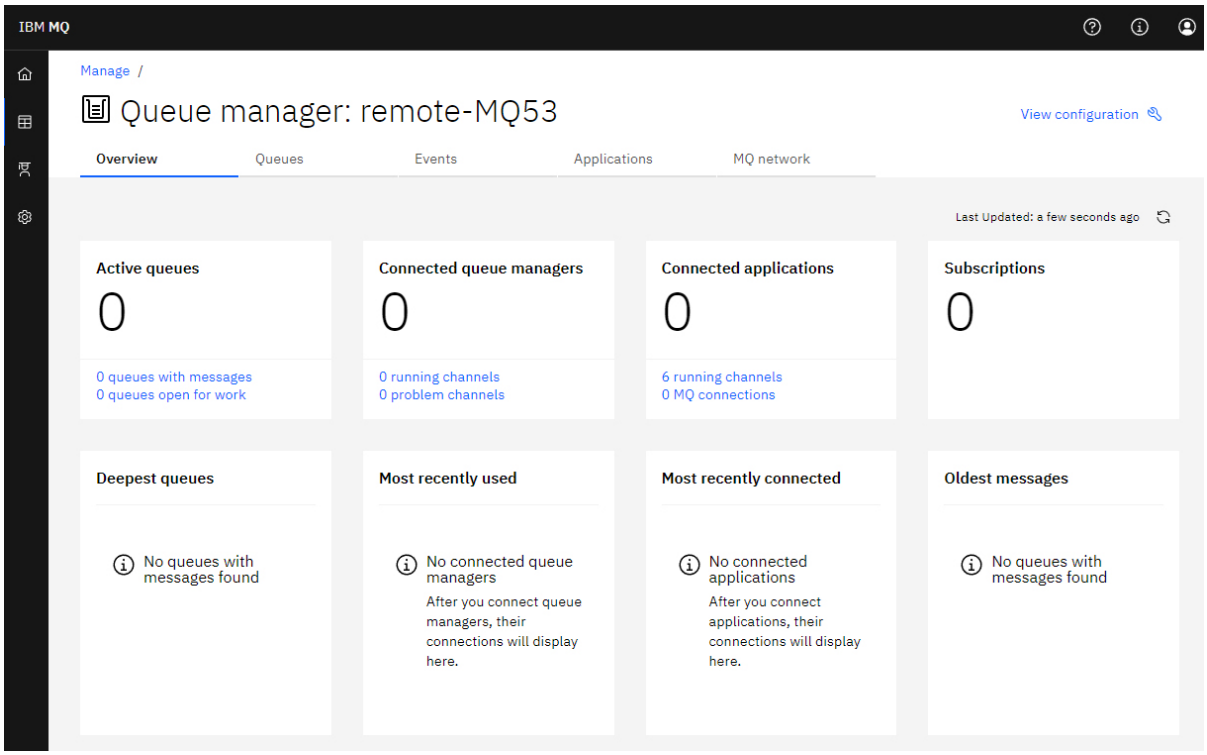
Deepest queues
Lists queues in order of depth. Shows current queue depth and maximum queue depth.

Most recently used
Lists currently connected queue managers, ordered by last message date.

Most recently connected
Lists currently connected applications as derived from active server-connection channels, ordered by channel start date and time.

Oldest messages
Lists queues ordered by the oldest message date and time.

The information displayed on the **Overview** tab is derived from the monitoring of system topics (see Metrics published on the system topics). z/OS does not support system topic monitoring, and the monitoring for console display purposes can be disabled on other platforms (see setmqweb properties). In these cases the **Overview** tab displays more limited information and its appearance is similar to the following example:



On the **Queues** tab:

- Create new queues
- Click on a queue name to view existing messages and create new ones, and to configure the queue.

On the **Events** tab:

Topics

- Create new topics
- Configure existing topics
- Click on a topic name to view matching subscriptions

Subscriptions

- Create new managed or unmanaged subscriptions
- Configure existing subscriptions

On the **Applications** tab:

Overview
Contains tiles which gives overviews of the following statistics:

Connected applications
Displays a count of the number of connected applications. Provides links to the following tabs:

- **Application instances**
- **Connections**

Running channel instances
Displays a count of the number of SVRCONN channel instances and from that links to those that are defined or stopped on the **App Channel** tab.

Connections
Displays a count of the number of connections. Provides links to the following information on the **Connections** tab:

- Local connections (those without a channel name)
- Remote connections (those with a channel name)

Most common applications
Displays a list of frequent applications, ordered by the number of connections they are using.

Most common channels
Displays a list of frequent channels, ordered by the number of instances that are active.

Oldest transactions
Displays a list of oldest transactions by application name. These transactions have connections with open units of work and are ordered by the UOW start date and time.

Remote connected versions
Displays a list of common connected IBM MQ versions, that is, those channel instances that have a specified REMOTE_VERSION.

Application channel security
Displays a list of common connected channel security protocols, that is, those channel instances that have a specified SECURITY_PROTOCOL.

Channel transfer rates
Displays a list of common channels ordered by the transfer rates of messages and bytes. Uses the channel start date and time to calculate the duration and uses MSGS and MQIACH_BYTES_SENT/ MQIACH_BYTES_RCVD to calculate the rate.

Applications
View information about applications that are connected to the queue manager.

Channels
View activity on channels connected to applications.

App channels

- Start, stop, ping, and configure channels
- Create new channels
- Reset channels

App channel instances

- View status of application channel instances

- Resolve in-doubt messages on channels

On the **MQ Network** tab:

Overview

Contains tiles which gives overviews of the following statistics:

Running queue manager channel instances

Displays a count of the number of non-SVRCONN channel instances. Displays links to the following types of channel instances on the **Queue managers connected** tab:

- Defined channels
- Stopped channels

Connected queue managers

Displays a count of the queue managers connected by MQCA_REMOTE_Q_MGR_NAME. Also provides a count of queue managers returned by MQCMD_INQUIRE_CLUSTER_Q_MGR.

Cluster membership

If there is only one queue manager cluster, displays the name of the cluster and whether the queue manager is a full or partial repository. Displays how many queue managers are visible in the cluster. If there is more than one cluster, then displays the number of clusters plus a count of full and partial repository queue managers in each cluster.

Failing queue manager channels

Displays a list of channels that are in a retrying state (non stopped/running). Calculates the number of retries remaining if in the retry state. The list contains channels with the following status types:

- MQCHS_PAUSED
- MQCHS_RETRYING

Longest message delays

Displays a list of channels that have an XMIT time indicator (long period).

Unattended transmission queues

Displays a list of transmission queues that have a non-zero queue depth and no associated handles.

Remote connection versions

Displays a list of common connected IBM MQ versions, that is, those channel instances that have a specified REMOTE_VERSION.

Queue manager channel security

Displays a list of common connected channel security protocols, that is, those channel instances that have a specified SECURITY_PROTOCOL.

Cluster health

Displays a number of independent statistics related to the health of the cluster. The status includes:

- The number of cluster objects (queues, topics, queue managers).
- The number of suspended queue managers (MQIACF_SUSPEND set to YES).
- The depth of the SYSTEM.CLUSTER.COMMAND.QUEUE queue.
- The number of cluster queue manager entries starting with SYSTEM.TEMP.

If all of these values are zero then this tile is not displayed and the **Listeners** tile is shown instead..

Listeners

Displays a list of listeners and whether they are in a running state. Only shown if the **Cluster health** tile is not displayed.

Queue managers connected
View details of queue managers currently connected to this queue manager.

Queue manager channels

- Start, stop, ping, and configure channels
- Create new channels
- Reset channels

Queue manager channel instances

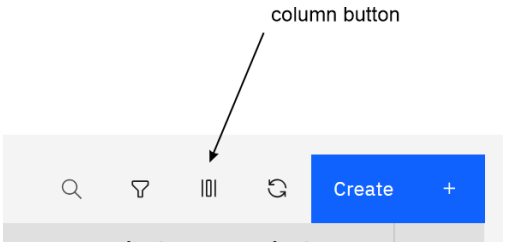
- View status of queue manager channel instances
- Resolve in-doubt messages on channels

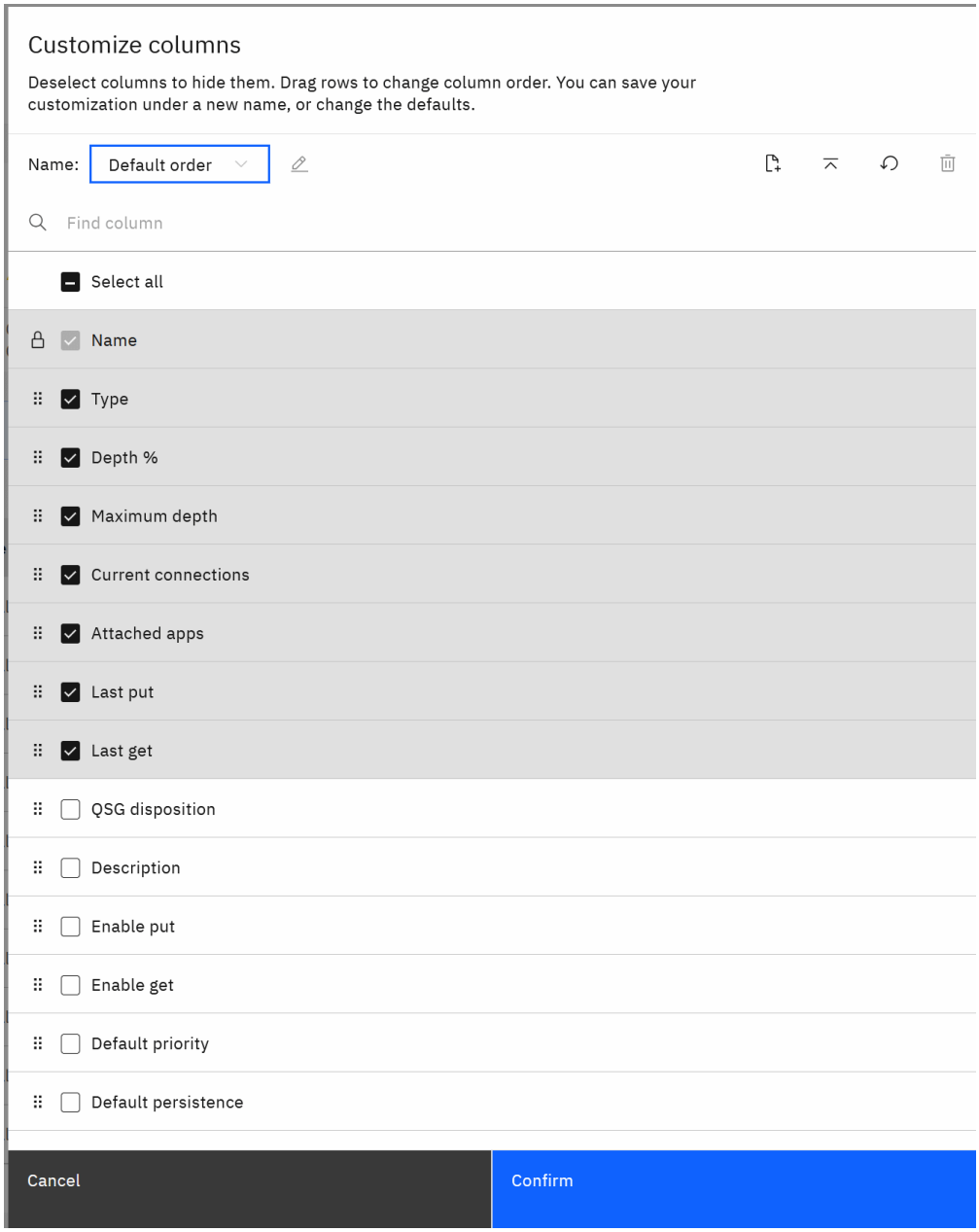
Customizing columns

V 10.0.0

You can simplify the view of an IBM MQ object by customizing the columns that are displayed.

Where a view has many columns, you can click the column button in the toolbar to open the Customize columns window. Select the columns that you want displayed and click **Confirm**.





You can create different column views and switch between them. The default view is always available to revert to.

To create a new view:

1. Open the Customize columns window.
2. Click the **add view** button in the toolbar . A blank view appears. It is automatically given the name **New View *n***.
3. Select the columns that you want to display, you can move to the top of the list by using the move button . Click **Confirm** when you are satisfied.

You can switch between views by selecting them from the **Name** list. Note that the columns view is saved in your browser. If you use a different browser, the views that you previously defined will not be available.

Filtering objects

V 10.0.0

You can create filters in some of the views on the console so that only the objects that you are interested in are displayed.

For information on creating and editing filters, see “IBM MQ Console: Filtering objects displayed in tables” on page 124.

IBM MQ Console: Working with local queue managers

You create, configure, and control local queue managers from the top level of the Manage view

.

About this task

Multi The Manage view lists the local queue managers that are added to the IBM MQ installation from which the IBM MQ Console is running. Queue managers that are associated with different installations of IBM MQ on the same system are not listed.

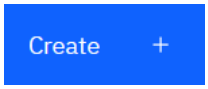
z/OS On z/OS, the Manage view lists the queue managers that are at the same version as the IBM MQ Console, and are defined on the system where the IBM MQ Console is running. Queue managers at a different version to the IBM MQ Console are not listed.

You can select individual queue managers from the list to work with. You an also view the status of a running queue manager.

Note: The IBM MQ Console can connect to a local RDQM queue manager when it is active (that is, has the primary role) , but does not offer any RDQM-specific features.

To enable the IBM MQ Console to connect to an RDQM, regardless of which node is the primary node, add the RDQM to the IBM MQ Console as a remote queue manager, and include the host names of all the nodes in the RDQM configuration in the remote queue manager connection. To simplify the connection to a remote HA RDQM, configure a floating IP address for the RDQM, and use this IP address in the remote queue manager configuration in the IBM MQ Console.

Procedure

- To create a new local queue manager:
  a) Click the create button 

in the queue manager list view.
  b) Enter a name for the new queue manager. The name can contain up to 48 characters. Valid characters are letters and numbers and the ".", "/", "_", and "%" characters.
  c) Optional: Enter an available TCP/IP port for the queue manager to listen on. The port number must not exceed 65535.
  d) Click **Create**. The new queue manager is created and started.
- To start a local queue manager:
  a) Locate the queue manager that you want to start in the list.
  b) Select **Start** from the menu ⋮.
- To stop a local queue manager:
  a) Select the queue manager that you want to stop from the list in the local queue manager widget.
  b) Select **Stop** from the menu ⋮.

- To delete a local queue manager:
  a) If the queue manager is running, stop it.
  b) Select **View configuration** from the menu and select **Delete queue manager**.
  c) Confirm that you want to delete the queue manager by entering its name in the confirmation window. The queue manager and all associated objects are deleted.
- To view and edit the properties of a local queue manager:
  a) Ensure that the queue manager is running, and locate it in the queue manager list.
  b) Select **View configuration** from the menu .
  c) Ensure the **Properties** tab is selected. View the properties and edit them as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For information about a property, you can view the property information in Queue manager properties.
- To work with security settings for the local queue manager:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu .
  c) Ensure the **Security** tab is selected.
  d) You can work with authentication objects, authorization records, or channel authentication objects. Visit the following topics for more information:
     - "IBM MQ Console: Working with authentication information objects" on page 100
     - "IBM MQ Console: Working with queue manager authority records" on page 102
     - "IBM MQ Console: Working with channel authentication records" on page 103

IBM MQ Console: Working with authentication information objects

You can use the console to add and delete authentication information objects on a queue manager. You can also view and set the properties, and manage the authority records for the objects.

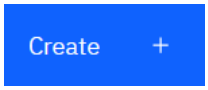
About this task

The authentication information view lists the authentication information that exists for a specific queue manager. You can select individual authentication information from the list to work with.

The queue manager authentication information forms part of IBM MQ support for Transport Layer Security (TLS). These objects contain the definitions that are required to perform certificate revocation checking by using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers, and the definitions that are required to enable user ID and password checking.

Procedure

- To view the authentication information for a queue manager:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu .
  c) Ensure the **Security** tab is selected.
  d) Select **Authentication information** from the navigation panel.
- To add an authentication information object:

a) Click the create button 

in the authentication information list view.

b) Specify the name of the authentication information object. Valid characters are letters and numbers and the ".", "/", "_", and "%" characters.

c) Specify the type of authentication information object.

d) Specify additional information appropriate to the object type:

- For **CRL LDAP**, specify the **LDAP server name**. This name is the host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number. You can optionally specify a username and password for the user accessing the LDAP server.
- For **OCSP**, specify the **OCSP responder URL**. This URL is the URL of the responder that is used to check for certificate revocation. This value must be an HTTP URL containing the host name and port number of the OCSP responder. If the OCSP responder is using port 80, which is the default for HTTP, then the port number can be omitted. HTTP URLs are defined in RFC 1738.
- For **IDPW OS**, there are no additional requirements although you can optionally specify further options for this authentication type.
- For **IDPW LDAP**, specify the **LDAP server name** and the **Short user** name. The LDAP server name is the host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number. The short user name is the field in the LDAP user record that is used as a short name for the connection. You can optionally specify further options for this authentication type.

e) Click **Add**.

- To delete an authentication information object:

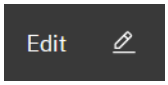
a) Select the spanner icon for the authentication information object that you want to delete from the list.

b) In the object properties view, click **Delete authentication info object**.

c) Confirm that you want to delete the authentication information object by clicking **Delete**. The object is deleted.

- To view and edit the properties of an authentication information object:

a) Select the spanner icon for the authentication information object that you want to view from the list.

b) To edit the displayed properties, click the Edit button 

c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line.

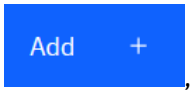
d) Click **Save** to save your changes.

- To view and edit authority records for an authentication information object:

a) Select the spanner icon for the authentication information object that you want to view authority record for from the list.

b) Select the **Security** tab.

c) To edit or delete an existing authority record, select **Edit** or **Delete** from the menu .

d) To add a new authority record, click the **Add** button 

, supply the details of the new authority record and click **Create**.

IBM MQ Console: Working with queue manager authority records

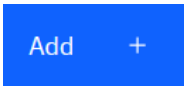
You can control the access that users and groups have to queue managers by specifying an authority record for that user or group.

About this task

You can fine-tune the access that a messaging user or group of messaging users has to a particular queue manager by using authority records. There are two types of authority records: the **queue manager access** records that control general authorities and the **permission to create** records that control which users and groups can create objects for the queue manager.

Procedure

- To view the authority records for a queue manager:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu ⋮.
  c) Ensure the **Security** tab is selected.
  d) Select **Authority records** from the navigation panel. The view shows the authority records in two panes, enabling you to work with general authority records and with create authority records.
- To add a general authority record:
  a) Click the add button

in the **Queue manager access** list view.
  b) Choose whether you are adding an authority record for a user or a group.
  c) Specify the name of the user or group you are adding an authority record for (the authority record takes this as its name).
  d) Select the authorities that you want to grant.
  e) Click **Create**.
- To add a create authority record:
  a) Click the add button 

in the **Permission to create** list view.
  b) Choose whether you are adding an authority record for a user or a group.
  c) Specify the name of the user or group you are adding an authority record for (the authority record takes this as its name).
  d) Select the types of object that you are granting the authority to create.
  e) Click **Create**.
- To delete an authority record:
  a) Select the authority record that you want to delete, and select **Delete**.
  b) Confirm that you want to delete the authentication information object by clicking **Delete**. The object is deleted.
- To view and edit the properties of an authority record:
  a) Click the authority record that you want to view.
  b) Change the settings as required and click **Save** to save your changes.

IBM MQ Console: Working with channel authentication records

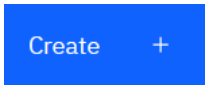
You can use the IBM MQ Console to add and delete channel authentication records on a queue manager. You can also view and set the properties for channel authentication records.

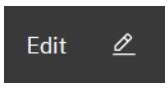
About this task

To exercise more precise control over the access that is granted to connecting systems at a channel level, you can use channel authentication records.

To enforce security, you can use blocking channel authentication records to block access to your channels. You can also use address map channel authentication records to allow access to specified users. To learn more about channel authentication records, see Channel authentication records.

Procedure

- To view the channel authentication information for a queue manager:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu ⋮.
  c) Ensure the **Security** tab is selected.
  d) Select **Channel authentication** from the navigation panel.
- To add a channel authentication record:
  a) Click the create button 

in the channel authentication information list view.
  b) Choose the rule type that you want to use. Select one **Allow**, **Block**, or **Warn**.
  c) Choose the type of identity that you are configuring a channel authentication rule for. Different identity types are available, depending on the rule type you selected.
  d) Provide the required information for the identity you are specifying. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.
  e) Click **Create** to create the channel authentication record.
  For more information about the available settings for channel authentication records, see Channel authentication records and SET CHLAUTH
- To delete a channel authentication record:
  a) Click the spanner icon next to the channel authentication record that you want to delete.
  b) In the Edit channel authentication view, click **Delete channel authentication object**.
  c) Confirm that you want to delete the channel authentication record by clicking **Delete**. The channel authentication record is deleted.
- To view and edit the properties of a channel authentication record:
  a) Click the spanner icon next to the channel authentication record that you want to edit or view. The properties are displayed.
  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line.
  d) Click **Save** to save your changes.

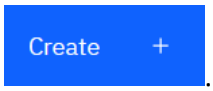
Multi *IBM MQ Console: Working with listeners*

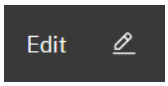
You can use the IBM MQ Console to add and delete listeners, start and stop listeners, view and set listener properties, view listener status details, and manage the authority records for a listener.

About this task

The listeners view displays the listeners that exist for a specific queue manager. You can select individual listeners to work with.

Procedure

- To view the listeners of a queue manager:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu ⋮.
  c) Select the **Listeners** tab.
- To create a listener:
  a) Click the create button 

.
  b) Provide the required information for the listener you are creating.
  c) Click **Create**. The new listener is created.
- To start a listener:
  a) Locate the listener that you want to start in the list.
  b) Select **Start** from the menu ⋮.
- To stop a listener:
  a) Locate the listener that you want to start in the list.
  b) Select **Stop** from the menu ⋮.
- To view and edit the properties of a listener:
  a) Locate the listener in the list.
  b) Select **View configuration** from the menu ⋮.
  c) Ensure the **Properties** tab is selected. To edit the properties, click the Edit button 

  d) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For more information on the properties, see Listener properties in the MQ Explorer documentation.
  e) Click **Save** to save your changes.
- To view and edit authority records for a listener:
  a) Locate the listener in the list.
  b) Select **View configuration** from the menu ⋮.
  c) Click the **Security** tab.
  d) Work with the authority records as described for queue manager authority records. See "IBM MQ Console: Working with queue manager authority records" on page 102.

- To delete a listener:
  a) Locate the listener in the list.
  b) Select **View configuration** from the menu .
  c) Click **Delete listener**.

IBM MQ Console: Viewing queue manager status

You can use the console to view the status of a running queue manager.

About this task

You can view status information equivalent to that returned by the **qmstatus** command for a running queue manager. The information displayed is shown in a **General** tab, a **Logging** tab, and a **Native HA** tab.

Procedure

- To view the status of a queue manager from the configuration view:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View configuration** from the menu .
  c) Click the **Status** tab to open the status view.
    - Click **General** to view general status information.
    - Click **Logging** to view status information about the queue manager log.
    - Click **Native HA** to view status information for a Native HA configuration (you are informed if you have no Native HA capabilities configured).

    If you have a Native HA configuration, a further tab appears:

    - Click **Replication** to view status details of the nodes and instances in your Native HA configuration. The nodes are displayed in tiles (one tile for Native HA, two tiles for Native HA CRR or Native IRR). Select a tile and the instances for that node are displayed in a table.

      Click the spanner icon in a node tile or against an instance entry to see detailed status for that entity.
- To view the status of a queue manager from the shortcut menu:
  a) Ensure that the queue manager is running, and select it in the queue manager list.
  b) Select **View status** from the menu .

IBM MQ Console: Adding a remote queue manager

You can use the IBM MQ Console to administer a queue manager that is running on a remote system. A remote queue manager can be either a queue manager that is running in a different installation on the same system as the IBM MQ Console, or a queue manager that is running on a different system

About this task

This task covers the steps that are required to prepare the remote queue manager to be administered with the IBM MQ Console. After you complete these steps, you must complete one of the tasks in the what to do next section before the queue manager is ready to be used with the IBM MQ Console.

Note: The steps in this task require you to run MQSC commands:

- ALW On AIX, Linux, and Windows, you issue MQSC commands from a **runmqsc** command prompt. See Running MQSC commands interactively under **runmqsc** and Running MQSC commands from text files under **runmqsc**. For this task, if you are running on AIX, Linux, and Windows, open a runmqsc command prompt that uses QM1:

```
runmqsc QM1
```

- IBM i On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. See Administration using MQSC commands on IBM i.
- z/OS On z/OS, MQSC commands can be issued from a number of sources, depending on the command. See Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

Attention: if you receive the following messages:

```
MQWB2026E: The request to connect to the remote queue manager 'rqmgr-qmgr_name' failed
with the error message:
'JMSCC0051: The property 'JMS_IBM_MQMD_AccountingToken' should be set using type '[B',
not 'java.lang.Object'.'
```

you are attempting to pass a `java.lang.Object` to the accounting token, when a Java object type `byte[]` is expected.

Procedure

1. If the mqweb server is not in a stand-alone IBM MQ Web Server installation, ensure that the mqweb server is configured to allow the IBM MQ Console to connect to remote queue managers:
   a) Check whether the mqweb server allows remote queue manager connections by entering the following command on the command line:

   ```
   dspmqweb properties -u
   ```

   If the property **mqConsoleRemoteSupportEnabled** appears in the output, and the property is set to true, then connections to remote queue managers are allowed.
   b) If the mqweb server does not allow remote queue manager connections, enable such connections by entering the following command on the command line:

   ```
   setmqweb properties -k mqConsoleRemoteSupportEnabled -v true
   ```

2. On the remote queue manager, create a server-connection channel to allow remote administration of the queue manager by using the **DEFINE CHANNEL** MQSC command.
   For example, to create a server-connection channel `QM1.SVRCONN` for queue manager QM1, enter the following MQSC command:

   ```
   runmqsc QM1
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   end
   ```

   For more information about **DEFINE CHANNEL** and the options available, see DEFINE CHANNEL.
3. Ensure that appropriate users are authorized to administer the queue manager and objects that are associated with the queue manager.
   For more information, see “Authorizing users to administer remote queue managers in the IBM MQ Console” on page 107.
4. ALW
   If no listener is running on the remote queue manager, create a listener to accept incoming network connections by using the **DEFINE LISTENER** MQSC command.
   For example, to create a listener `REMOTE.LISTENER` on port 1414 for remote queue manager QM1, enter the following MQSC command:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```

5. Ensure that the listener is running by using the **START LISTENER** MQSC command.

   ALW For example, on AIX, Linux, and Windows to start the listener REMOTE.LISTENER for queue manager QM1, enter the following MQSC command:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS For example, on z/OS, to start the listener, enter the following MQSC command:

   ```
   /cpf START LISTENER TRPTYPE(TCP) PORT(1414)
   ```

   The channel initiator address space must be started before you can start a listener on z/OS.

What to do next

Add the remote queue manager to the IBM MQ Console by using one of the following methods:

- "Adding a remote queue manager to the IBM MQ Console from the IBM MQ Console" on page 109
- "Adding a remote queue manager to the IBM MQ Console from the command line" on page 111

Authorizing users to administer remote queue managers in the IBM MQ Console

When you configure the IBM MQ Console to connect to a remote queue manager, you must consider which users must be authorized to connect to that queue manager. You need to consider two types of user: the user ID that is assigned to the channel that you use to connect to the remote queue manager, and any users with the MQWebUser role.

Users that require authorization

- The user ID that is assigned to the channel that you use to connect to the remote queue manager is used to authorize any users with the MQWebAdmin role when those users connect to the queue manager. Depending on the queue manager configuration, this user ID might be one of the following users:
  - A user ID that is the same as the user ID that starts the mqweb server that runs the IBM MQ Console on the system from which you want to remotely administer this queue manager.
  - A user ID that matches a user ID and password that is then included in the **setmqweb remote** command. By including the user ID and password in the **setmqweb remote** command, this user ID and password is used for authentication when the IBM MQ Console connects to the queue manager. That is, the user ID and password are checked against the connection authentication rules for the queue manager.
  - A user ID that is determined by channel security rules. For example, you might set a channel authentication rule on the server-connection channel to allow connections from the IP address from which you use the IBM MQ Console. This channel authentication rule might map all these connections to a specific user ID that is authorized to use the queue manager. For more information, see Creating new CHLAUTH rules for channels.

  For more information, see Determining which user is used for authorization.
- Any users that connect with the MQWebUser role also need authorization to access applicable objects on the queue manager. The initial connection to the remote queue manager is made by using the same user that authorizes users with the MQWebAdmin role. However, the MQWebUser user ID is then used for all subsequent authorization to access objects on the queue manager.

Objects that a user must be authorized to use

Beyond authority to connect to the remote queue manager, users must be authorized to use the following objects:

- The SYSTEM.REST.REPLY.QUEUE queue. This queue is used to handle requests made in the IBM MQ Console.
- The SYSTEM.ADMIN.COMMAND.QUEUE queue. This queue is used to handle requests made in the IBM MQ Console.
- The SYSTEM.REST.REPLY.QUEUE.TZ queue. This queue is temporarily created by the IBM MQ Console to determine which time zone the remote queue manager is in. Without permission to create, inquire, and delete this queue, the time-based information that is provided within the console might be inaccurate.
- The SYSTEM.ADMIN.TOPIC topic. On AIX, Linux, and Windows, this topic is used to get information for various tiles on the dashboards. Therefore, if the user does not have authority to use this topic, the information in some tiles might be missing or inaccurate.

Required authorities on AIX, Linux, and Windows

The single user that is used to authorize any users with the MQWebAdmin role must have the following authorities at minimum to connect to the remote queue manager:

- On the queue manager, **+inq +setall +dsp +connect +altuser** authorities. (The **+altuser** authority on the queue manager is required only if users with the MQWebUser role connect to the queue manager.)
- On the topic SYSTEM.ADMIN.TOPIC, **+sub +dsp** authorities.
- On the queue SYSTEM.REST.REPLY.QUEUE, **+put +get +browse +inq** authorities.
- On the queue SYSTEM.ADMIN.COMMAND.QUEUE, **+put +setall** authorities.
- On the queue SYSTEM.REST.REPLY.QUEUE.TZ, **+crt +inq +dlt** authorities.
- Other authorities to interact with objects such as queues, topics, subscriptions, and channels to administer the queue manager.

Any users with the MQWebUser role must have the following authorities at minimum:

- On the queue manager, **+dsp +connect** authorities.
- On the topic SYSTEM.ADMIN.TOPIC, **+sub +dsp** authorities.
- On the queue SYSTEM.REST.REPLY.QUEUE, **+put +get +browse +inq** authorities.
- On the queue SYSTEM.ADMIN.COMMAND.QUEUE, **+put** authority.
- Authority to interact with objects such as queues, topics, subscriptions, and channels to administer the queue manager, as appropriate for the user.

You can use the following tools to grant authority on the appropriate objects:

- ALW On AIX, Linux, and Windows, use the **setmqaut** control command on a standard command line. For more information, see **setmqaut**.
- On the IBM MQ Appliance, use the **SET AUTHREC** MQSC command. For more information, see **SET AUTHREC**.

Required authorities on z/OS

The single user that is used to authorize any users with the MQWebAdmin role needs the following authorities granted at minimum to connect to the remote queue manager:

- READ access to the hlq.DISPLAY.QMGR profile in the MQCMDS class.
- UPDATE access to the hlq.ALTERNATE.USER.userid in the MQADMIN or MXADMIN class for each user with the MQWebUser role that connects to the queue manager
- UPDATE access to the hlq.SYSTEM.REST.REPLY.QUEUE profile in the MQQUEUE or MXQUEUE class.
- UPDATE access to the hlq.SYSTEM.ADMIN.COMMAND.QUEUE profile in the MQQUEUE or MXQUEUE class.

- CONTROL access to the `hlq.CONTEXT.SYSTEM.ADMIN.COMMAND.QUEUE` profile in the MQADMIN or MXADMIN class.
- READ access to the `SYSTEM.REST.REPLY.QUEUE.TZ` profile in the MQQUEUE or MXQUEUE class.
- ALTER access to the `hlq.DEFINE.QLOCAL` and `hlq.DELETE.QLOCAL` profiles in the MQCMDS class.
- ALTER access to the `hlq.QUEUE.SYSTEM.REST.REPLY.QUEUE.TZ` profile in the MQADMIN or MXADMIN class.
- Other authorities to interact with objects such as queues, topics, subscriptions, and channels to administer the queue manager.

Any users with the `MQWebUser` role must have the following authorities granted at minimum:

- READ access to the `hlq.DISPLAY.QMGR` profile in the MQCMDS class.
- UPDATE access to the `hlq.SYSTEM.REST.REPLY.QUEUE` profile in the MQQUEUE or MXQUEUE class.
- UPDATE access to the `hlq.SYSTEM.ADMIN.COMMAND.QUEUE` profile in the MQQUEUE or MXQUEUE class.
- Authority to interact with objects such as queues, topics, subscriptions, and channels to administer the queue manager, as appropriate for the user.

On z/OS, define RACF profiles to give the authorized user access to the queue manager. For more information, see **RACF profiles**.

Adding a remote queue manager to the IBM MQ Console from the IBM MQ Console

You can add a remote queue manager to the IBM MQ Console from the IBM MQ Console either by using an existing CCDT, or by adding the connection information manually.

Before you begin

Before you begin this task, you must complete the steps in “IBM MQ Console: Adding a remote queue manager” on page 105 and Configuring remote queue manager connection behavior for the IBM MQ Console.

Procedure

- To add a remote queue manager and specify connection information manually:
  a) From the **Home** page, click **Connect remote queue manager**.

     Note: The **Connect remote queue manager** button appears only after the prerequisite steps identified in Before you begin have been completed.
  b) Specify the name of the remote queue manager.
  c) Optionally, specify a unique name for the queue manager. If you do not specify a unique name, the actual name is used with the prefix "remote-" added.
  d) Select **Manual Entry**.
  e) Enter the name of the client connection channel that the connection uses.
  f) Specify the name of the host where the remote queue manager is running. If remote IBM MQ installations are detected, the hostnames are displayed and you can select the host of the remote queue manager that you want to connect to. In some network configurations, it is not possible to detect remote IBM MQ instances. In this case, add the hostname and port manually.
  g) Click **Next** to move to the user page and optionally specify a username and password to connect to the remote queue manager. If you do not specify this information, then authentication information is taken from the remote connection configuration file.
  h) Click **Next** to move to the **Certificate** page. You can select an SSL cipher specification from the list. You can optionally paste a certificate (as a Base64 encoded public key) and this certificate is added to the global truststore.

The certificate is temporarily stored in *WLP_USER_DIR*/generated.crts/*uniqueName*-*qmgrName*.crt before it is added to the truststore. When the connection is successfully added, the certificate is deleted from this location.

i) Click **Next** to view the summary page. You can use the **Back** button to revisit previous pages and make corrections. If you are happy with the information, click **Connect** to connect to the remote queue manager.

The connection information that you specified is written to the CCDT file in your web directory. The path is *WLP_USER_DIR*/generated.ccdt/ccdt-*uniqueName*.

- To add a remote queue manager by specifying an existing CCDT:

a) Create a JSON CCDT file that contains the connection information for the remote queue manager:

– Generate a CCDT file by using the IBM MQ Console that is associated with the same installation as the queue manager that you want to connect to remotely. From the **Home** page, click the **Download connection file** tile.

– Create a JSON format CCDT file that defines the connection. For more information about creating a JSON format CCDT, see Configuring a JSON format CCDT.

The CCDT file must include the name, clientConnection, and type information. You can optionally include additional information such as transmissionSecurity information. For more information about all the CCDT channel attribute definitions, see Complete list of CCDT channel attribute definitions.

The following example shows a basic JSON CCDT file for a remote queue manager connection:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

b) Copy the JSON CCDT file to the system where the IBM MQ Console is running.

c) From the **Home** page, click **Connect remote queue manager**.

d) Specify the name of the remote queue manager.

e) Optionally, specify a unique name for the queue manager. If you do not specify a unique name, the actual name is used with the prefix "remote-" added.

f) Ensure that **Connect using a JSON CCDT** is selected.

g) Click **Browse** and select the file that contains the JSON CCDT that you want to use.

h) Click **Next** to move to the user page and optionally specify a username and password to connect to the remote queue manager. If you do not specify this information, then authentication information is taken from the remote connection configuration file.

i) Click **Next** to move to the **Certificate** page. If your CCDT specifies "transmissionSecurity" information, this is information is used. You can optionally paste a certificate (as a Base64 encoded public key) and this certificate is added to the global truststore.

The certificate is temporarily stored in *WLP_USER_DIR*/generated.crts/*uniqueName*-*qmgrName*.crt before it is added to the truststore. When the connection is successfully added, the certificate is deleted from this location.

j) Click **Next** to view the summary page. You can use the **Back** button to revisit previous pages and make corrections. If you are happy with the information, click **Connect** to connect to the remote queue manager.

Results

The remote queue manager appears in the remote queue manager list in the IBM MQ Console. If the connection is successful, you can administer the objects of the remote queue manager in the same way that you work with the objects of a local queue manager.

Adding a remote queue manager to the IBM MQ Console from the command line

You can add a remote queue manager to the IBM MQ Console by using the **setmqweb remote** command on the command line.

Before you begin

Before you begin this task, you must complete the steps in "IBM MQ Console: Adding a remote queue manager" on page 105.

Procedure

1. Create a JSON CCDT file that contains the connection information for the remote queue manager:
   - Generate a CCDT file by using the IBM MQ Console that is associated with the same installation as the queue manager that you want to connect to remotely.

     From the **Home** page, click the **Download connection file** tile.
   - Create a JSON format CCDT file that defines the connection. For more information about creating a JSON format CCDT, see Configuring a JSON format CCDT.

     The CCDT file must include the `name`, `clientConnection`, and `type` information. You can optionally include additional information such as `transmissionSecurity` information. For more information about all the CCDT channel attribute definitions, see Complete list of CCDT channel attribute definitions.

     The following example shows a basic JSON CCDT file for a remote queue manager connection:

     ```
     {
       "channel": [{
           "name": "QM1.SVRCONN",
           "clientConnection": {
             "connection": [{
                 "host": "example.com",
                 "port": 1414
               }],
             "queueManager": "QM1"
           },
           "type": "clientConnection"
         }]
     }
     ```
2. Copy the JSON CCDT file to the system where the IBM MQ Console is running.
3. From the installation that is running the IBM MQ Console, use the **setmqweb remote** command to add the remote queue manager information to the IBM MQ Console configuration.

   As a minimum to add a remote queue manager to the IBM MQ Console, you must provide the following details:
   - The queue manager name.
   - A unique name for the queue manager. This unique name differentiates this queue manager from other remote queue managers that might have the same queue manager name. The unique name is the name that is displayed in the IBM MQ Console, so specify a name that makes it clear that the queue manager is a remote queue manager, for example, `"remote-QM2"`.
   - The CCDT URL for the queue manager.

   You can specify several other options, such as the username and password to use for the remote queue manager connection, or details of the truststore and keystore. For a full list of parameters that can be specified with the **setmqweb remote** command, see setmqweb remote.

For example, to add the remote queue manager QM1, using the example CCDT file, enter the following command:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

Results

The remote queue manager appears in the remote queue manager list in the IBM MQ Console when the remote connection list is next refreshed. if the connection is successful, you can administer the objects of the remote queue manager in the same way that you work with the objects of a local queue manager.

Example

The following example sets up the remote queue manager connection for a queue manager QM1. The IBM MQ Console is authorized to administer the queue manager based on the authorization that is given to the user `exampleUser`. The credentials of this user are provided to the IBM MQ Console when the **setmqweb remote** command is used to configure the connection information for the remote queue manager.

1. On the system where the remote queue manager QM1 is, a server-connection channel and a listener are created. The listener is started, and authorization is given for user `exampleUser` to administer the queue manager. For example, on AIX, Linux, and Windows, run the following commands:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
```

   Note: The IBM MQ Appliance does not support the `setmqaut` command. If connecting to a remote queue manager on an appliance, use the MQSC command SET AUTHREC to set authorization.

2. On the system where the IBM MQ Console is running, a `QM1_ccdt.json` file is created with the following connection information:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. On the system where the IBM MQ Console is running, the connection information for the remote queue manager QM1 is added to the mqweb server. The credentials for `exampleUser` are included in the connection information:

```
setmqweb remote add -uniqueName "remote-QM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. The IBM MQ Console shows the remote queue manager QM1.

IBM MQ Console: Working with objects

Each IBM MQ queue manager has several different types of object associated with it.

About this task

You can use the console to work with the following types of IBM MQ object:

- Queues
- Events objects:
  - Topics
  - Subscriptions
- Applications objects:
  - Connections
  - App channels
  - App channel instances
- MQ network objects:
  - Connected queue managers
  - Queue manager channels
  - Queue manager channel instances

V 10.0.0 From IBM MQ 10.0.0, you can have custom filters on tables. You can use the Filter tab to create new filter combinations, and edit and delete existing filters.

Procedure

To work with an IBM MQ object:

1. In the queue manager list view, click on the queue manager that owns the objects you want to work with.
2. Click on the Queues, Events, Applications, or MQ network tab to select the type of object you want to work with.
3. Consult one of the following topics for detailed instructions for working with the objects.

IBM MQ Console: Working with queues

You can view the queues that exist for a specific queue manager in the **Queues** tab. You can add and delete queues, add and clear messages on a queue, browse messages, view and set the properties of a queue, and manage the authority records of a queue.

About this task

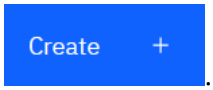
The queues view lists the queues that exist for a specific queue manager. You access the queues list by clicking on a queue manager and selecting the **Queues** tab. You can select individual queues from the list to work with.

z/OS You cannot view or edit authority records for queues on z/OS.

To use any of the move message or copy message features, you must be a user with the MQWebAdmin role.

Procedure

- To add a queue:

a) In the **Queues** tab, click the create button 

.

b) Select the type of queue that you want to create:

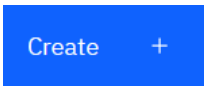
- Local queue - store messages within the queue manager it belongs to.
- Alias queue - a pointer to another queue on the same queue manager.
- Remote queue - a pointer to another queue on a different queue manager.
- Model queue - a template for a queue used when a dynamic queue manager is created.

c) Provide the required information for the type of queue you are creating. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.

d) Click **Create**. The new queue is created.

- To put messages to a queue:

a) Click the queue that you want to add messages to in the list in the queues list view. You cannot select a model queue.

b) Click the Create button 

c) Enter the message that you want to put onto the queue.

d) Click **Create**.

- To clear messages from a queue:

a) Click the local queue that you want to clear messages from in the queues list.

b) Click the Clear queue icon

c) Confirm that you want to clear the queue by clicking **Clear queue**.

- V 10.0.0

To copy all messages from one queue to another queue:

a) Click the local queue that you want to copy messages from in the queues list.

b) Click the Copy messages icon .

c) Select the queue that you want to move the message to and click **Copy messages**. You are told how many messages have been copied.

- V 10.0.0

To move all messages from one queue to another queue:

a) Click the local queue that you want to move messages from in the queues list.

b) Click the Move messages icon .

c) Select the queue that you want to move the message to and click **Move messages**. You are told how many messages have been moved.

- V 10.0.0

To delete an individual message from a queue:

a) Locate the message that you want to delete.

b) Click the menu icon next to the message and select **Delete**.

c) Confirm that you want to clear the message by clicking **Delete**.

- V 10.0.0

To move an individual message to a different queue:

a) Locate the message that you want to move.

b) Click the menu icon next to the message and select **Move**.

c) Select the queue that you want to move the message to and click **Move message**.

- V 10.0.0

  To copy an individual message to a different queue:

  a) Locate the message that you want to move.

  b) Click the menu icon next to the message and select **Copy**.

  c) Select the queue that you want to copy the message to and click **Copy message**.

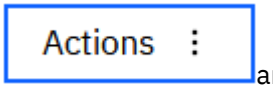
- To browse messages on a queue, click on the queue in the queues list view. A list of the messages on that queue is displayed, and you can select a message to view its properties.

  V 10.0.0

  The properties are displayed in tabs. The tabs displayed depend upon the properties that are present. You can use the **Show MQRFH2 Structure** control to toggle between displaying MQRFH2 message headers in a dedicated **MQRFH2 properties** tab or in the **Named properties** tab.

- To delete a queue:

  a) Click on the local queue that you want to delete in the queues list.

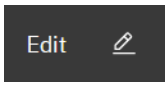
  b) Click the Actions button 

and select **Delete queue**.

  c) Confirm that you want to delete the queue by clicking **Delete**. The queue is deleted.

- To view and edit the properties of a queue:

  a) Select **View configuration** from the menu next to the queue that you want to edit.

  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For information about the properties, see Queue properties in the IBM MQ Explorer documentation

  d) Click **Save** to save your changes.

- To view and edit authority records for a queue:

  a) Select **View configuration** from the menu next to the queue that you want to edit.

  b) Click the **Security** tab.

  c) Work with the authority records as described for queue manager authority records. See "IBM MQ Console: Working with queue manager authority records" on page 102.

- To view the IBM MQ objects associated with a queue:

  a) Select **View associated objects** from the menu next to the queue that you want to view.

  b) View the objects in the panel that appears. Click the links to view more details about each of the listed objects.

  You can use the panel to view which applications are putting messages on queues and see the relationships between different queues. This can help you to identify and resolve issues.

IBM MQ Console: Working with topics

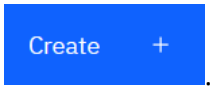
You can use the IBM MQ Console to add and delete topics, and view and set the properties of a topic.

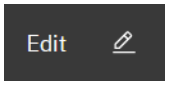
About this task

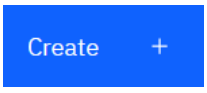
The topics view lists the topics that exist for a specific queue manager. You access the topics from the queue manager **Events** tab. You can select individual topics from the list to work with.

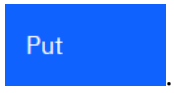
z/OS You cannot view or edit authority records for a topic on z/OS.

Procedure

- To add a topic:
  a) From the queue manager view, open the **Events** tab and click **Topics**.
  b) Click the create button 

.
  c) Provide the required information for the topic you are creating. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.
  d) Click **Create**. The new topic is created.
- To delete a topic:
  a) Click the spanner icon next to the topic that you want to delete.
  b) In the Edit queue view, click **Delete topic**.
  c) Confirm that you want to delete the topic by clicking **Delete**. The topic is deleted.
- To view and edit the properties of a topic:
  a) Click the spanner icon next to the topic that you want to edit.
  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For information about the properties, see Topic properties in the MQ Explorer documentation.
  d) Click **Save** to save your changes.
- To publish a message on a topic, you must have at least one matching subscription.
  a) Click the topic you want to publish to in the topic list.
  b) Click the matching subscription name.
  c) Click the Create button 

  d) Enter the message that you want to publish.
  e) Click the Put button 

. The message is written to all matching subscriptions.
- To subscribe to a topic, see "IBM MQ Console: Working with subscriptions" on page 117:
- To view and edit authority records for a topic:

a) Click the spanner icon next to the topic that you want to edit authority records for.

b) Click the **Security** tab.

c) Work with the authority records as described for queue manager authority records, see "IBM MQ Console: Working with queue manager authority records" on page 102.

IBM MQ Console: Working with subscriptions

You can use the IBM MQ Console to add and delete subscriptions, and view and set the properties of a subscription.

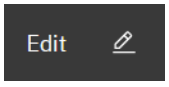
About this task

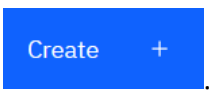
The subscriptions view lists the subscriptions that exist for a specific queue manager. You access the subscriptions from the queue manager **Events** tab. You can select individual topics from the list to work with. You can select individual subscriptions from the list to work with.

For more information about subscriptions, see Subscribers and subscriptions and DEFINE SUB.

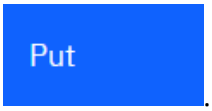
You cannot view or edit authority records for a subscription on z/OS.

Procedure

- To add a subscription:
  a) From the queue manager view, open the **Events** tab and click **Subscriptions**.
  b) Choose whether you want to create a managed or unmanaged subscription.
  c) Provide the required information for the subscription you are creating. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.
  d) Click **Create**. The new subscription is created.
- To delete a subscription:
  a) Click the spanner icon next to the subscription that you want to delete.
  b) In the Edit queue view, click **Delete subscription**.
  c) Confirm that you want to delete the subscription by clicking **Delete**. The subscription is deleted.
- To view and edit the properties of a subscription:
  a) Click the spanner icon next to the subscription that you want to edit.
  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line.
  d) Click **Save** to save your changes.
- To publish a message on the topic the subscription is subscribed to:
  a) Click the subscription whose topic you want to publish to in the subscription list.
  b) Click the Create button 

.
  c) Enter the message that you want to publish.

d) Click the Put button 

. The message is written to all subscriptions that match the topic you have published to.

IBM MQ Console: Working with queue manager channels

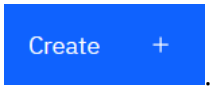
You can use the IBM MQ Console work with queue manager channels: you can add and delete queue manager channels, start and stop channels, reset and resolve channels, and ping channels. You can also view and set the properties of a queue manager channel, and manage authority records for the channel.

About this task

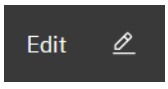
A queue manager channel is a logical communication link for transmitting messages between queue managers across a network. The queue manager channel view includes a panel that shows a quick view of how many channels are running, how many retrying, and how many stopped.

z/OS You cannot view or edit authority records for a channel on z/OS.

Procedure

- To add a queue manager channel:
  a) From the queue manager view, open the **MQ network** tab and click **Queue manager channels** and click the create button 

.
  b) Select the type of queue manager channel that you want to create and click the next button

.
  c) Provide the required information for the channel you are creating. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.
  d) Click **Create**. The new channel is created with the **inactive** status.
- To start a queue manager channel:
  a) Locate the channel that you want to start in the list.
  b) Select **Start** from the menu ⋮.
- To stop a queue manager channel:
  a) Locate the channel that you want to stop in the list.
  b) Select **Stop** from the menu ⋮.
- To view the properties of a queue manager channel:
  a) Locate the channel in the list.
  b) Select **View configurations** from the menu ⋮.
  c) Ensure the **Properties** tab is selected. To edit the properties, click the Edit button 

  d) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For more information about the properties, see Channel properties in the MQ Explorer documentation.
  e) Click **Save** to save your changes.

- To reset a queue manager channel:
  a) Locate the channel in the list.
  b) Select **Advanced** from the menu .
  c) In the **Reset** section, specify a message sequence number.
  You need to reset a channel if it will not start because the two ends disagree about the sequence number of the next message to send. The message sequence number specifies that number.
  d) Click **Reset Channel**.
- To resolve a sender or server channel:
  a) Locate the channel in the list.
  b) Select **Advanced** from the menu .
  c) In the **Resolve** section, choose whether to commit or back out the current batch of messages by clicking **Restore messages to transmission queue** or **Discard messages**.
- To ping a queue manager channel:
  a) Locate the channel in the list.
  b) Select **Ping** from the menu .
- To view and edit authority records for a queue manager channel:
  a) Locate the channel in the list.
  b) Select **View configuration** from the menu .
  c) Click the **Security** tab.
  d) Work with the authority records as described for queue manager authority records, see "IBM MQ Console: Working with queue manager authority records" on page 102.
- To delete a queue manager channel:
  a) Locate the channel in the list.
  b) Select **Configure** from the menu .
  c) Click **Delete channel**.

IBM MQ Console: Working with application channels

You can use the IBM MQ Console to work with application channels: you can add and delete channels, start and stop channels, reset and resolve channels, and ping channels. You can also view and set the properties of an application channel, and manage authority records for the channel.

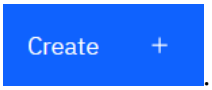
About this task

An application channel is a logical communication link, used by applications to connect to a queue manager across a network. The application channel view includes a panel that shows a quick view of how many channels are running, how many retrying, and how many stopped.

z/OS You cannot view or edit authority records for a channel on z/OS.

Procedure

- To add an application channel:

a) From the queue manager view, open the **Applications** tab and click **App channels** and click the create button 

.

b) Click the next button

.

c) Provide the required information for the channel you are creating. By default the minimum recommended properties are displayed for you to provide values for. You can view all of the available properties by selecting **Custom create**.

d) Click **Create**. The new channel is created with the **inactive** status.

- To start an application channel:

  a) Locate the channel that you want to start in the list.

  b) Select **Start** from the menu
.

- To stop an application channel:

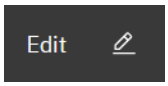
  a) Locate the channel that you want to stop in the list.

  b) Select **Stop** from the menu
.

- To view the properties of an application channel:

  a) Locate the channel in the list.

  b) Select **View configuration** from the menu
.

  c) Ensure the **Properties** tab is selected. To edit the properties, click the Edit button 

  d) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be edited only from the command line. For more information about the properties, see Channel properties in the MQ Explorer documentation.

  e) Click **Save** to save your changes.

- To reset an application channel:

  a) Locate the channel in the list.

  b) Select **Advanced** from the menu
.

  c) In the **Reset** section, specify a message sequence number.

  You need to reset a channel if it will not start because the two ends disagree about the sequence number of the next message to send. The message sequence number specifies that number.

  d) Click **Reset Channel**.

- To resolve a sender or server channel:

  a) Locate the channel in the list.

  b) Select **Advanced** from the menu
.

  c) In the **Resolve** section, choose whether to commit or back out the current batch of messages by clicking **Restore messages to transmission queue** or **Discard messages**.

- To ping a channel:

  a) Locate the channel in the list.

b) Select **Ping** from the menu ⋮.

- To view and edit authority records for an application channel:
  a) Locate the channel in the list.
  b) Select **Configure** from the menu ⋮.
  c) Click the **Security** tab.
  d) Work with the authority records as described for queue manager authority records, see "IBM MQ Console: Working with queue manager authority records" on page 102 .
- To delete an application channel:
  a) Locate the channel in the list.
  b) Select **Configure** from the menu ⋮.
  c) Click **Delete channel**.

IBM MQ Console: Working with applications

You can use the IBM MQ Console to view information about applications connected to a queue manager.

About this task

An application is connected to a queue manager across a network by using a server-conn channel. The applications view includes a panel that shows a quick view of how many applications are connected to a queue manager.

Procedure

- To view application information:
  a) From the queue manager view, open the **Applications** tab.
  b) Click **Connected applications** to open the applications view.
  c) If there are multiple instances of an application, click the down arrow to view details of each instance.
  d) Click objects in the view to get more details.

z/OS *IBM MQ Console: Working with storage classes*

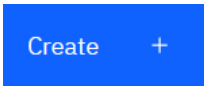
You can use the IBM MQ Console to add, view, delete and update storage classes on z/OS queue managers.

About this task

The storage classes view lists the storage classes that exist for a specific queue manager. You access **Storage classes** from the sidebar on the queue manager **Queues** tab.

See Storage classes for IBM MQ for z/OS and DEFINE STGLASS for more information about storage classes.

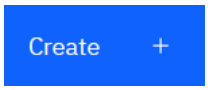
Procedure

- To add a storage class:
  a) From the queue manager view, open the **Queues** tab, and click **Storage classes**.
  b) On the **Storage classes** screen, click the **Create** 

button.

c) Provide the required information for the storage class you are creating.

By default, the minimum recommended properties you need to provide values for are displayed. You can view all of the available properties by selecting **Custom create**.

d) Click the **Create** 

button.

The new storage class is created.

- To delete a storage class:

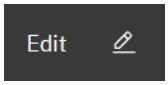
  a) Click the spanner button next to the storage class that you want to delete.

  b) In the Edit storage class view, click **Delete storage class**.

  c) Confirm that you want to delete the queue by clicking **Delete**. The storage class is deleted.

- To view and edit the properties of a storage class:

  a) Click the spanner button next to the storage class that you want to edit.

  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be set only at the time of creation.

  d) Click **Save** to save your changes.

z/OS *IBM MQ Console: Working with page sets and buffer pools*

You can use the IBM MQ Console to view page sets and buffer pools on z/OS queue managers.

About this task

The page sets and buffer pools views list the page sets and buffer pools that exist for a specific queue manager. You access the **Page sets** and **Buffer pools** views from the sidebar of the queue manager **Queues** tab

See Page sets for IBM MQ for z/OS for more information about page sets, and Buffers and buffer pools for IBM MQ for z/OS for more information about buffer pools.

Procedure

- To view the properties of a page set

  Click the spanner button next to the page set that you want to view.

- To view the properties of a buffer pool

  Click the spanner button next to the buffer pool that you want to view.

z/OS *IBM MQ Console: Working with coupling facility structures*

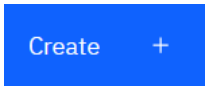
You can use the IBM MQ Console to add, view, delete and update coupling facility structures on z/OS queue managers.

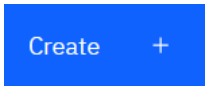
About this task

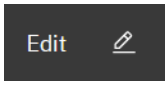
The coupling facility structures view lists the coupling facility structures that exist for a specific queue manager. You access **Coupling facility structures** from the sidebar on the queue manager **Queues** tab.

For more information about coupling facility structures, see Where are shared queue messages held?.

Procedure

- To add a coupling facility structure:
  a) From the queue manager view, open the **Queues** tab, and click **Coupling facility structures**.
  b) On the **Coupling facility structures** screen, click the **Create** 

button.
  c) Provide the required information for the coupling facility structure that you are creating.
  By default, the minimum recommended properties that you need to provide values for are displayed. You can view all of the available properties by selecting **Custom create**.
  d) Click the **Create** 

button.
  The new coupling facility structure is created.
- To delete a coupling facility structure:
  a) Click the row or select **View configuration** from the **Options** menu next to the coupling facility structure that you want to delete.
  b) In the Edit coupling facility structure view, click **Delete coupling facility structure**.
  c) Confirm that you want to delete the coupling facility structure by clicking **Delete**. The coupling facility structure is deleted.
- To view and edit the properties of a coupling facility structure:
  a) Click the row or select **View configuration** from the **Options** menu next to the coupling facility structure that you want to edit.
  b) Click the Edit button 

  c) Edit the properties as required. If the property text box is disabled, the property is read-only, or can be set only at the time of creation.
  d) Click **Save** to save your changes.

z/OS *IBM MQ Console: Working with shared message data sets*

You can use the IBM MQ Console to view shared message data sets on z/OS queue managers.

About this task

The shared message data sets view lists the shared message data sets that exist for a specific queue sharing group. You access the **Shared message data sets** view from the sidebar of the queue manager **Queues** tab.

For more information about coupling facility structures, see Where are shared queue messages held?.

Procedure

- • To view the connection status of a shared message data set to a queue manager, click the information button ⓘ next to the shared message data set that you want to view the connection status for.

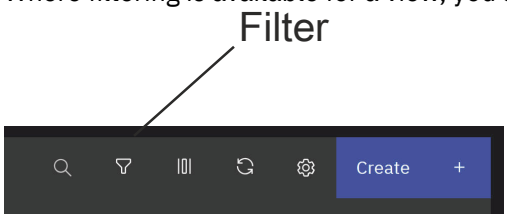
IBM MQ Console: Filtering objects displayed in tables

You can create, use and save custom table filters. This enables you to filter the data so that only the objects that you are interested in are displayed.

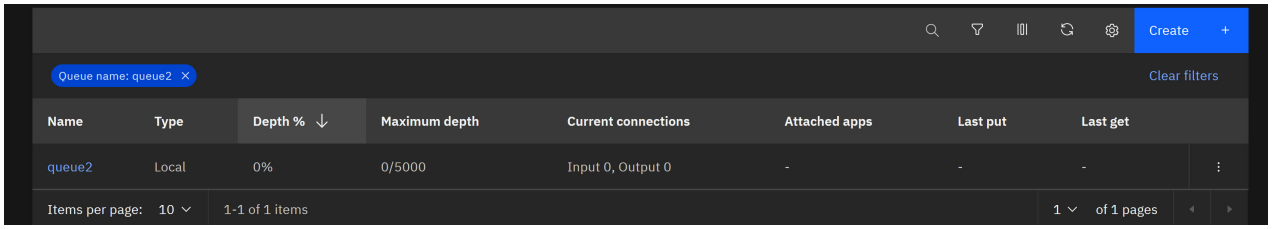
About this task

The Filters tab provides a list of all available attributes, split into categories. This is the same set used on the properties page. Each attribute has various filtering conditions available, based on their type.

Where filtering is available for a view, you see the filter icon.



When a view is filtered, the filter indications are displayed at the top of the table. For example, the following view is filtered to display the queue name queue2:



You can click the filter indicator to clear the filter.

Procedure

- To add a filter:
  a) Click the filter icon . The filter window opens.
  b) Select the required filter conditions. The attributes and settings available depend on the current view.
  c) If you want to save the filter, click the save icon next to **Saved Filters** , type a name for the filter, then click the save icon again.
  d) Click **Apply** to apply the filter to the current view.
- To delete a filter:
  a) Click the filter icon . The filter window opens.
  b) Select the filter that you want to delete from the **Saved filters** list.
  c) Click the **Delete filter group** icon to delete the filter.
  d) Click **Apply** to apply the changes.
- To view and edit a filter:

a) Click the filter icon. The filter window opens.

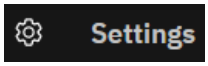
b) Select the filter that you want to delete from the **Saved filters** list.

c) Make any required changes to the filter, and click the save icon .

d) Click **Apply** to apply the changes.

IBM MQ Console settings

You can specify some general settings for the IBM MQ Console.

Click the settings icon 

to switch to the IBM MQ Console settings view.

Use the settings to control the following features:

- Automatic refresh of queue managers every ten seconds. This feature can be turned on or off.
- Whether system objects are displayed. You can specify this for all object types, or select object types individually.
- Whether trace information is collected or not.

Windows Linux Administration using the IBM MQ Explorer

The IBM MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux x86-64 only.

IBM MQ for Windows and IBM MQ for Linux x86-64 provide an administration interface called the IBM MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. Comparing command sets shows you what you can do using the IBM MQ Explorer.

The IBM MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux x86-64, by pointing the IBM MQ Explorer at the queue managers and clusters you are interested in. It can remotely connect to queue managers that are running on any supported platform including z/OS, enabling your entire messaging backbone to be viewed, explored, and altered from the console.

To configure remote IBM MQ queue managers so that IBM MQ Explorer can administer them, see “Prerequisite software and definitions for IBM MQ Explorer” on page 127.

It allows you to perform tasks, typically associated with setting up and fine-tuning the working environment for IBM MQ, either locally or remotely within a Windows or Linux x86-64 system domain.

On Linux, the IBM MQ Explorer might fail to start if you have more than one Eclipse installation. If this happens, start the IBM MQ Explorer using a different user ID to the one you use for the other Eclipse installation.

On Linux, to start the IBM MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

IBM MQ Explorer can be installed from the stand-alone IBM MQ Explorer download available from Fix Central. For more information, see Installing and uninstalling IBM MQ Explorer as a stand-alone application on Linux and Windows.

Windows Linux What you can do with the IBM MQ Explorer

You can use the IBM MQ Explorer to perform administration tasks using a series of Content Views and Property dialogs. You can also extend the IBM MQ Explorer by writing one, or more, Eclipse plugins.

IBM MQ Explorer tasks

With the IBM MQ Explorer, you can perform the following tasks:

- Create and delete a queue manager (on your local machine only).
- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of IBM MQ objects such as queues and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel, listener, queue, or service objects.
- View queue managers in a cluster.
- Check to see which applications, users, or channels have a particular queue open.
- Create a new queue manager cluster using the Create New Cluster wizard.
- Add a queue manager to a cluster using the Add Queue Manager to Cluster wizard.
- Manage the authentication information object, used with Transport Layer Security (TLS) channel security.
- Create and delete channel initiators, trigger monitors, and listeners.
- Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- Set specific services to start automatically when a queue manager is started.
- Modify the properties of queue managers.
- Change the local default queue manager.
- Create JMS objects from IBM MQ objects, and IBM MQ objects from JMS objects.
- Create a JMS Connection Factory for any of the currently supported types.
- Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- Start or stop the service trace.

Content Views and Property dialogs

You perform administration tasks using a series of Content Views and Property dialogs.

Content View
A Content View is a panel that can display the following:

- Attributes, and administrative options relating to IBM MQ itself.
- Attributes, and administrative options relating to one or more related objects.
- Attributes, and administrative options for a cluster.

Property dialogs
A property dialog is a panel that displays attributes relating to an object in a series of fields, some of which you can edit.

You navigate through the IBM MQ Explorer using the Navigator view. The Navigator allows you to select the Content View you require.

Extending the IBM MQ Explorer

The IBM MQ Explorer presents information in a style consistent with that of the Eclipse framework and the other plug-in applications that Eclipse supports.

Through extending the IBM MQ Explorer, system administrators have the ability to customize the IBM MQ Explorer to improve the way they administer IBM MQ.

For more information, see Extending MQ Explorer.

Windows Linux Deciding whether to use the IBM MQ Explorer

When deciding whether to use the IBM MQ Explorer at your installation, consider the information listed in this topic.

You need to be aware of the following points:

Object names
If you use lowercase names for queue managers and other objects with the IBM MQ Explorer, when you work with the objects using MQSC commands, you must enclose the object names in single quotation marks, or IBM MQ does not recognize them.

Large queue managers
The IBM MQ Explorer works best with small queue managers. If you have a large number of objects on a single queue manager, you might experience delays while the IBM MQ Explorer extracts the required information to present in a view.

Clusters
IBM MQ clusters can potentially contain hundreds or thousands of queue managers. The IBM MQ Explorer presents the queue managers in a cluster using a tree structure. The physical size of a cluster does not affect the speed of the IBM MQ Explorer dramatically because the IBM MQ Explorer does not connect to the queue managers in the cluster until you select them.

Setting up the IBM MQ Explorer

This section outlines the steps you need to take to set up the IBM MQ Explorer.

- "Prerequisite software and definitions for IBM MQ Explorer" on page 127
- "Security for IBM MQ Explorer" on page 127
- "Showing and hiding queue managers and clusters in IBM MQ Explorer" on page 131
- "Cluster membership and IBM MQ Explorer" on page 132
- "Data conversion for IBM MQ Explorer" on page 132

Prerequisite software and definitions for IBM MQ Explorer

Ensure that you satisfy the following requirements before trying to use the IBM MQ Explorer.

The IBM MQ Explorer can connect to remote queue managers using the TCP/IP communication protocol only.

Check that:

1. A command server is running on every remotely administered queue manager.
2. A suitable TCP/IP listener object must be running on every remote queue manager. This object can be the IBM MQ listener or, on AIX and Linux systems, the inetd daemon.
3. A server-connection channel, by default named SYSTEM.ADMIN.SVRCONN, exists on all remote queue managers.

   You can create the channel using the following MQSC command:

   ```
   DEFINE CHANNEL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(SVRCONN)
   ```

   This command creates a basic channel definition. If you want a more sophisticated definition (to set up security, for example), you need additional parameters. For more information, see DEFINE CHANNEL.
4. The system queue, SYSTEM.MQEXPLORER.REPLY.MODEL, must exist.

Security for IBM MQ Explorer

If you are using IBM MQ in an environment where it is important for you to control user access to particular objects, you might need to consider the security aspects of using the IBM MQ Explorer.

Authorization to use the IBM MQ Explorer

Any user can use the IBM MQ Explorer, but certain authorities are required to connect, access, and manage queue managers.

To perform local administrative tasks using the IBM MQ Explorer, a user is required to have the necessary authority to perform the administrative tasks. If the user is a member of the mqm group, the user has authority to perform all local administrative tasks.

To connect to a remote queue manager and perform remote administrative tasks using the IBM MQ Explorer, the user executing the IBM MQ Explorer is required to have the following authorities:

- CONNECT authority on the target queue manager object
- INQUIRE authority on the target queue manager object
- DISPLAY authority to the target queue manager object
- INQUIRE authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- DISPLAY authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- INPUT (get) authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- OUTPUT (put) authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- OUTPUT (put) authority to the queue, SYSTEM.ADMIN.COMMAND.QUEUE
- INQUIRE authority on the queue, SYSTEM.ADMIN.COMMAND.QUEUE
- Authority to perform the action selected

Note: INPUT authority relates to input to the user from a queue (a get operation). OUTPUT authority relates to output from the user to a queue (a put operation).

To connect to a remote queue manager on IBM MQ for z/OS and perform remote administrative tasks using the IBM MQ Explorer, the following must be provided:

- A RACF® profile for the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- A RACF profile for the queues, AMQ.MQEXPLORER.*

In addition, the user executing the IBM MQ Explorer is required to have the following authorities:

- RACF UPDATE authority to the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- RACF UPDATE authority to the queues, AMQ.MQEXPLORER.*
- CONNECT authority on the target queue manager object
- Authority to perform the action selected
- READ authority to all the hlq.DISPLAY.object profiles in the MQCMDS class

For information about how to grant authority to IBM MQ objects, see Giving access to an IBM MQ object on AIX, Linux, and Windows systems.

If a user attempts to perform an operation that they are not authorized to perform, the target queue manager invokes authorization failure procedures and the operation fails.

The default filter in the IBM MQ Explorer is to display all IBM MQ objects. If there are any IBM MQ objects that a user does not have DISPLAY authority to, authorization failures are generated. If authority events are being recorded, restrict the range of objects that are displayed to those objects that the user has DISPLAY authority to.

Security for connecting to remote queue managers from IBM MQ Explorer

You must secure the channel between the IBM MQ Explorer and each remote queue manager.

The IBM MQ Explorer connects to remote queue managers as an MQI client application. This means that each remote queue manager must have a definition of a server-connection channel and a suitable TCP/IP listener. If you do not secure your server connection channel it is possible for a malicious application to connect to the same server connection channel and gain access to the queue manager objects with unlimited authority. In order to secure your server connection channel either specify a non-blank value for the MCAUSER attribute of the channel, use channel authentication records, or use a security exit.

The default value of the MCAUSER attribute is the local user ID. If you specify a non-blank user name as the MCAUSER attribute of the server connection channel, all programs connecting to the queue manager using this channel run with the identity of the named user and have the same level of authority. This does not happen if you use channel authentication records.

Using a security exit with the IBM MQ Explorer

You can specify a default security exit and queue manager specific security exits using the IBM MQ Explorer.

You can define a default security exit, which can be used for all new client connections from the IBM MQ Explorer. This default exit can be overridden at the time a connection is made. You can also define a security exit for a single queue manager or a set of queue managers, which takes effect when a connection is made. You specify exits using the IBM MQ Explorer. For more information, see the IBM MQ Explorer Help.

Using the IBM MQ Explorer to connect to a remote queue manager using TLS-enabled MQI channels

The IBM MQ Explorer connects to remote queue managers using an MQI channel. If you want to secure the MQI channel using TLS security, you must establish the channel using a client channel definition table.

For information how to establish an MQI channel using a client channel definition table, see IBM MQ MQI clients.

When you have established the channel using a client channel definition table, you can use the IBM MQ Explorer to connect to a remote queue manager using TLS-enabled MQI channel, as described in "Tasks on the system that hosts the remote queue manager" on page 129 and "Tasks on the system that hosts the IBM MQ Explorer" on page 129.

Tasks on the system that hosts the remote queue manager

On the system hosting the remote queue manager, perform the following tasks:

1. Define a server connection and client connection pair of channels, and specify the appropriate value for the *SSLCIPH* attribute on the server connection on both channels. For more information about the *SSLCIPH* attribute, see Protecting channels with TLS.
2. Send the channel definition table `AMQCLCHL.TAB`, which is found in the queue manager's `@ipcc` directory, to the system hosting the IBM MQ Explorer.
3. Start a TCP/IP listener on a designated port.
4. Place both the CA and personal TLS certificates into the SSL directory of the queue manager:
   - Linux AIX `/var/mqm/qmgrs/+QMNAME+/SSL` for AIX and Linux systems.
   - Windows `C:\Program Files\IBM\MQ\qmgrs\+QMNAME+\SSL` for Windows systems.

     Where `+QMNAME+` is a token representing the name of the queue manager.
5. Create a key database file of type CMS named `key.kdb`. Stash the key database password in a file by specifying the `-stash` parameter in the **runmqakm** command that is used to create the key database.
6. Add the CA certificates to the key database created in the previous step.
7. Import the personal certificate for the queue manager into the key database.

For more detailed information about working with TLS on Windows systems, see Working with TLS on AIX, Linux, and Windows.

Tasks on the system that hosts the IBM MQ Explorer

On the system hosting the IBM MQ Explorer, perform the following tasks:

1. Create a key database file of type JKS named `key.jks`. Set a password for this key database file.

   The keystore that IBM MQ Explorer uses for TLS security must be a Java keystore (JKS) file.

2. Add the CA certificates to the key database created in the previous step.
3. Import the personal certificate for the queue manager into the key database.
4. On Windows and Linux systems, start IBM MQ Explorer by using the system menu, the `MQExplorer` executable file, or the **strmqcfg** command.
5. From the IBM MQ Explorer toolbar, click **Window -> Preferences**, then expand **IBM MQ Explorer** and click **SSL Client Certificate Stores**. Enter the name of, and password for, the JKS file created in step 1 of "Tasks on the system that hosts the IBM MQ Explorer" on page 129, in both the Trusted Certificate Store and the Personal Certificate Store, then click **OK**.
6. Close the **Preferences** window, and right-click **Queue Managers**. Click **Show/Hide Queue Managers**, and then click **Add** on the **Show/Hide Queue Managers** screen.
7. Type the name of the queue manager, and select the **Connect directly** option. Click next.
8. Select **Use client channel definition table (CCDT)** and specify the location of the channel table file that you transferred from the remote queue manager in step 2 in "Tasks on the system that hosts the remote queue manager" on page 129 on the system hosting the remote queue manager.
9. Click **Finish**. You can now access the remote queue manager from the IBM MQ Explorer.

Connecting through another queue manager with IBM MQ Explorer

The IBM MQ Explorer allows you to connect to a queue manager through an intermediate queue manager, to which the IBM MQ Explorer is already connected.

In this case, the IBM MQ Explorer puts PCF command messages to the intermediate queue manager, specifying the following:

- The *ObjectQMgrName* parameter in the object descriptor (MQOD) as the name of the target queue manager. For more information on queue name resolution, see the Name resolution.
- The *UserIdentifier* parameter in the message descriptor (MQMD) as the local userId.

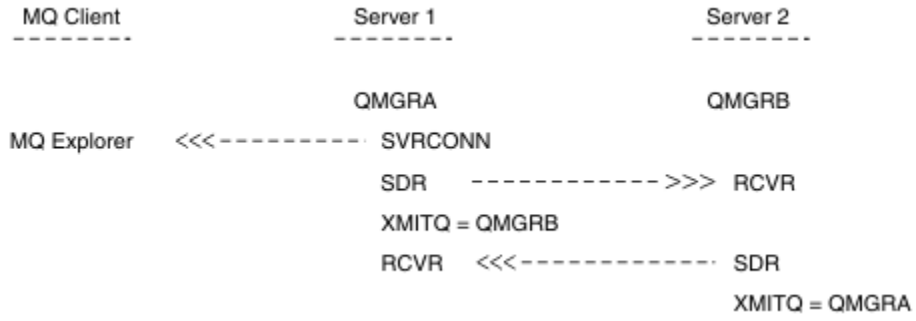
If the connection is then used to connect to the target queue manager via an intermediate queue manager, the userId is flowed in the *UserIdentifier* parameter of the message descriptor (MQMD) again. In order for the MCA listener on the target queue manager to accept this message, either the MCAUSER attribute must be set, or the userId must already exist with put authority.

The command server on the target queue manager puts messages to the transmission queue specifying the userId in the *UserIdentifier* parameter in the message descriptor (MQMD). For this put to succeed the userId must already exist on the target queue manager with put authority.

The following example shows you how to connect a queue manager, through an intermediate queue manager, to the IBM MQ Explorer.

Establish a remote administration connection to a queue manager. Verify that the:

- Queue manager on the server is active and has a server-connection channel (SVRCONN) defined.
- Listener is active.
- Command server is active.
- SYSTEM.MQ EXPLORER.REPLY.MODEL queue has been created and that you have sufficient authority.
- Queue manager listeners, command servers, and sender channels are started.



In this example:

- IBM MQ Explorer is connected to queue manager QMGRA (running on Server1) using a client connection.
- Queue manager QMGRB on Server2 can be now connected to IBM MQ Explorer through an intermediate queue manager (QMGRA)
- When connecting to QMGRB with IBM MQ Explorer, select QMGRA as the intermediate queue manager

In this situation, there is no direct connection to QMGRB from IBM MQ Explorer; the connection to QMGRB is through QMGRA.

Queue manager QMGRB on Server2 is connected to QMGRA on Server1 using sender-receiver channels. The channel between QMGRA and QMGRB must be set up in such a way that remote administration is possible; see “Configuring queue managers for remote administration” on page 201.

Showing and hiding queue managers and clusters in IBM MQ Explorer

The IBM MQ Explorer can display more than one queue manager at a time. From the Show/Hide Queue Manager panel (selectable from the menu for the Queue Managers tree node), you can choose whether you display information about another (remote) machine. Local queue managers are detected automatically.

To show a remote queue manager:

1. Right-click the **Queue Managers** tree node, then select **Show/Hide Queue Managers**.
2. Click **Add**. The Show/Hide Queue Managers panel is displayed.
3. Enter the name of the remote queue manager and the host name or IP address in the fields provided.

   The host name or IP address is used to establish a client connection to the remote queue manager using either its default server connection channel, SYSTEM.ADMIN.SVRCONN, or a user-defined server connection channel.
4. Click **Finish**.

The Show/Hide Queue Managers panel also displays a list of all visible queue managers. You can use this panel to hide queue managers from the navigation view.

If the IBM MQ Explorer displays a queue manager that is a member of a cluster, the cluster is detected, and displayed automatically.

To export the list of remote queue managers from this panel:

1. Close the Show/Hide Queue Managers panel.
2. Right-click the highest **IBM MQ** tree node in the Navigation pane of the IBM MQ Explorer, then select **Export IBM MQ Explorer Settings**
3. Click **IBM MQ Explorer > IBM MQ Explorer Settings**
4. Select **Connection Information > Remote queue managers**.
5. Select a file to store the exported settings in.
6. Finally, click **Finish** to export the remote queue manager connection information to the specified file.

To import a list of remote queue managers:

1. Right-click the highest **IBM MQ** tree node in the Navigation pane of the IBM MQ Explorer, then select **Import IBM MQ Explorer Settings**
2. Click **IBM MQ Explorer > IBM MQ Explorer Settings**
3. Click **Browse**, and navigate to the path of the file that contains the remote queue manager connection information.
4. Click **Open**. If the file contains a list of remote queue managers, the **Connection Information > Remote queue managers** box is selected.
5. Finally, click **Finish** to import the remote queue manager connection information into the IBM MQ Explorer.

Cluster membership and IBM MQ Explorer

IBM MQ Explorer requires information about queue managers that are members of a cluster.

If a queue manager is a member of a cluster, then the cluster tree node will be populated automatically.

If queue managers become members of clusters while the IBM MQ Explorer is running, then you must maintain the IBM MQ Explorer with up-to-date administration data about clusters so that it can communicate effectively with them and display correct cluster information when requested. In order to do this, the IBM MQ Explorer needs the following information:

- The name of a repository queue manager
- The connection name of the repository queue manager if it is on a remote queue manager

With this information, the IBM MQ Explorer can:

- Use the repository queue manager to obtain a list of queue managers in the cluster.
- Administer the queue managers that are members of the cluster and are on supported platforms and command levels.

Administration is not possible if:

- The chosen repository becomes unavailable. The IBM MQ Explorer does not automatically switch to an alternative repository.
- The chosen repository cannot be contacted over TCP/IP.
- The chosen repository is running on a queue manager that is running on a platform and command level not supported by the IBM MQ Explorer.

The cluster members that can be administered can be local, or they can be remote if they can be contacted using TCP/IP. The IBM MQ Explorer connects to local queue managers that are members of a cluster directly, without using a client connection.

Data conversion for IBM MQ Explorer

The IBM MQ Explorer works in CCSID 1208 (UTF-8). This enables the IBM MQ Explorer to display the data from remote queue managers correctly. Whether connecting to a queue manager directly, or by using an intermediate queue manager, the IBM MQ Explorer requires all incoming messages to be converted to CCSID 1208 (UTF-8).

An error message is issued if you try to establish a connection between the IBM MQ Explorer and a queue manager with a CCSID that the IBM MQ Explorer does not recognize.

Supported conversions are described in Code page conversion.

Windows Using the IBM MQ Taskbar application (Windows only)

The IBM MQ Taskbar application displays an icon in the Windows system tray on the server. The icon provides you with the current status of IBM MQ and a menu from which you can perform some simple actions.

On Windows, the IBM MQ icon is in the system tray on the server and is overlaid with a color-coded status symbol, which can have one of the following meanings:

Green
Working correctly; no alerts at present

Blue
Indeterminate; IBM MQ is starting up or shutting down

Yellow
Alert; one or more services are failing or have already failed

To display the menu, right-click the IBM MQ icon. From the menu you can perform the following actions:

- Click **Open** to open the IBM MQ Alert Monitor.
- Click **Exit** to exit the IBM MQ Taskbar application.
- Click **IBM MQ Explorer** to start the IBM MQ Explorer.
- Click **Stop IBM MQ** to stop IBM MQ.
- Click **About IBM MQ** to display information about the IBM MQ Alert Monitor.

Windows The IBM MQ alert monitor application (Windows only)

The IBM MQ alert monitor is an error detection tool that identifies and records problems with IBM MQ on a local machine.

The alert monitor displays information about the current status of the local installation of an IBM MQ server. It also monitors the Windows Advanced Configuration and Power Interface (ACPI) and ensures the ACPI settings are enforced.

From the IBM MQ alert monitor, you can:

- Access the IBM MQ Explorer directly
- View information relating to all outstanding alerts
- Shut down the IBM MQ service on the local machine
- Route alert messages over the network to a configurable user account, or to a Windows workstation or server

Working with local IBM MQ objects

You can administer local IBM MQ objects to support application programs that use the Message Queue Interface (MQI).

About this task

In this context, local administration means creating, displaying, changing, copying, and deleting IBM MQ objects.

In addition to the approaches described in this section, you can use the IBM MQ Explorer to administer local IBM MQ objects. For more information, see “Administration using the IBM MQ Explorer” on page 125.

Procedure

- Use the information in the following topics to help you with administering local IBM MQ objects.

- Application programs using the MQI
- “Administering IBM MQ using MQSC commands” on page 12
- “Displaying and altering queue manager attributes” on page 141
- “Working with local queues” on page 144
- “Working with alias queues” on page 157
- “Working with model queues” on page 159
- “Working with services” on page 186
- “Managing objects for triggering” on page 193

Working with queue managers

You can use control commands to start and stop a queue manager. You can use MQSC commands to display or alter queue manager attributes.

Related tasks

Creating queue managers on Multiplatforms

Multi Starting a queue manager

When you create a queue manager, you must start it to enable it to process commands or MQI calls.

About this task

You can start a queue manager by using the **strmqm** command. For a description of the **strmqm** command and its options, see strmqm.

Windows Linux Alternatively, on Windows and Linux (x86 and x86-64 platforms) systems, you can start a queue manager by using the IBM MQ Explorer.

Windows On Windows you can start a queue manager automatically when the system starts using the IBM MQ Explorer. For more information, see “Administration using the IBM MQ Explorer” on page 125.

Procedure

- To start a queue manager by using the **strmqm** command, enter the command followed by the name of the queue manager that you want to start.

  For example, to start a queue manager called QMB, enter the following command:

  ```
  strmqm QMB
  ```

  Note: You must use the **strmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

  The `strmqm` command does not return control until the queue manager has started and is ready to accept connection requests.

- Windows Linux

  To start a queue manager by using the IBM MQ Explorer, complete the following steps:

  a) Open the IBM MQ Explorer.

  b) In the Navigator view, select the queue manager.

  c) Click **Start**.

Results

The queue manager starts.

If the queue manager start-up takes more than a few seconds IBM MQ issues information messages intermittently detailing the start-up progress.

Multi Stopping a queue manager

You can use the **endmqm** command to stop a queue manager. This command provides four ways to stop a queue manager: a controlled, or quiesced, shutdown, an immediate shutdown, a preemptive shutdown, and a wait shutdown. Alternatively, on Windows and Linux, you can stop a queue manager by using the IBM MQ Explorer.

About this task

There are four ways to stop a single instance queue manager with the **endmqm** command:

Controlled (quiesced) shutdown
By default, the **endmqm** command performs a quiesced shutdown of the specified queue manager. A quiesced shutdown waits until all connected applications have disconnected, so might take a while to complete.

Immediate shutdown
For an immediate shutdown, any current MQI calls are allowed to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager.

Preemptive shutdown
The queue manager stops immediately. Use this type of shutdown only in exceptional circumstances, for example, when a queue manager does not stop as a result of a normal **endmqm** command.

Wait shutdown
This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped.

The **endmqm** command stops all instances of a multi-instance queue manager in the same way as it stops a single instance queue manager. You can issue the **endmqm** on either the active instance, or one of the standby instances of a multi-instance queue manager. However, you must issue **endmqm** on the active instance to end the queue manager.

You have the option to end the queue manager within a target time of a number of seconds that you specify, either with or without interrupting non-essential queue manager maintenance tasks, see “Ending a queue manager within a target time” on page 137.

Attention:

- Persistent messages will persist regardless of the type of shutdown used (including manually ending IBM MQ processes), whereas non-persistent messages cannot be guaranteed to survive any type of shutdown.

  Specifying the queue property NPMCLASS(HIGH) saves non-persistent messages on a best-can-do basis. Using **endmqm -t**, **endmqm -tp**, **endmqm -p**, or manually ending IBM MQ processes reduces the chances of NPMCLASS(HIGH) messages surviving an IBM MQ shutdown or restart cycle compared to **endmqm -w** or **endmqm -i**
- The combined time to both end and restart the queue manager can be longer as a result of using a more abrupt shutdown method, particularly when using the **-p** and **-tp** options.

  If the queue manager has to resort to ending IBM MQ processes to end the queue manager, then more reconciliation of queue manager state is likely to be needed when the queue manager is restarted.

For a detailed description of the **endmqm** command and its options, see endmqm.

Tip: Problems with shutting down a queue manager are often caused by applications. For example, when applications:

- Do not check MQI return codes properly
- Do not request notification of a quiesce

- Terminate without disconnecting from the queue manager (by issuing an MQDISC call)

If a problem occurs when you try to stop the queue manager, you can break out of the **endmqm** command by using Ctrl-C. You can then issue another **endmqm** command, but this time with a parameter that specifies the type of shutdown that you require.

Windows Linux As an alternative to using the **endmqm** command, on Windows and Linux, you can stop a queue manager by using the IBM MQ Explorer to carry out either a controlled or an immediate shutdown.

Procedure

- To stop the queue manager by using the **endmqm** command, enter the command followed by the appropriate parameter, if required, and the name of the queue manager that you want to stop.

  Note: You must use the **endmqm** command from the installation associated with the queue manager that you are working with. To find out which installation a queue manager is associated with, use the **dspmq** command:

  ```
  dspmq -o installation
  ```

  - To carry out a controlled (quiesced) shutdown, enter the **endmqm** command as shown in the following example, which stops a queue manager called QMB:

    ```
    endmqm QMB
    ```

    Alternatively, entering the **endmqm** command with the **-c** parameter , as shown in the following example, is equivalent to an endmqm QMB command.

    ```
    endmqm -c QMB
    ```

    In both cases, control is returned to you immediately and you are not notified when the queue manager has stopped. If you want the command to wait until all applications have stopped and the queue manager has ended before returning control to you, use the **-w** parameter instead as shown in the following example.

    ```
    endmqm -w QMB
    ```

  - To carry out an immediate shutdown, enter the **endmqm** command with the **-i** parameter as shown in the following example:

    ```
    endmqm -i QMB
    ```

  - To carry out a preemptive shutdown, enter the **endmqm** command with the **-p** parameter as shown in the following example:

    ```
    endmqm -p QMB
    ```

    Attention: A preemptive shutdown can have unpredictable consequences for connected applications. Do not use this option unless all other attempts to stop the queue manager by using a normal **endmqm** command have failed. ALW If the preemptive shutdown does not work, try "Stopping a queue manager manually" on page 138 instead.

  - To request automatic client reconnection, enter the **endmqm** command with the -r parameter. This parameter has the effect of reestablishing the connectivity of clients to other queue managers in their queue manager group.

    Note: Ending a queue manager by using the default **endmqm** command does not trigger automatic client reconnection.

- To transfer to a standby instance of a multi-instance queue manager after shutting down the active instance, enter the **endmqm** command with the `-s` parameter on the active instance of the multi-instance queue manager.
- To end the standby instance of a multi-instance queue manager and leave the active instance running, enter the **endmqm** command with the `-x` parameter on the standby instance of the multi-instance queue manager.

- Windows Linux

  On Windows and Linux, to stop the queue manager by using IBM MQ Explorer, complete the following steps:

  a) Open the IBM MQ Explorer.

  b) Select the queue manager from the Navigator View.

  c) Click **Stop**.

  The **End Queue Manager** panel is displayed.

  d) Select **Controlled**, or **Immediate**.

  e) Click **OK**.

  The queue manager stops.

Related tasks

Applying maintenance level updates to multi-instance queue managers on AIX

Applying maintenance level updates to multi-instance queue managers on Linux

Applying maintenance level updates to multi-instance queue managers on Windows

Related reference

endmqm (end queue manager)

Ending a queue manager within a target time

You can end the queue manager within a target time of a number of seconds that you specify, either with or without interrupting non-essential queue manager maintenance tasks.

There are two ways of specifying a target time when you use the **endmqm** command. The **-t** option allows all queue manager maintenance tasks to complete, which might prolong the phase of the queue manager ending. The **-tp** option interrupts non-essential queue manager maintenance tasks if necessary to comply with the specified target time.

Non-essential maintenance tasks include: queue file compaction, and persisting NPMCLASS(HIGH) messages. In the rest of this page, the word 'housekeeping' is used.

Depending on your application usage patterns, queue file compaction can take a long time, so if your primary goal is ending the queue manager quickly, then use the **-tp** option.

When you specify a target time, the shutdown type of **-w**, **-i**, or **-p** indicates the starting shutdown type.

Note: An `immediate` shutdown is still orderly, differing from a `controlled` shutdown primarily in the way that any running applications are quiesced. An `immediate` shutdown still performs housekeeping. A time limited shutdown quits these actions when they interfere with meeting the target time.

The queue manager escalates the shutdown type as necessary, in an attempt to meet the target time. For example:

- A 10 second **-t** target starting at **-w** might be seven seconds quiescing, two seconds immediate shutdown of the queue manager, including housekeeping, then immediate shutdown without further housekeeping:

```
endmqm  -w -t 10 queue_manager
```

- A 10 second **-tp** target might be seven seconds quiescing, two seconds immediate shutdown of the queue manager, including housekeeping, one second immediate shutdown without further housekeeping, then start ending IBM MQ processes:

```
endmqm -c -tp 10 queue_manager
```

- A two second **-tp** target at **-i** might be one second immediate shutdown of the queue manager, including housekeeping, one second immediate shutdown without further housekeeping, then start ending IBM MQ processes:

```
endmqm -i -tp 2 queue_manager
```

- A one second target at **-w** could be 0.1 seconds at `wait`, for example, just long enough to send out IBM MQ return codes to connected applications, 0.9 seconds immediate shutdown of the queue manager, including housekeeping, then immediate shutdown without further housekeeping; then start ending IBM MQ processes.

Related reference

endmqm (end queue manager)

ALW Stopping a queue manager manually

If the standard methods for stopping and removing a queue manager fail, you can try to stop the queue manager manually.

About this task

The standard way of stopping queue managers is by using the **endmqm** command, as described in "Stopping a queue manager" on page 135. If you are unable to stop a queue manager in the standard way, you can try to stop a queue manager manually. The way in which you do this depends on which platform you are using.

Procedure

- Windows

  To stop a queue manager on Windows, see "Stopping a queue manager manually on Windows" on page 138.

- Linux AIX

  To stop a queue manager on AIX or Linux, see "Stopping a queue manager manually on AIX and Linux" on page 139.

Related tasks

Creating and managing queue managers on Multiplatforms

Related reference

endmqm

Windows *Stopping a queue manager manually on Windows*

If you are unable to stop a queue manager on Windows by using the **endmqm** command, you can try to stop the queue manager manually by ending any processes that are running and stopping the IBM MQ service.

About this task

Tip: The Windows Task Manager and the **tasklist** command give limited information about tasks. For more information to help to determine which processes relate to a particular queue manager, consider using a tool such as *Process Explorer* (`procexp.exe`), which is available for download from the Microsoft website at http://www.microsoft.com.

To stop a queue manager on Windows, complete the following steps.

Procedure

1. List the names (IDs) of the processes that are running, by using the Windows Task Manager.
2. End the processes by using Windows Task Manager, or the **taskkill** command, in the following order (if they are running):

Table 7. Windows processes to be stopped if running

Process name	Description
AMQZMUC0	Critical process manager
AMQZXMA0	Execution controller
AMQZFUMA	OAM process
AMQZLAA0	LQM agents
AMQZLSA0	LQM agents
AMQZMUF0	Utility Manager
AMQZMGR0	Process controller
AMQZMUR0	Restartable process manager
AMQFQPUB	Publish Subscribe process
AMQFCXBA	Broker worker process
AMQRMPPA	Process pooling process
AMQCRSTA	Non-threaded responder job process
AMQCRS6B	LU62 receiver channel and client connection
AMQRRMFA	The repository process (for clusters)
AMQPCSEA	The command server
RUNMQTRM	Invoke a trigger monitor for a server
RUNMQDLQ	Invoke dead-letter queue handler
RUNMQCHI	The channel initiator process
RUNMQLSR	The channel listener process
AMQXSSVN	Shared memory servers

3. Stop the IBM MQ service from **Administration tools** > **Services** on the Windows Control Panel.
4. If you have tried all methods and the queue manager has not stopped, reboot your system.

Linux AIX *Stopping a queue manager manually on AIX and Linux*

If you are unable to stop a queue manager on AIX or Linux by using the **endmqm** command, you can try to stop the queue manager manually by ending any processes that are running and stopping the IBM MQ service.

About this task

To stop a queue manager on AIX or Linux, complete the following steps.

If you stop the queue manager manually, FFST might be taken, and FDC files placed in `/var/mqm/errors`. This should not be regarded as a defect in the queue manager.

The queue manager will restart normally, even after you have stopped it using this method of stopping it manually.

Procedure

1. Find the process identifiers (PIDs) of the queue manager programs that are still running by using the **ps** command.

   For example, if the queue manager is called QMNAME, use the following command:

   ```
   ps -ef | grep QMNAME
   ```

2. End any queue manager processes that are still running by using the **kill** command, specifying the PIDs discovered by using the **ps** command.

   To end a process, use either **kill -KILL <pid>** or the equivalent **kill -9 <pid>** command.

   You have to work through the PIDs you want to kill, one by one, issuing that command each time.

   Important: If you use any signal other than **9(SIGKILL)** the process probably will not stop and you will get unpredictable results.

   End the processes in the following order:

Table 8. AIX and Linux processes to be stopped if running

Process name	Description
amqzmuc0	Critical process manager
amqzxma0	Execution controller
amqzfuma	OAM process
amqzlaa0	LQM agents
amqzlsa0	LQM agents
amqzmuf0	Utility Manager
amqzmur0	Restartable process manager
amqzmgr0	Process controller
amqfqpub	Publish Subscribe process
amqfcxba	Broker worker process
amqrmppa	Process pooling process
amqcrsta	Non-threaded responder job process
amqcrs6b	LU62 receiver channel and client connection
amqrrmfa	The repository process (for clusters)
amqpcsea	The command server
runmqtrm	Invoke a trigger monitor for a server
runmqdlq	Invoke dead-letter queue handler
runmqchi	The channel initiator process
runmqlsr	The channel listener process

Related tasks

“Stopping a queue manager” on page 135
You can use the **endmqm** command to stop a queue manager. This command provides four ways to stop a queue manager: a controlled, or quiesced, shutdown, an immediate shutdown, a preemptive shutdown,

and a wait shutdown. Alternatively, on Windows and Linux, you can stop a queue manager by using the IBM MQ Explorer.

Multi Restarting a queue manager

You can use the **strmqm** command to restart a queue manager, or on Windows and Linux x86-64 systems, you can restart a queue manager from IBM MQ Explorer.

About this task

You can restart a queue manager by using the **strmqm** command. For a description of the **strmqm** command and its options, see strmqm.

Windows Linux On Windows and Linux x86-64 systems, you can restart a queue manager by using the IBM MQ Explorer in the same way as for starting a queue manager.

Procedure

- To restart a queue manager by using the **strmqm** command, enter the command followed by the name of the queue manager that you want to restart.

  For example, to start a queue manager called `strmqm saturn.queue.manager`, enter the following command:

  ```
  strmqm saturn.queue.manager
  ```

- Windows Linux

  To start a queue manager by using the IBM MQ Explorer, complete the following steps:

  a) Open the IBM MQ Explorer.

  b) In the Navigator view, select the queue manager.

  c) Click **Start**.

Results

The queue manager restarts.

If the queue manager restart takes more than a few seconds IBM MQ issues information messages intermittently detailing the start-up progress.

Displaying and altering queue manager attributes

Use the **DISPLAY QMGR** MQSC command to display the queue manager parameters for a queue manager. Use the **ALTER QMGR** MQSC command to alter the queue manager parameters for a local queue manager.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Procedure

- To display the attributes of the queue manager specified on the **runmqsc** command, use the **DISPLAY QMGR** MQSC command:

  ```
  DISPLAY QMGR
  ```

  The following example shows typical output from this command:

  ```
  DISPLAY QMGR
       1 : DISPLAY QMGR
  ```

```
AMQ8408I: Display Queue Manager details.
   QMNAME(QM1)                             ACCTCONO(DISABLED)
   ACCTINT(1800)                           ACCTMQI(OFF)
   ACCTQ(OFF)                              ACTIVREC(MSG)
   ACTVCONO(DISABLED)                      ACTVTRC(OFF)
   ADVCAP(DISABLED)                        ALTDATE(2022-05-05)
   ALTTIME(14.24.34)                       AMQPCAP(NO)
   AUTHOREV(DISABLED)                      CCSID(437)
   CERTLABL(ibmwebspheremqqm1)             CERTVPOL(ANY)
   CHAD(DISABLED)                          CHADEV(DISABLED)
   CHADEXIT( )                             CHLEV(DISABLED)
   CHLAUTH(ENABLED)                        CLWLDATA( )
   CLWLEXIT( )                             CLWLLEN(100)
   CLWLMRUC(999999999)                     CLWLUSEQ(LOCAL)
   CMDEV(DISABLED)                         CMDLEVEL(930)
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)    CONFIGEV(DISABLED)
   CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
   CRDATE(2020-12-22)                      CRTIME(15.42.49)
   CUSTOM( )                               DEADQ( )
   DEFCLXQ(SCTQ)                           DEFXMITQ( )
   DESCR( )                                DISTL(YES)
   IMGINTVL(60)                            IMGLOGLN(OFF)
   IMGRCOVO(YES)                           IMGRCOVQ(YES)
   IMGSCHED(MANUAL)                        INHIBTEV(DISABLED)
   INITKEY( )                              IPADDRV(IPV4)
   LOCALEV(DISABLED)                       LOGGEREV(DISABLED)
   MARKINT(5000)                           MAXHANDS(256)
   MAXMSGL(4194304)                        MAXPROPL(NOLIMIT)
   MAXPRTY(9)                              MAXUMSGS(10000)
   MONACLS(QMGR)                           MONCHL(OFF)
   MONQ(OFF)                               PARENT( )
   PERFMEV(DISABLED)                       PLATFORM(WINDOWS10)
   PSMODE(ENABLED)                         PSCLUS(ENABLED)
   PSNPMSG(DISCARD)                        PSNPRES(NORMAL)
   PSRTYCNT(5)                             PSSYNCPT(IFPER)
   QMID(QM1_2020-12-22_15.42.49)           REMOTEEV(DISABLED)
   REPOS( )                                REPOSNL( )
   REVDNS(ENABLED)                         ROUTEREC(MSG)
   SCHINIT(QMGR)                           SCMDSERV(QMGR)
   SPLCAP(DISABLED)                        SSLCRLNL( )
   SSLCRYP( )                              SSLEV(DISABLED)
   SSLFIPS(NO)                             KEYRPWD( )
   SSLKEYR(C:\ProgramData\IBM\MQ\qmgrs\QM1\ssl\key)
   SSLRKEYC(32767)                         STATACLS(QMGR)
   STATCHL(OFF)                            STATINT(1800)
   STATMQI(OFF)                            STATQ(OFF)
   STRSTPEV(ENABLED)                       SUITEB(NONE)
   SYNCPT                                  TREELIFE(1800)
   TRIGINT(999999999)                      VERSION(09030000)
   XRCAP(NO)
```

Note: SYNCPT is a read only queue manager attribute.

The **ALL** parameter is the default on the **DISPLAY QMGR** command. It displays all the queue manager attributes. In particular, the output tells you the default queue manager name, the dead-letter queue name, and the command queue name.

You can confirm that these queues exist by entering the command:

```
DISPLAY QUEUE (SYSTEM.*)
```

This displays a list of queues that match the stem SYSTEM.*. The parentheses are required.

- To alter the attributes of the queue manager specified on the **runmqsc** command, use the MQSC command **ALTER QMGR**, specifying the attributes and values that you want to change.

  For example, use the following commands to alter the attributes of `jupiter.queue.manager`:

  ```
  runmqsc jupiter.queue.manager
  ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
  ```

  The **ALTER QMGR** command changes the dead-letter queue used, and enables inhibit events.

  Parameters not specified in the **ALTER QMGR** command result in the existing values for those parameters being left unchanged.

Related tasks
Creating queue managers on Multiplatforms
Related reference
Attributes for the queue manager
runmqsc (run MQSC commands)
DISPLAY QMGR
ALTER QMGR

Multi Deleting a queue manager

You can delete a queue manager using the **dltmqm** control command. Alternatively, on Windows and Linux systems, you can use the IBM MQ Explorer to delete a queue manager.

Before you begin

Attention:

- Deleting a queue manager is a drastic step, because you also delete all resources associated with the queue manager, including all queues and their messages and all object definitions. If you use the **dltmqm** control command, there is no displayed prompt that allows you to change your mind; when you press the Enter key all the associated resources are lost.
- Windows On Windows, deleting a queue manager also removes the queue manager from the automatic startup list (described in “Starting a queue manager” on page 134). When the command has completed, an `IBM MQ queue manager ending` message is displayed; you are not told that the queue manager has been deleted.
- Deleting a cluster queue manager does not remove it from the cluster. For more information, see the usage notes in dltmqm.

About this task

You can delete a queue manager by using the **dltmqm** control command. For a description of the **dltmqm** command and its options, see dltmqm. Ensure that only trusted administrators have the authority to use this command. (For information about security, see Setting up security on AIX, Linux, and Windows.)

Windows Linux Alternatively, on Windows and Linux (x86 and x86-64 platforms) systems, you can delete a queue manager by using the IBM MQ Explorer.

Procedure

- To delete a queue manager by using the **dltmqm** command, complete the following steps:
  a) Stop the queue manager.
  b) Issue the following command:

  ```
  dltmqm QMB
  ```

  Note: You must use the **dltmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.
- Windows Linux
  To delete a queue manager by using the IBM MQ Explorer, complete the following steps:
  a) Open the IBM MQ Explorer.
  b) In the Navigator view, select the queue manager.
  c) If the queue manager is not stopped, stop it.
  To stop the queue manager, right-click it and then click **Stop**.

d) Delete the queue manager.

To delete the queue manager, right-click it and then click **Delete**.

Results

The queue manager is deleted.

Stopping MQI channels

When you issue a STOP CHANNEL command against a server-connection channel, you can choose what method to use to stop the client-connection channel. This means that a client channel issuing an MQGET wait call can be controlled, and you can decide how and when to stop the channel.

The STOP CHANNEL command can be issued with three modes, indicating how the channel is to be stopped:

Quiesce

Stops the channel after any current messages have been processed.

If sharing conversations is enabled, the IBM MQ MQI client becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM MQ.

Force

Stops the channel immediately.

Terminate

Stops the channel immediately. If the channel is running as a process, it can terminate the channel's process, or if the channel is running as a thread, its thread.

This is a multi-stage process. If mode terminate is used, an attempt is made to stop the server-connection channel, first with mode quiesce, then with mode force, and if necessary with mode terminate. The client can receive different return codes during the different stages of termination. If the process or thread is terminated, the client receives a communication error.

The return codes returned to the application vary according to the MQI call issued, and the STOP CHANNEL command issued. The client will receive either an MQRC_CONNECTION_QUIESCING or an MQRC_CONNECTION_BROKEN return code. If a client detects MQRC_CONNECTION_QUIESCING it should try to complete the current transaction and terminate. This is not possible with MQRC_CONNECTION_BROKEN. If the client does not complete the transaction and terminate fast enough it will get CONNECTION_BROKEN after a few seconds. A STOP CHANNEL command with MODE(FORCE) or MODE(TERMINATE) is more likely to result in a CONNECTION_BROKEN than with MODE(QUIESCE).

Related concepts

Channels

Working with local queues

This section contains examples of some MQSC commands that you can use to manage local, model, and alias queues.

See MQSC commands for detailed information about these commands.

Related reference

Naming restrictions for queues

Naming restrictions for other objects

Defining a local queue with DEFINE QLOCAL

For an application, the local queue manager is the queue manager to which the application is connected. Queues managed by the local queue manager are said to be local to that queue manager. You use the MQSC command **DEFINE QLOCAL** to create a local queue.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

You use the MQSC command **DEFINE QLOCAL** to create a local queue. You can also use the default defined in the default local queue definition, or you can modify the queue characteristics from those of the default local queue.

Note: The default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE and it is created on system installation.

Procedure

- To create a local queue, enter the **DEFINE QLOCAL** command as shown in the following example.

  In this example, the **DEFINE QLOCAL** command defines a queue called ORANGE.LOCAL.QUEUE with these characteristics:

  – It is enabled for gets, enabled for puts, and operates on a priority order basis.
  – It is an *normal* queue; it is not an initiation queue or transmission queue, and it does not generate trigger messages.
  – The maximum queue depth is 5000 messages; the maximum message length is 4194304 bytes.

```
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) +
       DESCR('Queue for messages from other systems') +
       PUT(ENABLED) +
       GET(ENABLED) +
       NOTRIGGER +
       MSGDLVSQ(PRIORITY) +
       MAXDEPTH(5000) +
       MAXMSGL(4194304) +
       USAGE(NORMAL)
```

  Notes:

  1. With the exception of the value for the description, all the attribute values shown in the example are the default values. These examples are included for illustration purposes. You can omit them if you are sure that the defaults are what you want or have not been changed. See also “Displaying default object attributes with DISPLAY QUEUE” on page 146.
  2. **USAGE(NORMAL)** indicates that this queue is not a transmission queue.
  3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the **REPLACE** attribute if you want to overwrite the existing definition of a queue, but see also “Changing local queue attributes with ALTER QLOCAL or DEFINE QLOCAL” on page 147.

Related reference

DEFINE QLOCAL

Displaying default object attributes with DISPLAY QUEUE

You can use the **DISPLAY QUEUE** MQSC command to display attributes that were taken from the default object when an IBM MQ object was defined.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

When you define an IBM MQ object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called SYSTEM.DEFAULT.LOCAL.QUEUE. You can use the **DISPLAY QUEUE** command to see exactly what these attributes are.

Procedure

- To display the default object attributes for a local queue, use the following command:

  ```
  DISPLAY QUEUE (SYSTEM.DEFAULT.LOCAL.QUEUE)
  ```

  The syntax of the **DISPLAY** command is different from that of the corresponding **DEFINE** command. On the **DISPLAY** command you can give just the queue name, whereas on the **DEFINE** command you have to specify the type of the queue, that is, QLOCAL, QALIAS, QMODEL, or QREMOTE.

  You can selectively display attributes by specifying them individually. For example:

  ```
  DISPLAY QUEUE (ORANGE.LOCAL.QUEUE) +
  MAXDEPTH +
  MAXMSGL +
  CURDEPTH;
  ```

  This command displays the three specified attributes as follows:

  ```
  AMQ8409: Display Queue details.
  QUEUE(ORANGE.LOCAL.QUEUE)        TYPE(QLOCAL)
  CURDEPTH(0)               MAXDEPTH(5000)
  MAXMSGL(4194304)
  ```

  CURDEPTH is the current queue depth, that is, the number of messages on the queue. This is a useful attribute to display, because by monitoring the queue depth you can ensure that the queue does not become full.

Related reference

DISPLAY QUEUE

DEFINE queues

Copying a local queue definition with DEFINE QLOCAL

You can copy a queue definition using the **LIKE** attribute on the **DEFINE QLOCAL** MQSC command.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

You can use the **DEFINE** command with the **LIKE** attribute to creates a queue with the same attributes as the specified queue, rather than those of the system default local queue. You can also use this form of the

DEFINE command to copy a queue definition, but substitute one or more changes to the attributes of the original.

Notes:

1. When you use the **LIKE** attribute on a **DEFINE** command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you a define a local queue, without specifying **LIKE**, it is the same as:

```
DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE)
```

Procedure

- To creates a queue with the same attributes as the specified queue, rather than those of the system default local queue, enter the **DEFINE** command as shown in the following example.

  Enter the name of the queue to be copied exactly as it was entered when you created the queue. If the name contains lowercase characters, enclose the name in single quotation marks.

  This example creates a queue with the same attributes as the queue ORANGE.LOCAL.QUEUE, rather than those of the system default local queue:

```
DEFINE QLOCAL (MAGENTA.QUEUE) +
LIKE (ORANGE.LOCAL.QUEUE)
```

- To copy a queue definition, but substitute one or more changes to the attributes of the original, enter the **DEFINE** command as shown in the following example.

  This command copies the attributes of the queue ORANGE.LOCAL.QUEUE to the queue THIRD.QUEUE, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 4194304:

```
DEFINE QLOCAL (THIRD.QUEUE) +
LIKE (ORANGE.LOCAL.QUEUE) +
MAXMSGL(1024);
```

Related reference

DEFINE queues

Changing local queue attributes with ALTER QLOCAL or DEFINE QLOCAL

You can change queue attributes in two ways, by using either the **ALTER QLOCAL** or the **DEFINE QLOCAL** MQSC command with the **REPLACE** attribute.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

You can use the **REPLACE** attribute of either the **ALTER** and **DEFINE** command to replace an existing definition with the specified new definition. The difference between using **ALTER** and **DEFINE** is that **ALTER** with **REPLACE** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets all the parameters.

Procedure

- To change queue attributes, use either the **ALTER** command or the **DEFINE** command as shown in the following examples.

  In these examples, the maximum message length on the queue ORANGE.LOCAL.QUEUE is decreased to 10,000 bytes.

– Using the **ALTER** command:

```
ALTER QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000)
```

This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.

– Using the **DEFINE** command with the **REPLACE** option, for example:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000) REPLACE
```

This command changes not only the maximum message length, but also all the other attributes, which are given their default values. So, for example, if the queue was previously put inhibited, this is changed to put enabled because put enabled is the default, as specified by the queue SYSTEM.DEFAULT.LOCAL.QUEUE.

If you decrease the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Related reference
ALTER queues
ALTER QLOCAL
DEFINE queues
DEFINE QLOCAL

Clearing a local queue with CLEAR QLOCAL

You can use the **CLEAR QLOCAL** MQSC command to clear a local queue.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under sync point.
- An application currently has the queue open.

About this task

If you want to clear a local queue by using the **CLEAR QLOCAL** command, the name of the queue must be defined to the local queue manager.

Note: There is no prompt that enables you to change your mind; once you press the Enter key the messages are lost.

Procedure

To clear the messages from a local queue, use the **CLEAR QLOCAL** as shown in the following example.

In this example, all the messages are deleted from a local queue called MAGENTA.QUEUE:

```
CLEAR QLOCAL (MAGENTA.QUEUE)
```

Related reference
CLEAR QLOCAL

Deleting a local queue with DELETE QLOCAL

You can use the **DELETE QLOCAL** MQSC command to delete a local queue.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

A queue cannot be deleted if it contains uncommitted messages.

If a queue has one or more committed messages and no uncommitted messages, it can be deleted only if you specify the **PURGE** option. The deletion then goes ahead even if there are committed messages on the named queue, and these messages are also purged.

Specifying **NOPURGE** instead of **PURGE** ensures that the queue is not deleted if it contains any committed messages.

Procedure

- To delete a local queue, use the **DELETE QLOCAL** command as shown in the following example.
  This example deletes the queue PINK.QUEUE if there are no committed messages on the queue:

```
DELETE QLOCAL (PINK.QUEUE) NOPURGE
```

  This example deletes the queue PINK.QUEUE even if there are committed messages on the queue:

```
DELETE QLOCAL (PINK.QUEUE) PURGE
```

Related reference
DELETE QLOCAL

Browsing queues with the sample program

IBM MQ provides a sample queue browser that you can use to look at the contents of the messages on a queue.

About this task

The browser is supplied in both source and executable formats in the following locations, where *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

Windows On Windows, the file names and paths for the sample queue browser are as follows:

Source
MQ_INSTALLATION_PATH\tools\c\samples\

Executable
MQ_INSTALLATION_PATH\tools\c\samples\bin\amqsbcg.exe

Linux AIX On AIX and Linux, the file names and paths are as follows:

Source
MQ_INSTALLATION_PATH/samp/amqsbcg0.c

Executable
MQ_INSTALLATION_PATH/samp/bin/amqsbcg

Procedure

- To run the sample program, enter a command as shown in the following example.
  The sample program requires two input parameters, the name of the queue on which the messages are going to be browsed, and the queue manager that owns that queue. For example:

```
amqsbcg SYSTEM.ADMIN.QMGREVENT.tpp01 saturn.queue.manager
```

Results

Typical results from this command are shown in the following example:

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'SYSTEM.ADMIN.QMGR.EVENT'


 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 850
  Format : 'MQEVENT '
  Priority : 0  Persistence : 0
  MsgId : X'414D512073617475726E2E71756575650005D30033563DB8'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'saturn.queue.manager                           '
  ** Identity Context
  UserIdentifier : '            '
  AccountingToken :
   X'0000000000000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                        '
  ** Origin Context
  PutApplType    : '7'
  PutApplName    : 'saturn.queue.manager        '
  PutDate  : '19970417'    PutTime  : '15115208'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '104'

****  Message   ****

length - 104 bytes

00000000: 0700 0000 2400 0000 0100 0000 2C00 0000 '....→.......,...'
00000010: 0100 0000 0100 0000 0100 0000 AE08 0000 '................'
00000020: 0100 0000 0400 0000 4400 0000 DF07 0000 '........D.......'
00000030: 0000 0000 3000 0000 7361 7475 726E 2E71 '....0...saturn.q'
00000040: 7565 7565 2E6D 616E 6167 6572 2020 2020 'ueue.manager    '
00000050: 2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060: 2020 2020 2020 2020                     '                '

No more messages
MQCLOSE
MQDISC
```

Related reference

The Browser sample program

Enabling large queues

IBM MQ supports queues larger than 2 TB.

Windows On Windows systems, support for large files is available without any additional enablement.

Linux AIX On AIX and Linux systems, you need to explicitly enable large file support before you can create queue files of multiple gigabytes or terabytes. See your operating system documentation for information on how to do this.

Some utilities, such as tar, cannot cope with files of multiple gigabytes or terabytes. Before enabling large file support, check your operating system documentation for information on restrictions on utilities you use.

For information about planning the amount of storage you need for queues, see MQ Performance documents for platform-specific performance reports.

You can control the size of queue files using a new attribute on local and model queues. See “Modifying IBM MQ queue files” on page 151 for more information.

Multi Modifying IBM MQ queue files

You can control the size of queue files using an attribute on local and model queues. You can display the current size of a queue file, and the maximum size it is currently able to grow to (based on the block size currently in use in that file), using two queue status attributes.

Attribute used to modify queue files

The attribute on local and model queues is:

MAXFSIZE
Denotes the maximum size of the queue file used by the queue, in megabytes.

You can set or display the value of this attribute using MQSC commands, IBM MQ Explorer, or the administrative REST API. You can also display the value of this attribute in the IBM MQ Console.

See MAXFSIZE and “Changing the size of an IBM MQ queue file ” on page 152 for more information.

The PCF equivalent of this attribute is **MQIA_MAX_Q_FILE_SIZE**. See Change, Copy, and Create Queue.

The two attributes on queue status are:

CURFSIZE
Displays the current size of the queue file in megabytes, rounded up to the nearest megabyte.

You can set or display the value of this attribute using MQSC commands, IBM MQ Explorer, or the administrative REST API.

See CURFSIZE for more information.

The PCF equivalent of this attribute is **MQIA_CUR_Q_FILE_SIZE**. See Inquire Queue and Inquire Queue (response).

CURMAXFS
Indicates the current maximum size the queue file can grow to, rounded up to the nearest megabyte, given the current block size in use on a queue.

You can set or display the value of this attribute using MQSC commands, IBM MQ Explorer, or the administrative REST API.

See CURMAXFS for more information.

The PCF equivalent of this attribute is **MQIA_CUR_MAX_FILE_SIZE**. See Inquire Queue and Inquire Queue (response).

Block size and granularity

Queue files are divided into segments called blocks. To increase the maximum size of a queue file, the block size or granularity of the queue might need to be changed by the queue manager.

If a newly defined queue is created with a large **MAXFSIZE** value, the queue is created with a suitable block size. However if an existing queue has its **MAXFSIZE** value increased, for example by using the **ALTER QLOCAL** MQSC command, it might be necessary to allow the queue to be emptied in order for the queue manager to reconfigure the queue.

See “Calculating how much data an IBM MQ queue file can store” on page 153 for more information.

Attention: Some file systems and operating systems have limits on the size of the entire file system, or the size of an individual file. You should check the limits on the systems your enterprise uses.

Related reference
ALTER QUEUES
DISPLAY QUEUE
DISPLAY QSTATUS

Multi

Changing the size of an IBM MQ queue file

You can increase or decrease the maximum size of a queue file.

Before you begin

Note: The steps in this task require you to run MQSC commands:

- ALW On AIX, Linux, and Windows, you issue MQSC commands from a **runmqsc** command prompt. See Running MQSC commands interactively under **runmqsc** and Running MQSC commands from text files under **runmqsc**.
- IBM i On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. See Administration using MQSC commands on IBM i.

Before setting a new size for a queue file, use the DISPLAY QLOCAL MQSC command to see the size of the queue file you want to change. For example, issue the following command:

```
DISPLAY QLOCAL(SYSTEM.DEFAULT.LOCAL.QUEUE) MAXFSIZE
```

You receive the following output:

```
AMQ8409I: Display queue details
  QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)        TYPE(QLOCAL)
  MAXFSIZE(DEFAULT)
```

which shows that the maximum size of the queue file is the default value of 2,088,960 MB.

About this task

The following procedures show you how to:

- Reduce the maximum size that a queue file can grow to.
- Increase the maximum size that a queue file can grow to.

Attention: You should be cautious of increasing the size of queue files without considering the way that your applications are written and the possible effect on performance. Accessing messages at random in a very large queue file can be very slow.

If you are considering increasing the maximum size of a queue file beyond the default you should be cautious of using message selectors such as correlation IDs and IBM MQ classes for JMS JM 3.0 or IBM MQ classes for Jakarta Messaging selector strings. Larger queue files are better suited to first-in first-out access to the queue.

Having very large amounts of data in individual queue files should only be done on queue managers that are configured for circular logging, or where media imaging has not been enabled for the individual queue.

You should not limit the size of SYSTEM queues as this could affect the operation of the queue manager.

Procedure

1. Reduce the maximum queue file size
   a) Issue the following MQSC command to create a local file called SMALLQUEUE, with a size of 500 gigabytes :

```
DEFINE QLOCAL(SMALLQUEUE) MAXFSIZE(512000)
     2 : DEFINE QLOCAL(SMALLQUEUE) MAXFSIZE(512000)
AMQ8006I: IBM MQ queue created
```

   and you receive message: AMQ8006I:

   Note: If you configure a queue with a value smaller than the amount of data already in the file, new messages cannot be put to the queue.

   If an application attempts to put a message to a queue file that does not have sufficient space, the application receives the return code MQRC_Q_SPACE_NOT_AVAILABLE. When enough messages are destructively read from the queue, applications can begin to put new messages to the queue.
2. Increase the maximum queue file size.
   a) Issue the following MQSC command to create a local file called LARGEQUEUE, with a size of 5 terabytes:

```
DEFINE QLOCAL(LARGEQUEUE) MAXFSIZE(5242880)
     3 : DEFINE QLOCAL(LARGEQUEUE) MAXFSIZE(5242880)
AMQ8006I: IBM MQ queue created
```

Multi *Calculating how much data an IBM MQ queue file can store*

The amount of data that can be stored on a queue is limited by the size of the individual blocks the queue is divided into. Use MQSC commands to confirm block size and granularity, and check the size of a queue file.

Before you begin

Note: The steps in this task require you to run MQSC commands:

- ALW On AIX, Linux, and Windows, you issue MQSC commands from a **runmqsc** command prompt. See Running MQSC commands interactively under **runmqsc** and Running MQSC commands from text files under **runmqsc**.
- IBM i On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. See Administration using MQSC commands on IBM i.

Procedure

- Confirm block size and granularity.

  The default block size is 512 bytes. To support queue files greater than two terabytes, the queue manager will need to increase the block size.

  The block size is automatically calculated when you configure the MAXFSIZE for a queue, but the revised block size cannot be applied to the queue if the queue already has messages on it. Once a queue is empty, the queue manager automatically modifies the block size to support the configured **MAXFSIZE**.

  The **DISPLAY QSTATUS** command has a new attribute, CURMAXFS, that allows you to confirm that a queue has been modified to use a new block size.

  In the following example, the **CURMAXFS** value of 4177920 confirms that the queue file is currently able to grow to approximately four terabytes in size. If the value of **MAXFSIZE** configured on the queue

is greater than the value of **CURMAXFS**, then the queue manager is still waiting for the queue to be emptied before it reconfigures the block size of the queue file.

```
DISPLAY QSTATUS(LARGEQUEUE)  CURMAXFS
     2 : DISPLAY QSTATUS(LARGEQUEUE)  CURMAXFS
AMQ8450I: Display queue status details
  QUEUE(LARGEQUEUE)                       TYPE(QUEUE)
  CURMAXFS(4177920)                       CURDEPTH(100000)
```

- Check the size of a queue file.

  You can display the current size of a queue file on disk, in megabytes, using the CURFSIZE attribute in the **DISPLAY QSTATUS** command. This can be useful on platforms such as the IBM MQ Appliance, where it is not possible to access the file system directly.

```
DISPLAY QSTATUS(SMALLQUEUE)  CURFSIZE
     1 : DISPLAY QSTATUS(SMALLQUEUE)  CURFSIZE
AMQ8450I: Display queue status details
  QUEUE(SMALLQUEUE)                       TYPE(QUEUE)
  CURDEPTH(4024)                          CURFSIZE(10)
```

  Note: When a queue has messages removed from it, the **CURFSIZE** attribute does not decrease immediately.

  Typically, space in a queue file is only released when no applications have the queue open and there are no in-doubt messages stored on the queue. Any necessary truncation or compaction of a queue file that is loaded by the queue manager occurs during checkpoint, queue manager shutdown or whilst recording a media image of the queue.

Related reference

ALTER QUEUES

DISPLAY QSTATUS

Working with remote queues

A remote queue is a definition on a local queue manager that refers to a queue on a remote queue manager. You do not have to define a remote queue from a local position, but if you do so then applications can refer to the remote queue by its locally-defined name instead of having to specify a name that is qualified by the ID of the queue manager on which the remote queue is located.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

An application connects to a local queue manager and then issues an MQOPEN call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the target queue, the target queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an MQPUT call, specifying the handle returned from the MQOPEN call. The queue manager uses the remote queue name and the remote queue manager name in a transmission header at the start of the message. This information is used to route the message to its correct destination in the network.

As administrator, you can control the destination of the message by altering the remote queue definition.

Procedure

- Put a message on a queue owned by a remote queue manager.

  The application connects to a queue manager, for example, `saturn.queue.manager`. The target queue is owned by another queue manager.

On the MQOPEN call, the application specifies these fields:

Field value	Description
ObjectName CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the target queue and the target queue manager.
ObjectType (Queue)	Identifies this object as a queue.
ObjectQmgrName Blank or `saturn.queue.manager`	This field is optional. If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition exists.)

After this, the application issues an MQPUT call to put a message onto this queue.

On the local queue manager, you can create a local definition of a remote queue using the following MQSC commands:

```
DEFINE QREMOTE (CYAN.REMOTE.QUEUE) +
DESCR ('Queue for auto insurance requests from the branches') +
RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE) +
RQMNAME (jupiter.queue.manager) +
XMITQ (INQUOTE.XMIT.QUEUE)
```

where:

QREMOTE (CYAN.REMOTE.QUEUE)
Specifies the local name of the remote queue object. This is the name that applications connected to this queue manager must specify in the MQOPEN call to open the queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE on the remote queue manager `jupiter.queue.manager`.

DESCR ('Queue for auto insurance requests from the branches')
Provides additional text that describes the use of the queue.

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)
Specifies the name of the target queue on the remote queue manager. This is the real target queue for messages sent by applications that specify the queue name CYAN.REMOTE.QUEUE. The queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE must be defined as a local queue on the remote queue manager.

RQMNAME (jupiter.queue.manager)
Specifies the name of the remote queue manager that owns the target queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE.

XMITQ (INQUOTE.XMIT.QUEUE)
Specifies the name of the transmission queue. This is optional; if the name of a transmission queue is not specified, a queue with the same name as the remote queue manager is used.

In either case, the appropriate transmission queue must be defined as a local queue with a **Usage** attribute specifying that it is a transmission queue (USAGE(XMITQ) in MQSC commands).

- Put messages on a remote queue (alternative method).

  Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, including the remote queue manager name, as part of the MQOPEN call. In this case, you do not need a local definition of a remote queue. However, this means that applications must either know, or have access to, the name of the remote queue manager at run time.
- Use other commands with remote queues.

You can use MQSC commands to display or alter the attributes of a remote queue object, or you can delete the remote queue object. For example:

- To display the remote queue's attributes:

```
DISPLAY QUEUE (CYAN.REMOTE.QUEUE)
```

- To change the remote queue to enable puts. This does not affect the target queue, only applications that specify this remote queue:

```
ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)
```

- To delete this remote queue. This does not affect the target queue, only its local definition:

```
DELETE QREMOTE (CYAN.REMOTE.QUEUE)
```

Note: When you delete a remote queue, you delete only the local representation of the remote queue. You do not delete the remote queue itself or any messages on it.

Remote queue definitions can be used as aliases

In addition to locating a queue on another queue manager, you can also use a local definition of a remote queue for queue manager aliases and reply-to queue aliases. Both types of alias are resolved through the local definition of a remote queue. You must set up the appropriate channels for the message to arrive at its destination.

Queue manager aliases

An alias is the process by which the name of the target queue manager, as specified in a message, is modified by a queue manager on the message route. Queue manager aliases are important because you can use them to control the destination of messages within a network of queue managers.

You do this by altering the remote queue definition on the queue manager at the point of control. The sending application is not aware that the queue manager name specified is an alias.

For more information about queue manager aliases, see What are aliases?

Reply-to queue aliases

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue.

If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

A reply-to queue alias is the process by which a reply-to queue, as specified in a request message, is altered by a queue manager on the message route. The sending application is not aware that the reply-to queue name specified is an alias.

A reply-to queue alias lets you alter the name of the reply-to queue and optionally its queue manager. This in turn lets you control which route is used for reply messages.

For more information about request messages, reply messages, and reply-to queues, see Types of message and Reply-to queue and queue manager.

For more information about reply-to queue aliases, see Reply-to queue aliases and clusters.

Working with alias queues

You can define an alias queue to refer indirectly to another queue or topic.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Attention: Distribution lists do not support the use of alias queues that point to topic objects. If an alias queue points to a topic object in a distribution list, IBM MQ returns MQRC_ALIAS_BASE_Q_TYPE_ERROR.

About this task

The queue to which an alias queue refers can be any of the following:

- A local queue (see “Defining a local queue with DEFINE QLOCAL” on page 145).
- A local definition of a remote queue (see “Working with remote queues” on page 154).
- A topic.

An alias queue is not a real queue, but a definition that resolves to a real (or target) queue at run time. The alias queue definition specifies the target queue. When an application makes an MQOPEN call to an alias queue, the queue manager resolves the alias to the target queue name.

An alias queue cannot resolve to another locally defined alias queue. However, an alias queue can resolve to alias queues that are defined elsewhere in clusters of which the local queue manager is a member. See Name resolution for further information.

Alias queues are useful for:

- Giving different applications different levels of access authorities to the target queue.
- Allowing different applications to work with the same queue in different ways. (Perhaps you want to assign different default priorities or different default persistence values.)
- Simplifying maintenance, migration, and workload balancing. (Perhaps you want to change the target queue name without having to change your application, which continues to use the alias.)

For example, assume that an application has been developed to put messages on a queue called MY.ALIAS.QUEUE. It specifies the name of this queue when it makes an MQOPEN request and, indirectly, if it puts a message on this queue. The application is not aware that the queue is an alias queue. For each MQI call using this alias, the queue manager resolves the real queue name, which could be either a local queue or a remote queue defined at this queue manager.

By changing the value of the TARGET attribute, you can redirect MQI calls to another queue, possibly on another queue manager. This is useful for maintenance, migration, and load-balancing.

Procedure

- Define an alias queue.

  The following MQSC command creates an alias queue:

  ```
  DEFINE QALIAS (MY.ALIAS.QUEUE) TARGET (YELLOW.QUEUE)
  ```

  This command redirects MQI calls that specify MY.ALIAS.QUEUE to the queue YELLOW.QUEUE. The command does not create the target queue; the MQI calls fail if the queue YELLOW.QUEUE does not exist at run time.

  If you change the alias definition, you can redirect the MQI calls to another queue. For example:

  ```
  ALTER QALIAS (MY.ALIAS.QUEUE) TARGET (MAGENTA.QUEUE)
  ```

This command redirects MQI calls to another queue, MAGENTA.QUEUE.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

– Application ALPHA can put messages on YELLOW.QUEUE, but is not allowed to get messages from it.
– Application BETA can get messages from YELLOW.QUEUE, but is not allowed to put messages on it.

The following MQSC command defines an alias that is put enabled and get disabled for application ALPHA:

```
DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +
TARGET (YELLOW.QUEUE) +
PUT (ENABLED) +
GET (DISABLED)
```

The following command defines an alias that is put disabled and get enabled for application BETA:

```
DEFINE QALIAS (BETAS.ALIAS.QUEUE) +
TARGET (YELLOW.QUEUE) +
PUT (DISABLED) +
GET (ENABLED)
```

ALPHA uses the queue name ALPHAS.ALIAS.QUEUE in its MQI calls; BETA uses the queue name BETAS.ALIAS.QUEUE. They both access the same queue, but in different ways.

You can use the LIKE and REPLACE attributes when you define queue aliases, in the same way that you use these attributes with local queues.

- Use other commands with alias queues.

  You can use the appropriate MQSC commands to display or alter alias queue attributes, or to delete the alias queue object. For example:

  Use the **DISPLAY QALIAS** command to display the alias queue's attributes:

  ```
  DISPLAY QALIAS (ALPHAS.ALIAS.QUEUE)
  ```

  Use the **ALTER QALIAS** command to alter the base queue name, to which the alias resolves, where the `force` option forces the change even if the queue is open:

  ```
  ALTER QALIAS (ALPHAS.ALIAS.QUEUE) TARGET(ORANGE.LOCAL.QUEUE) FORCE
  ```

  Use the **DELETE QALIAS** command to delete this queue alias:

  ```
  DELETE QALIAS (ALPHAS.ALIAS.QUEUE)
  ```

  You cannot delete an alias queue if an application currently has the queue open.

Related concepts
Distribution lists
Related reference
ALTER QALIAS
DEFINE QALIAS
DELETE QALIAS

Working with model queues

Model queues provide a convenient method for applications to create queues as required.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

A queue manager creates a *dynamic queue* if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A *model queue* is a template that specifies the attributes of any dynamic queues created from it.

Procedure

- Define a model queue.

  You use the **DEFINE QMODEL** MQSC command to define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes, except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not.) For example:

  ```
  DEFINE QMODEL (GREEN.MODEL.QUEUE) +
  DESCR('Queue for messages from application X') +
  PUT (DISABLED) +
  GET (ENABLED) +
  NOTRIGGER +
  MSGDLVSQ (FIFO) +
  MAXDEPTH (1000) +
  MAXMSGL (2000) +
  USAGE (NORMAL) +
  DEFTYPE (PERMDYN)
  ```

  This command creates a model queue definition. From the **DEFTYPE** attribute, you can see that the actual queues created from this template are permanent dynamic queues. Any attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

  You can use the **LIKE** and **REPLACE** attributes when you define model queues, in the same way that you use them with local queues.

- Use other commands with model queues.

  You can use the appropriate MQSC commands to display or alter a model queue's attributes, or to delete the model queue object. For example:

  Use the **DISPLAY QUEUE** command to display the model queue's attributes:

  ```
  DISPLAY QUEUE (GREEN.MODEL.QUEUE)
  ```

  Use the **ALTER QMODEL** command to alter the model to enable puts on any dynamic queue created from this model:

  ```
  ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)
  ```

  Use the **DELETE QMODEL** command to delete this model queue:

  ```
  DELETE QMODEL (RED.MODEL.QUEUE)
  ```

Related reference

ALTER QMODEL

DEFINE QMODEL

DELETE QMODEL
DISPLAY QUEUE

Working with dead-letter queues

Each queue manager typically has a local queue to use as a dead-letter queue, so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You tell the queue manager about the dead-letter queue, and specify how messages found on a dead-letter queue are to be processed. Using dead-letter queues can affect the sequence in which messages are delivered, so you might choose not to use them.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is available with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required, and rename it.

A dead-letter queue has no special requirements except that:

- It must be a local queue
- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH)

Using dead-letter queues can affect the sequence in which messages are delivered, so you might choose not to use them.

Procedure

- Tell the queue manager about the dead-letter queue.

  To do this, specify a dead-letter queue name on the **crtmqm** command (`crtmqm -u DEAD.LETTER.QUEUE`, for example), or by using the **DEADQ** attribute on the **ALTER QMGR** command to specify one later. You must define the dead-letter queue before using it.
- Specify how messages found on a dead-letter queue are to be processed.

  You set the USEDLQ channel attribute to determine whether the dead-letter queue is used when messages cannot be delivered. This attribute can be configured so that some functions of the queue manager use the dead-letter queue, while other functions do not. For more information about the use of the USEDLQ channel attribute on different MQSC commands, see DEFINE CHANNEL, DISPLAY CHANNEL, ALTER CHANNEL, and DISPLAY CLUSQMGR.

  You use the IBM MQ dead-letter queue handler to specify how messages found on a dead-letter queue are to be processed or removed. See “Processing messages on an IBM MQ dead-letter queue” on page 161.

Related concepts
Dead-letter queues
Related tasks
Undelivered messages troubleshooting
Related reference
ALTER QMGR
crtmqm (create queue manager)

Processing messages on an IBM MQ dead-letter queue

To process messages on a dead-letter queue (DLQ), use the default DLQ handler that is provided by IBM MQ. The handler matches messages on the DLQ against entries in a rules table that you define.

About this task

Messages can be put on a DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH. Messages put on the DLQ by a queue manager or a message channel agent always have this header; applications putting messages on the DLQ must supply this header. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

All IBM MQ environments need a routine to process messages on the DLQ regularly. IBM MQ supplies a default routine, called the *dead-letter queue handler* (the DLQ handler), that you invoke using the **runmqdlq** MQSC command.

Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table; when a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Related tasks

Undelivered messages troubleshooting

Related reference

Dead-letter queues

z/OS ALW *Invoking the dead-letter queue handler*

Invoke the dead-letter queue (DLQ) handler using the **runmqdl**q control command. You can name the DLQ you want to process and the queue manager you want to use in two ways.

Before you begin

To run the DLQ handler you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. For the DLQ handler to put messages on queues with the authority of the user ID in the message context, you must also be authorized to assume the identity of other users.

About this task

The following examples apply to the DLQ called `ABC1.DEAD.LETTER.QUEUE`, owned by the queue manager `ABC1.QUEUE.MANAGER`.

If you do not specify the DLQ or the queue manager as shown, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The **runmqdlq** command takes its input from `stdin`. You associate the rules table with **runmqdlq** by redirecting `stdin` from the rules table.

For more information about the **runmqdlq** command, see runmqdlq.

Procedure

- You can name the DLQ and the queue manager as parameters to the **runmqdlq** command.

  For example, from the command prompt:

  ```
  runmqdlq ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER <qrule.rul
  ```

- You can name the DLQ and the queue manager in the rules table.

For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
```

Related concepts

Dead-letter queues

Related tasks

Undelivered messages troubleshooting

z/OS ALW *The sample DLQ handler **amqsdlq***

In addition to the dead-letter queue handler invoked using the **runmqdlq** command, IBM MQ provides the source of a sample DLQ handler **amqsdlq** with a function that is similar to that provided by **runmqdlq**.

You can customize **amqsdlq** to provide a DLQ handler that meets your requirements. For example, you might decide that you want a DLQ handler that can process messages without dead-letter headers. (Both the default DLQ handler and the sample, **amqsdlq**, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.

In IBM MQ for Windows, the source of amqsdlq is supplied in the directory:

```
MQ_INSTALLATION_PATH\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:

```
MQ_INSTALLATION_PATH\tools\c\samples\bin
```

In IBM MQ for UNIX and Linux systems, the source of **amqsdlq** is supplied in the directory:

MQ_INSTALLATION_PATH/samp/dlq

and the compiled version is supplied in the directory:

MQ_INSTALLATION_PATH/samp/bin

A built version of the sample program, named **amqsdlqc**, is included. You can use this to connect to a remote queue manager in client mode. To make use of **amqsdlqc** you must set one of the environment variables *MQSERVER* , **MQCHLLIB** , or **MQCHLTAB** to identify how to connect to the queue manager. For example:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/myappliance.co.uk(1414)"
```

z/OS ALW *The DLQ handler rules table*

The dead-letter queue handler rules table defines how the DLQ handler processes messages that arrive on the DLQ.

There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Related concepts

Dead-letter queues

Related tasks

Undelivered messages troubleshooting

z/OS ALW *DLQ control data*

You can include keywords in a control-data entry in a dead-letter queue handler rules table.

Note:

- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName*|' ' (default))
The name of the DLQ you want to process:

1. Any INPUTQ value you supply as a parameter to the `runmqdlq` command overrides any INPUTQ value in the rules table.
2. If you do not specify an INPUTQ value as a parameter to the `runmqdlq` command, but you **do** specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If no DLQ is specified or you specify INPUTQ(' ') in the rules table, the name of the DLQ belonging to the queue manager with the name that is supplied as a parameter to the `runmqdlq` command is used.
4. If you do not specify an INPUTQ value as a parameter to the `runmqdlq` command or as a value in the rules table, the DLQ belonging to the queue manager named on the INPUTQM keyword in the rules table is used.

INPUTQM (*QueueManagerName*|' ' (default))
The name of the queue manager that owns the DLQ named on the INPUTQ keyword:

1. Any INPUTQM value you supply as a parameter to the `runmqdlq` command overrides any INPUTQM value in the rules table.
2. If you do not specify an INPUTQM value as a parameter to the `runmqdlq` command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval*|60 (default))
The interval, in seconds, at which the DLQ handler should reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (YES (default) |NO|*nnn*)
Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES
The DLQ handler waits indefinitely.

NO
The DLQ handler ends when it detects that the DLQ is either empty or contains no messages that it can process.

nnn
The DLQ handler waits for *nnn* seconds for new work to arrive before ending, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, invoke it again using triggering. For more information about triggering, see Starting IBM MQ applications using triggers.

An alternative to including control data in the rules table is to supply the names of the DLQ and its queue manager as input parameters to the `runmqdlq` command. If you specify a value both in the rules table and as input to the `runmqdlq` command, the value specified on the `runmqdlq` command takes precedence.

All control-data attributes added in the rules table must be the **first** entry in the table.

z/OS ALW *DLQ rules (patterns and actions)*

A description of the pattern-matching keywords (those against which messages on the dead-letter queue are matched), and the action keywords (those that determine how the DLQ handler is to process a matching message). An example rule is also provided.

The pattern-matching keywords

The pattern-matching keywords, which you use to specify values against which messages on the DLQ are matched, are as follows. (All pattern-matching keywords are optional):

APPLIDAT (*ApplIdentityData*|* (default))
The *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName*|* (default))
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType*|* (default))
The *PutApplType* value, specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName*|* (default))
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName*|* (default))
The name of the queue manager of the message queue for which the message is destined.

FEEDBACK (*Feedback*|* (default))
When the *MsgType* value is MQFB_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that need confirmation of their arrival on their destination queues.

FORMAT (*Format*|* (default))
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType*|* (default))
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that need replies.

PERSIST (*Persistence*|* (default))
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify messages on the DLQ that are persistent.

REASON (*ReasonCode*|* (default))
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName*|* (default))
The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName*|* (default))
The name of the queue manager of the reply-to queue, as specified in the message descriptor, MQMD, of the message on the DLQ.

USERID (*UserIdentifier*|* (default))
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD, of the message on the DLQ.

The action keywords

The action keywords, used to describe how a matching message is to be processed, are as follows:

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Delete the message from the DLQ.

IGNORE
Leave the message on the DLQ.

RETRY
If the first attempt to put the message on its destination queue fails, try again. The RETRY keyword sets the number of tries made to implement an action. The RETRYINT keyword of the control data controls the interval between attempts.

FWD
Forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword.

FWDQ (*QueueName*|&DESTQ|&REPLYQ)
The name of the message queue to which to forward the message when ACTION (FWD) is requested.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, specify REPLYQ (?*) in the message pattern.

FWDQM (*QueueManagerName*|&DESTQM|&REPLYQM|' ' (default))
The queue manager of the queue to which to forward a message.

QueueManagerName
The name of the queue manager of the queue to which to forward a message when ACTION (FWD) is requested.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' '
FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (YES (default) |NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF (default) | CTX)
The authority with which messages should be put by the DLQ handler:

DEF
Put messages with the authority of the DLQ handler itself.

CTX
Put the messages with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (*RetryCount*|1 (default))
The number of times, in the range 1 - 999,999,999, to try an action (at the interval specified on the RETRYINT keyword of the control data). The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Example rule

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

All keywords that you can use on a rule are described in the rest of this section. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

DLQ rules table conventions

The syntax, structure and contents of the dead-letter queue handler rules table must adhere to these conventions.

The rules table must adhere to the following conventions:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included only once in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- There can be any number of blanks at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- On Windows systems, the last rule in the table must end with a carriage return/line feed character. You can achieve this by ensuring that you press the Enter key at the end of the rule, so that the last line of the table is a blank line.
- For reasons of portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last non-blank character on a line to indicate that the rule continues from the first non-blank character in the next line. Use the minus sign (-) as the last non-blank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

  ```
  APPLNAME('ABC+
  D')
  ```

  results in 'ABCD', and

  ```
  APPLNAME('ABC-
  D')
  ```

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting single quotation marks in values that are enclosed in quotation marks are not considered to be significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

    These parameters are invalid because they contain no significant characters:

    ```
    FORMAT('')
    FORMAT( )
    FORMAT()
    FORMAT
    ```

  - Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank; you can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
  - Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings that are enclosed in single quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible
`MSGTYPE('*')`	Any message type is eligible

    However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.
  - Numeric parameters must be in the range 0-999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
  - If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8 character field:

`'ABCDEFGH'`	8 characters
`'A*C*E*G*I'`	5 characters excluding asterisks

```
'*A*C*E*G*I*K*M*O   8 characters excluding asterisks
*'
```

- Enclose strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (?), underscore (_), and percent sign (%) in single quotation marks. Lowercase characters not enclosed in single quotation marks are folded to uppercase. If the string includes a quotation, use two single quotation marks to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

z/OS ALW *How the DLQ rules table is processed*

The dead-letter queue handler searches the rules table for a rule where the pattern matches a message on the DLQ.

The search begins with the first rule in the table, and continues sequentially through the table. When the DLQ handler finds a rule with a matching pattern, it takes the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it applies that rule. If the first try fails, the DLQ handler tries again until the number of tries matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors are flagged at that time. You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler restarts.
4. The DLQ handler does not alter the content of messages, the MQDLH, or the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the rules table is designed to eliminate the generation of repetitive errors during validation.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Related concepts

Dead-letter queues

Related tasks

Undelivered messages troubleshooting

z/OS ALW *An example DLQ handler rules table*

An example dead-letter queue rules table for the **runmqdlq** command, containing a single control-data entry and several rules.

```
*********************************************************************
*    An example rules table for the runmqdlq command        *
*********************************************************************
* Control data entry
* ------------------
```

```
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.
* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)
```

```
* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message.  PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Related concepts

Dead-letter queues

Related tasks

Undelivered messages troubleshooting

Related reference

runmqdlq (run dead-letter queue handler)

IBM i *Invoking the dead-letter queue handler on IBM i*

On IBM MQ for IBM i, you invoke the DLQ handler by setting the **STRMQMDLQ** command.

Before you begin

You must be authorized to access both the DLQ itself, and any message queues to which messages on the DLQ are forwarded, in order to run the DLQ handler. You must also be authorized to assume the identity of other users, for the DLQ to put messages on queues with the authority of the user ID in the message context.

Note: It is often preferable to avoid placing messages on a DLQ. For information about the use and avoidance of DLQs, see "Working with dead-letter queues" on page 160.

About this task

A *dead-letter queue* (DLQ), sometimes referred to as an *undelivered-message queue*, is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have an associated DLQ.

Queue managers, message channel agents, and applications can put messages on the DLQ. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH. Messages put on the DLQ by a queue manager or by a message channel agent always have an MQDLH. For applications putting messages on the DLQ, you must supply an MQDLH.

The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

In all IBM MQ environments, there must be a routine that runs regularly to process messages on the DLQ. IBM MQ supplies a default routine, called the *dead-letter queue handler* (the DLQ handler), that you invoke using the **STRMQMDLQ** command. A user-written *rules table* supplies instructions to the DLQ handler, for processing messages on the DLQ. That is, the DLQ handler matches messages on the DLQ against entries in the rules table. When a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Procedure

- Invoke the DLQ handler

  Use the **STRMQMDLQ** command to invoke the DLQ handler. You can name the DLQ that you want to process and the queue manager that you want to use in two ways:

  – As parameters to **STRMQMDLQ** from the command prompt. For example:

  ```
  STRMQMDLQ UDLMSGQ(ABC1.DEAD.LETTER.QUEUE) SRCMBR(QRULE) SRCFILE(library/QTXTSRC)
  MQMNAME(MY.QUEUE.MANAGER)
  ```

  – In the rules table. For example:

  ```
  INPUTQ(ABC1.DEAD.LETTER.QUEUE)
  ```

  Note: The rules table is a member within a source physical file that can take any name.

  The examples apply to the DLQ called `ABC1.DEAD.LETTER.QUEUE`, owned by the default queue manager.

  If you do not specify the DLQ or the queue manager as shown, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

Related concepts

Dead-letter queues

Related tasks

Undelivered messages troubleshooting

IBM i *The DLQ handler rules table on IBM i*

The dead-letter queue handler rules table defines how the DLQ handler processes messages that arrive on the IBM i DLQ.

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table. Note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives. You can specify only one of these.
- All keywords are optional.

INPUTQ (*QueueName*|' ' (default))

The name of the DLQ you want to process:

1. Any UDLMSGQ value (or *DFT) you specify as a parameter to the **STRMQMDLQ** command overrides any INPUTQ value in the rules table.
2. If you specify a blank UDLMSGQ value as a parameter to the **STRMQMDLQ** command, the INPUTQ value in the rules table is used.
3. If you specify a blank UDLMSGQ value as a parameter to the **STRMQMDLQ** command, and a blank INPUTQ value in the rules table, the system default dead-letter queue is used.

INPUTQM (*QueueManagerName*|' ' (default))

The name of the queue manager that owns the DLQ named on the INPUTQ keyword.

If you do not specify a queue manager, or you specify INPUTQM(' ') in the rules table, the system uses the default queue manager for the installation.

RETRYINT (*Interval*|60 (default))

The interval, in seconds, at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (YES (default) |NO|*nnn*)

Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES

Causes the DLQ handler to wait indefinitely.

NO

Causes the DLQ handler to terminate when it detects that the DLQ is either empty or contains no messages that it can process.

nnn
Causes the DLQ handler to wait for *nnn* seconds for new work to arrive before terminating, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, re-invoke it using triggering.

You can supply the name of the DLQ as an input parameter to the **STRMQMDLQ** command, as an alternative to including control data in the rules table. If any value is specified both in the rules table and on input to the **STRMQMDLQ** command, the value specified on the **STRMQMDLQ** command takes precedence.

Note: If a control-data entry is included in the rules table, it must be the first entry in the table.

IBM i *DLQ rules (patterns and actions) on IBM i*

A description of the patterns and actions for each of the IBM i dead-letter queue rules.

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make 3 attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

This section describes the keywords that you can include in a rule. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives. You can specify only one of these.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched). It then describes the action keywords (those that determine how the DLQ handler is to process a matching message).

IBM i *DLQ pattern-matching keywords on IBM i*

The pattern-matching keywords are described in an example. Use these keywords to specify values against which messages on the IBM i dead-letter queue are matched. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData*|* (default))
The *ApplIdentityData* value of the message on the DLQ, specified in the message descriptor, MQMD.

APPLNAME (*PutApplName*|* (default))
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType*|* (default))
The *PutApplType* value specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName*|* (default))
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName*|* (default))
The queue manager name for the message queue for which the message is destined.

FEEDBACK (*Feedback*|* (default))
When the *MsgType* value is MQMT_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues.

FORMAT (*Format*|* (default))
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType*|* (default))
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

PERSIST (*Persistence*|* (default))
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

REASON (*ReasonCode*|* (default))
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName*|* (default))
The reply-to queue name specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName*|* (default))
The queue manager name of the reply-to queue specified in the REPLYQ keyword.

USERID (*UserIdentifier*|* (default))
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD.

IBM i *DLQ action keywords on IBM i*

Use these dead-letter queue action keywords to determine how a matching message on the IBM i dead-letter queue is processed.

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Causes the message to be deleted from the DLQ.

IGNORE
Causes the message to be kept on the DLQ.

RETRY
Causes the DLQ handler to try again to put the message on its destination queue.

FWD
Causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The RETRYINT keyword of the control data controls the interval between attempts.

FWDQ (*QueueName*|&DESTQ|&REPLYQ)
The name of the message queue to which the message is forwarded when you select the ACTION keyword.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

You can specify REPLYQ (?*) in the message pattern to avoid error messages, when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field.

FWDQM (*QueueManagerName*|&DESTQM|&REPLYQM|' ' (default))
The queue manager of the queue to which a message is forwarded.

QueueManagerName
The queue manager name for the queue to which the message is forwarded when you select the ACTION (FWD) keyword.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' '
FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (YES (default) |NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF (default) |CTX)
The authority with which messages should be put by the DLQ handler:

DEF
Puts messages with the authority of the DLQ handler itself.

CTX
Causes the messages to be put with the authority of the user ID in the message context. You must be authorized to assume the identity of other users, if you specify PUTAUT (CTX).

RETRY (*RetryCount*|1 (default))
The number of times, in the range 1 - 999,999,999, to attempt an action (at the interval specified on the RETRYINT keyword of the control data).

Note: The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If you restart the DLQ handler, the count of attempts made to apply a rule is reset to zero.

IBM i *DLQ rules table conventions on IBM i*

The IBM i dead-letter queue rules table must adhere to specific conventions regarding its syntax, structure, and contents.

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last non-blank character on a line to indicate that the rule continues from the first non-blank character in the next line. Use the minus sign (-) as the last non-blank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

```
APPLNAME('ABC+
D')
```

results in 'ABCD'.

```
APPLNAME('ABC-
D')
```

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in values enclosed in quotation marks are not considered significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

    These parameters are invalid because they contain no significant characters:

    `FORMAT('')`

    `FORMAT( )`

    `FORMAT()`

    `FORMAT`
  - Wildcard characters are supported. You can use the question mark (?) in place of any single character, except a trailing blank. You can use the asterisk (*) in place of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
  - You cannot include wildcard characters in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can include the asterisk (*) in place of an entire numeric parameter, but the asterisk cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible
`MSGTYPE('*')`	Any message type is eligible

    However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.
  - Numeric parameters must be in the range 0-999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.

– If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8-character field:

`'ABCDEFGH'`	8 characters
`'A*C*E*G*I'`	5 characters excluding asterisks
`'*A*C*E*G*I*K*M*O*'`	8 characters excluding asterisks

– Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (?), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation mark, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

IBM i *How the DLQ rules table is processed on IBM i*

The dead-letter queue handler searches the rules table for a rule with a pattern that matches a message on the IBM i dead-letter queue.

The search begins with the first rule in the table, and continues sequentially through the table. When a rule with a matching pattern is found, the rules table attempts the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can default, so that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors flagged at that time. (Error messages issued by the DLQ handler are described in Messages and reason codes.) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option **MQPMO_PASS_ALL_CONTEXT**.
5. Consecutive syntax errors in the rules table might not be recognized, because the validation of the rules table eliminates the generation of repetitive errors.
6. The DLQ handler opens the DLQ with the **MQOO_INPUT_AS_Q_DEF** option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

IBM i *An example DLQ handler rules table on IBM i*

Example code for a dead-letter queue handler rules table on IBM i. This example rules table contains a single control-data entry and several rules.

```
*********************************************************************
*    An example rules table for the STRMQMDLQ command        *
*********************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* STRMQMDLQ, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to STRMQMDLQ,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.

* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation is always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never does things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it must be able
* to cope with the message being lost, so we can afford to
* discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We do not have the message origin,
```

```
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Ensuring that all DLQ messages are processed

The dead-letter queue handler keeps a record of all messages on the DLQ that have been seen but not removed. Ensure that the DLQ contains as few messages as possible.

About this task

If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still keeps a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are seen, even if the DLQ is defined as first-in-first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, you should ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

To enable the DLQ handler to empty the DLQ, take the following measures:

Procedure

- For messages that you would otherwise ignore, use an action that moves the messages to another queue.

  Try not to use the **ACTION (IGNORE)** command, which leaves messages on the DLQ - and remember that **ACTION (IGNORE)** is assumed for messages that are not explicitly addressed by other rules in the table. Instead, use an action that moves the messages to another queue. For example:

  ```
  ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
  ```

- Make the final rule in the table a "catch-all" to process messages that have not been addressed by earlier rules in the table.

  For example, the final rule in the table could be something like this:

  ```
  ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
  ```

  This forwards messages that fall through to the final rule in the table to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

Working with administrative topics

Use MQSC commands to manage administrative topics.

See MQSC commands for detailed information about these commands.

Related concepts

Administrative topic objects

Related tasks

“Defining an administrative topic” on page 179

Use the MQSC command **DEFINE TOPIC** to create an administrative topic. When defining an administrative topic you can optionally set each topic attribute.

"Displaying administrative topic object attributes" on page 180
Use the MQSC command **DISPLAY TOPIC** to display an administrative topic object.

"Changing administrative topic attributes" on page 180
You can change topic attributes in two ways, using either the **ALTER TOPIC** command or the **DEFINE TOPIC** command with the **REPLACE** attribute.

"Copying an administrative topic definition" on page 181
You can copy a topic definition using the LIKE attribute on the **DEFINE** command.

"Deleting an administrative topic definition" on page 182
You can use the MQSC command **DELETE TOPIC** to delete an administrative topic.

Defining an administrative topic

Use the MQSC command **DEFINE TOPIC** to create an administrative topic. When defining an administrative topic you can optionally set each topic attribute.

Before you begin

Note: The example in this task requires you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

Any attribute of the topic that is not explicitly set is inherited from the default administrative topic, SYSTEM.DEFAULT.TOPIC, that was created when the system installation was installed.

Example

For example, the **DEFINE TOPIC** command that follows, defines a topic called `ORANGE.TOPIC` with these characteristics:

- Resolves to the topic string ORANGE. For information about how topic strings can be used, see Combining topic strings.
- Any attribute that is set to ASPARENT uses the attribute as defined by the parent topic of this topic. This action is repeated up the topic tree as far as the root topic, SYSTEM.BASE.TOPIC is found. For more information, see Topic trees.

```
DEFINE TOPIC (ORANGE.TOPIC) +
TOPICSTR (ORANGE) +
DEFPRTY(ASPARENT) +
NPMSGDLV(ASPARENT)
```

Note:

- Except for the value of the topic string, all the attribute values shown are the default values. They are shown here only as an illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying administrative topic object attributes" on page 180.
- If you already have an administrative topic on the same queue manager with the name ORANGE.TOPIC, this command fails. Use the REPLACE attribute if you want to overwrite the existing definition of a topic, but see also "Changing administrative topic attributes" on page 180.

Related reference

DEFINE TOPIC

Displaying administrative topic object attributes

Use the MQSC command **DISPLAY TOPIC** to display an administrative topic object.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

This command displays all topics:

```
DISPLAY TOPIC(ORANGE.TOPIC)
```

You can selectively display attributes by specifying them individually with the **DISPLAY TOPIC** command. For example:

```
DISPLAY TOPIC(ORANGE.TOPIC) +
TOPICSTR +
DEFPRTY +
NPMSGDLV
```

This command displays the three specified attributes:

```
AMQ8633: Display topic details.
     TOPIC(ORANGE.TOPIC)                                     TYPE(LOCAL)
     TOPICSTR(ORANGE)                                        DEFPRTY(ASPARENT)
     NPMSGDLV(ASPARENT)
```

To display the topic ASPARENT values as they are used at run time, use the **DISPLAY TPSTATUS** command. For example, use:

```
DISPLAY TPSTATUS(ORANGE) DEFPRTY NPMSGDLV
```

The command displays the following details:

```
AMQ8754: Display topic status details.
TOPICSTR(ORANGE)            DEFPRTY(0)
NPMSGDLV(ALLAVAIL)
```

When you define an administrative topic, it takes any attributes that you do not specify explicitly from the default administrative topic, which is called SYSTEM.DEFAULT.TOPIC. To see what these default attributes are, use the following command:

```
DISPLAY TOPIC (SYSTEM.DEFAULT.TOPIC)
```

Related reference

DISPLAY TOPIC

DISPLAY TPSTATUS

Changing administrative topic attributes

You can change topic attributes in two ways, using either the **ALTER TOPIC** command or the **DEFINE TOPIC** command with the **REPLACE** attribute.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

If, for example, you want to change the default priority of messages delivered to a topic called ORANGE.TOPIC, to be 5, use either of the following commands:

- Using the **ALTER** command:

```
ALTER TOPIC(ORANGE.TOPIC) DEFPRTY(5)
```

  This command changes a single attribute, that of the default priority of message delivered to this topic to 5; all other attributes remain the same.

- Using the **DEFINE** command:

```
DEFINE TOPIC(ORANGE.TOPIC) DEFPRTY(5) REPLACE
```

  This command changes the default priority of messages delivered to this topic. All the other attributes are given their default values.

If you alter the priority of messages sent to this topic, existing messages are not affected. Any new message, however, uses the specified priority if not provided by the publishing application.

Related reference

ALTER TOPIC

DISPLAY TOPIC

Copying an administrative topic definition

You can copy a topic definition using the LIKE attribute on the **DEFINE** command.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

The following command creates a topic, MAGENTA.TOPIC, with the same attributes as the original topic, ORANGE.TOPIC, rather than those of the system default administrative topic. Enter the name of the topic to be copied exactly as it was entered when you created the topic. If the name contains lowercase characters, enclose the name in single quotation marks.

```
DEFINE TOPIC (MAGENTA.TOPIC) +
LIKE (ORANGE.TOPIC)
```

You can also use this form of the **DEFINE** command to copy a topic definition, but make changes to the attributes of the original. For example:

```
DEFINE TOPIC(BLUE.TOPIC) +
TOPICSTR(BLUE) +
LIKE(ORANGE.TOPIC)
```

You can also copy the attributes of the topic BLUE.TOPIC to the topic GREEN.TOPIC and specify that when publications cannot be delivered to their correct subscriber queue they are not placed onto the dead-letter queue. For example:

```
DEFINE TOPIC(GREEN.TOPIC) +
TOPICSTR(GREEN) +
LIKE(BLUE.TOPIC) +
USEDLQ(NO)
```

Related reference

DEFINE TOPIC

Deleting an administrative topic definition

You can use the MQSC command **DELETE TOPIC** to delete an administrative topic.

Before you begin

Note: The example in this task requires you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

```
DELETE TOPIC(ORANGE.TOPIC)
```

Applications will no longer be able to open the topic for publication or make new subscriptions using the object name, ORANGE.TOPIC. Publishing applications that have the topic open are able to continue publishing the resolved topic string. Any subscriptions already made to this topic continue receiving publications after the topic has been deleted.

Applications that are not referencing this topic object but are using the resolved topic string that this topic object represented, 'ORANGE' in this example, continue to work. In this case they inherit the properties from a topic object higher in the topic tree. For more information, see Topic trees.

Related reference

DELETE TOPIC

Working with subscriptions

Use MQSC commands to manage subscriptions.

About this task

Subscriptions can be one of three types, defined in the **SUBTYPE** attribute:

ADMIN
Administratively defined by a user.

PROXY
An internally created subscription for routing publications between queue managers.

API
Created programmatically, for example, using the MQI MQSUB call.

See MQSC commands for detailed information about these commands.

Defining an administrative subscription

Use the MQSC command **DEFINE SUB** to create an administrative subscription. You can also use the default defined in the default local subscription definition. Or, you can modify the subscription characteristics from those of the default local subscription, SYSTEM.DEFAULT.SUB that was created when the system was installed.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

The **DEFINE SUB** command that follows defines a subscription called ORANGE with these characteristics:

- Durable subscription, meaning that it persists over queue manager restart, with unlimited expiry.

- Receive publications made on the ORANGE topic string, with the message priorities as set by the publishing applications.
- Publications delivered for this subscription are sent to the local queue SUBQ, this queue must be defined before the definition of the subscription.

```
DEFINE SUB (ORANGE) +
TOPICSTR (ORANGE) +
DESTCLAS (PROVIDED) +
DEST (SUBQ) +
EXPIRY (UNLIMITED) +
PUBPRTY (ASPUB)
```

Note:

- The subscription and topic string name do not have to match.
- Except for the values of the destination and topic string, all the attribute values shown are the default values. They are shown here only as an illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also “Displaying attributes of subscriptions” on page 183.
- If you already have a local subscription on the same queue manager with the name ORANGE, this command fails. Use the **REPLACE** attribute if you want to overwrite the existing definition of a queue, but see also “Changing local subscription attributes” on page 184.
- If the queue SUBQ does not exist, this command fails.

Related reference

DEFINE SUB

Displaying attributes of subscriptions

You can use the **DISPLAY SUB** command to display configured attributes of any subscription known to the queue manager.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

```
DISPLAY SUB(ORANGE)
```

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY SUB(ORANGE) +
        SUBID +
        TOPICSTR +
        DURABLE
```

This command displays the three specified attributes as follows:

```
AMQ8096: IBM MQ subscription inquired.
     SUBID(414D5120414141202020202020202020EE921E4E20002A03)
     SUB(ORANGE)                                                    TOPICSTR(ORANGE)
     DURABLE(YES)
```

TOPICSTR is the resolved topic string on which this subscriber is operating. When a subscription is defined to use a topic object the topic string from that object is used as a prefix to the topic string provided when making the subscription. SUBID is a unique identifier assigned by the queue manager when a subscription is created. This is a useful attribute to display because some subscription names might be long or in a different character sets for which it might become impractical.

An alternate method for displaying subscriptions is to use the SUBID:

```
DISPLAY SUB +
        SUBID(414D5120414141202020202020202020EE921E4E20002A03) +
        TOPICSTR +
        DURABLE
```

This command gives the same output as before:

```
AMQ8096: IBM MQ subscription inquired.
 SUBID(414D5120414141202020202020202020EE921E4E20002A03)
   SUB(ORANGE)                                                          TOPICSTR(ORANGE)
   DURABLE(YES)
```

Proxy subscriptions on a queue manager are not displayed by default. To display them specify a **SUBTYPE** of PROXY or ALL.

You can use the DISPLAY SBSTATUS command to display the runtime attributes. For example, use the command:

```
DISPLAY SBSTATUS(ORANGE) NUMMSGS
```

The following output is displayed:

```
AMQ8099: IBM MQ subscription status inquired.
   SUB(ORANGE)
   SUBID(414D5120414141202020202020202020EE921E4E20002A03)
   NUMMSGS(0)
```

When you define an administrative subscription, it takes any attributes that you do not specify explicitly from the default subscription, which is called SYSTEM.DEFAULT.SUB. To see what these default attributes are, use the following command:

```
DISPLAY SUB (SYSTEM.DEFAULT.SUB)
```

Related reference

DISPLAY SUB

Changing local subscription attributes

You can change subscription attributes in two ways, using either the **ALTER SUB** command or the **DEFINE SUB** command with the **REPLACE** attribute.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

If you want to change the priority of messages delivered to a subscription called ORANGE to be 5, use either of the following commands:

- Using the **ALTER** command:

  ```
  ALTER SUB(ORANGE) PUBPRTY(5)
  ```

  This command changes a single attribute, that of the priority of messages delivered to this subscription to 5; all other attributes remain the same.

- Using the **DEFINE** command:

  ```
  DEFINE SUB(ORANGE) PUBPRTY(5) REPLACE
  ```

  This command changes not only the priority of messages delivered to this subscription, but all the other attributes which are given their default values.

If you alter the priority of messages sent to this subscription, existing messages are not affected. Any new messages, however, are of the specified priority.

Related reference
ALTER SUB
DEFINE SUB

Copying a local subscription definition

You can copy a subscription definition using the **LIKE** attribute on the **DEFINE** command.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

```
DEFINE SUB(BLUE) +
       LIKE(ORANGE)
```

You can also copy the attributes of the sub REAL to the sub THIRD.SUB, and specify that the **correlID** of delivered publications is THIRD, rather than the publishers **correlID**. For example:

```
DEFINE SUB(THIRD.SUB) +
       LIKE(BLUE) +
       DESTCORL(ORANGE)
```

Related reference
DEFINE SUB

Deleting a local subscription

You can use the MQSC command **DELETE SUB** to delete a local subscription.

Before you begin

Note: The examples in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Example

```
DELETE SUB(ORANGE)
```

You can also delete a subscription using the SUBID:

```
DELETE SUB SUBID(414D5120414141202020202020202020EE921E4E20002A03)
```

Related reference
DELETE SUB

Checking messages on a subscription

When a subscription is defined it is associated with a queue. Published messages matching this subscription are put to this queue. Use MQSC commands to check for messages currently queued for a subscription.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

Note that the following MQSC commands show only those subscriptions that received messages.

To check for messages currently queued for a subscription perform the following steps:

Procedure

1. To check for messages queued for a subscription type `DISPLAY SBSTATUS(sub_name) NUMMSGS`, see "Displaying attributes of subscriptions" on page 183.
2. If the **NUMMSGS** value is greater than zero identify the queue associated with the subscription by typing `DISPLAY SUB(sub_name)DEST`.
3. Using the name of the queue returned you can view the messages by following the technique described in "Browsing queues with the sample program" on page 149.

Related reference

DISPLAY SBSTATUS

Working with services

Service objects are a means by which additional processes can be managed as part of a queue manager. With services, you can define programs that are started and stopped when the queue manager starts and ends. IBM MQ services are always started under the user ID of the user who started the queue manager.

About this task

Service objects can be either of the following types:

Server

A server is a service object that has the parameter **SERVTYPE** specified as SERVER. A server service object is the definition of a program that is executed when a specified queue manager is started. Server service objects define programs that typically run for a long time. For example, a server service object can be used to execute a trigger monitor process, such as **runmqtrm**.

Only one instance of a server service object can run concurrently. The status of running server service objects can be monitored using the MQSC command, **DISPLAY SVSTATUS**.

Command

A command is a service object that has the parameter **SERVTYPE** specified as COMMAND. Command service objects are similar to server service objects, however multiple instances of a command service object can run concurrently, and their status cannot be monitored using the MQSC command **DISPLAY SVSTATUS**.

If the MQSC command, **STOP SERVICE**, is executed, no check is made to determine whether the program started by the MQSC command, **START SERVICE**, is still active before stopping program.

Related reference

DEFINE SERVICE

DISPLAY SVSTATUS

START SERVICE

Defining a service object

You define a service object with the MQSC command **DEFINE SERVICE**.

Before you begin

Note: This task requires you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Procedure

- Define a service object with the MQSC command **DEFINE SERVICE**.

  The attributes that you need to define are as follows:

  SERVTYPE
  Defines the type of the service object. Possible values are as follows:

  SERVER
  A server service object.

  Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the MQSC command, **DISPLAY SVSTATUS**.

  COMMAND
  A command service object.

  Multiple instances of a command service object can be executed concurrently. The status of a command service objects cannot be monitored.

  STARTCMD
  The program that is executed to start the service. A fully qualified path to the program must be specified.

  STARTARG
  Arguments passed to the start program.

  STDERR
  Specifies the path to a file to which the standard error (stderr) of the service program should be redirected.

  STDOUT
  Specifies the path to a file to which the standard output (stdout) of the service program should be redirected.

  STOPCMD
  The program that is executed to stop the service. A fully qualified path to the program must be specified.

  STOPARG
  Arguments passed to the stop program.

  CONTROL
  Specifies how the service is to be started and stopped:

  MANUAL
  The service is not to be started automatically or stopped automatically. It is controlled by use of the **START SERVICE** and **STOP SERVICE** commands. This is the default value.

  QMGR
  The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

Related tasks

"Managing services" on page 188
An instance of a service object can be started and stopped automatically by the queue manager, or started and stopped using the MQSC commands **START SERVICE** and **STOP SERVICE**.

Related reference

DEFINE SERVICE
DISPLAY SVSTATUS
START SERVICE
STOP SERVICE

Managing services

An instance of a service object can be started and stopped automatically by the queue manager, or started and stopped using the MQSC commands **START SERVICE** and **STOP SERVICE**.

Before you begin

Note: This task requires you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

Procedure

- Set the **CONTROL** parameter on the queue manager to start or stop an instance of a service object automatically, or use the MQSC commands **START SERVICE** and **STOP SERVICE** to do this manually.

  When an instance of a service object is started, a message is written to the queue manager error log containing the name of the service object and the process ID of the started process. Here is an example log entry for a server service object starting:

```
02/15/2005 11:54:24 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5028: The Server 'S1' has started. ProcessId(13031).

EXPLANATION:
The Server process has started.
ACTION:
None.
```

  Here is an example log entry for a command service object starting:

```
02/15/2005 11:53:55 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5030: The Command 'C1' has started. ProcessId(13030).

EXPLANATION:
The Command has started.
ACTION:
None.
```

  When an instance server service stops, a message is written to the queue manager error logs containing the name of the service and the process ID of the ending process. Here is an example log entry for a server service object stopping:

```
02/15/2005 11:54:54 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5029: The Server 'S1' has ended. ProcessId(13031).

EXPLANATION:
```

```
The Server process has ended.
ACTION:
None.
```

Related tasks

“Defining additional environment variables in the service.env file” on page 189
When a service is started, the environment in which the service process is started is inherited from the environment of the queue manager. It is possible to define additional environment variables to be set in the environment of the service process by adding the variables you want to define to one of the `service.env` environment override files.

Related reference

STOP SERVICE (stop a service) on Multiplatforms

START SERVICE (start a service) on Multiplatforms

Defining additional environment variables in the service.env file

When a service is started, the environment in which the service process is started is inherited from the environment of the queue manager. It is possible to define additional environment variables to be set in the environment of the service process by adding the variables you want to define to one of the `service.env` environment override files.

About this task

There are two possible files to which you can add environment variables:

- The machine scope `service.env` file
- The queue manager scope `service.env` file

Both files are processed, if available, with definitions in the queue manager scope file taking precedence over those definitions in the machine scope file.

You can specify any environment variable in the `service.env` file. For example, if the IBM MQ service runs a number of commands, it might be useful to set the **PATH** user variable in the `service.env` file.

Note: The values that you set the variable to cannot be environment variables; for example `CLASSPATH=` *%CLASSPATH%* is incorrect. Similarly, on Linux `PATH=` *$PATH* `:/opt/mqm/bin` would give unexpected results.

CLASSPATH must be capitalized, and the class path statement can contain only literals. Some services (IBM MQ Telemetry for example) set their own class path. The **CLASSPATH** defined in `service.env` is added to it.

The format of the variables defined in the `service.env` file is a list of name and value variable pairs. Each variable must be defined on a new line, and each variable is taken as it is explicitly defined, including white space.

Procedure

- Add environment variables to the machine scope `service.env` file.
  This file is located in:
  - Linux AIX `/var/mqm` on AIX and Linux systems.
  - Windows The data directory selected during installation on Windows systems.
- Add environment variables to the queue manager scope `service.env` file.
  This file is located in the queue manager data directory. For example, the location of the environment override file for a queue manager named `QMNAME` is:
  - Linux AIX On AIX and Linux systems, `/var/mqm/qmgrs/QMNAME/service.env`

– Windows On Windows systems, `C:\ProgramData\IBM\MQ\qmgrs\QMNAME\service.env`

Example of a `service.env` file

```
#********************************************************************#
#*                                                                  *#
#* <N_OCO_COPYRIGHT>                                                *#
#* Licensed Materials - Property of IBM                             *#
#*                                                                  *#
#* 63H9336                                                          *#
#* (C) Copyright IBM Corporation 2005, 2026.                        *#
#*                                                                  *#
#* <NOC_COPYRIGHT>                                                  *#
#*                                                                  *#
#********************************************************************#
#********************************************************************#
#* Module Name: service.env                                         *#
#* Type       : IBM MQ service environment file                     *#
#* Function   : Define additional environment variables to be set   *#
#*              for SERVICE programs.                               *#
#* Usage      : <VARIABLE>=<VALUE>                                  *#
#*                                                                  *#
#********************************************************************#
MYLOC=/opt/myloc/bin
MYTMP=/tmp
TRACEDIR=/tmp/trace
MYINITQ=ACCOUNTS.INITIATION.QUEUE
```

Related tasks

“Using replaceable inserts on service definitions” on page 190
You can substitute tokens in the definition of a service object. Tokens that are substituted are automatically replaced with their expanded text when the service program is executed.

Related reference

Environment variables descriptions

Using replaceable inserts on service definitions

You can substitute tokens in the definition of a service object. Tokens that are substituted are automatically replaced with their expanded text when the service program is executed.

About this task

Substitute tokens can be taken from the following list of common tokens, or from any variables that are defined in the file, `service.env`.

Procedure

- To use replaceable inserts, insert the token within + characters into any of the **STARTCMD**, **STARTARG**, **STOPCMD**, **STOPARG**, **STDOUT** or **STDERR** strings.

  For examples of this, see “Using a server service object” on page 191 and “Using a command service object” on page 193.

  The following are common tokens that can be used to substitute tokens in the definition of a service object:

  MQ_INSTALL_PATH
  The location where IBM MQ is installed.

  MQ_DATA_PATH
  The location of the IBM MQ data directory:

  – Linux AIX On AIX and Linux systems, the IBM MQ data directory location is `/var/mqm/`

– Windows On Windows systems, the location of the IBM MQ data directory is the data directory selected during the installation of IBM MQ

QMNAME
The current queue manager name.

MQ_SERVICE_NAME
The name of the service.

MQ_SERVER_PID
This token can only be used by the **STOPARG** and **STOPCMD** arguments.

For server service objects this token is replaced with the process ID of the process started by the **STARTCMD** and **STARTARG** arguments. Otherwise, this token is replaced with 0.

MQ_Q_MGR_DATA_PATH
The location of the queue manager data directory.

MQ_Q_MGR_DATA_NAME
The transformed name of the queue manager. For more information on name transformation, see Understanding IBM MQ file names.

Using a server service object

These examples shows how to define, use, and alter a server service object to start a trigger monitor or other program.

Before you begin

Note: These examples require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

These examples are written with UNIX style path separator characters, except where otherwise stated.

Procedure

1. **Define a server service object, using the DEFINE SERVICE MQSC command:**

```
DEFINE SERVICE(S1) +
CONTROL(QMGR) +
SERVTYPE(SERVER) +
STARTCMD('+MQ_INSTALL_PATH+bin/runmqtrm') +
STARTARG('-m +QMNAME+ -q ACCOUNTS.INITIATION.QUEUE') +
STOPCMD('+MQ_INSTALL_PATH+bin/amqsstop') +
STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+')
```

Where:

+MQ_INSTALL_PATH+ is a token representing the installation directory.
+QMNAME+ is a token representing the name of the queue manager.
ACCOUNTS.INITIATION.QUEUE is the initiation queue.
amqsstop is a sample program provided with IBM MQ which requests the queue manager to break all connections for the process ID. amqsstop generates PCF commands, therefore the command server must be running.
+MQ_SERVER_PID+ is a token representing the process ID passed to the stop program.

See “Using replaceable inserts on service definitions” on page 190 for a list of the common tokens.

2. **An instance of the server service object executes when the queue manager is next started. However, you can start an instance of the server service object immediately with the START SERVICE MQSC command:**

```
START SERVICE(S1)
```

3. **Display the status of the server service process, using the DISPLAY SVSTATUS MQSC command:**

```
DISPLAY SVSTATUS(S1)
```

4. **Alter the server service object and have the updates picked up by manually restarting the server service process, using the ALTER SERVICE MQSC command.**

   The server service object is altered so that the initiation queue is specified as JUPITER.INITIATION.QUEUE.

   ```
   ALTER SERVICE(S1) +
   STARTARG('-m +QMNAME+ -q JUPITER.INITIATION.QUEUE')
   ```

   Note: A running service does not pick up any updates to its service definition until it is restarted.

5. **Restart the server service process so that the alteration is picked up, using the STOP SERVICE and START SERVICE MQSC commands:**

   ```
   STOP SERVICE(S1)
   ```

   Followed by:

   ```
   START SERVICE(S1)
   ```

   The server service process is restarted and picks up the alterations made in "4" on page 192.

   Note: The MQSC command, **STOP SERVICE**, can only be used if a **STOPCMD** argument is specified in the service definition.

More examples of passing arguments

- **Define a server service object to start a program called runserv when a queue manager is started.**

  Do this using the **DEFINE SERVICE** MQSC command.

  This example is written with Windows style path separator characters.

  One of the arguments that is passed to the starting program is a string containing a space. This argument needs to be passed as a single string. To achieve this, double quotation marks are used as shown in the following command to define the command service object.

  ```
  DEFINE SERVICE(S1) SERVTYPE(SERVER) CONTROL(QMGR) +
  STARTCMD('C:\Program Files\Tools\runserv.exe') +
  STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
  STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')


  DEFINE SERVICE(S4) +
  CONTROL(QMGR) +
  SERVTYPE(SERVER) +
  STARTCMD('C:\Program Files\Tools\runserv.exe') +
  STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
  STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
  ```

  Where:

  +QMNAME+ is a token representing the name of the queue manager.
  "C:\Program Files\Tools\" is a string containing a space, which will be passed as a single string.

- **Define a server service object that can be used to automatically start the Trigger Monitor when the queue manager starts.**

  Do this using the **DEFINE SERVICE** MQSC command.

  ```
  DEFINE SERVICE(TRIG_MON_START) +
  CONTROL(QMGR) +
  SERVTYPE(SERVER) +
  STARTCMD('runmqtrm') +
  STARTARG('-m +QMNAME+ -q +IQNAME+')
  ```

Where:

+QMNAME+ is a token representing the name of the queue manager.

+IQNAME+ is an environment variable defined by the user in one of the service.env files representing the name of the initiation queue.

Related reference

ALTER SERVICE

DEFINE SERVICE

DISPLAY SVSTATUS

START SERVICE

STOP SERVICE

Using a command service object

These examples show how to define a command service object to start a program that writes entries to the operating system's system log when a queue manager is started or stopped.

Before you begin

Note: These examples require you to run the **DEFINE SERVICE** MQSC command. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

These examples are written with UNIX style path separator characters.

About this task

In the following examples:

`logger` is a sample program provided with IBM MQ that can write entries to the operating system's system log.

+QMNAME+ is a token representing the name of the queue manager.

Procedure

- Define a command service object to start a program that writes entries to the operating system's system log when a queue manager is started or stopped:

```
DEFINE SERVICE(S2) +
CONTROL(QMGR) +
SERVTYPE(COMMAND) +
STARTCMD('/usr/bin/logger') +
STARTARG('Queue manager +QMNAME+ starting') +
STOPCMD('/usr/bin/logger') +
STOPARG('Queue manager +QMNAME+ stopping')
```

- Define a command service object to start a program that writes entries to the operating system's system log only when a queue manager is stopped:

```
DEFINE SERVICE(S3) +
CONTROL(QMGR) +
SERVTYPE(COMMAND) +
STOPCMD('/usr/bin/logger') +
STOPARG('Queue manager +QMNAME+ stopping')
```

Related reference

DEFINE SERVICE

Managing objects for triggering

These examples show how to start an application automatically when certain conditions on a queue are met. For example, you might want to start an application when the number of messages on a queue

reaches a specified number. This facility is called *triggering*. You have to define the objects that support triggering.

Before you begin

Note: These examples require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

These examples are written with UNIX style path separator characters.

About this task

For a detailed description of triggering, see Starting IBM MQ applications using triggers.

Procedure

- Define an application queue for triggering.

  An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue.

  Triggering itself is enabled by the **Trigger** attribute (TRIGGER in MQSC commands). In this example, a trigger event is to be generated when there are 100 messages of priority 5 or greater on the local queue MOTOR.INSURANCE.QUEUE, as follows:

  ```
  DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
  PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
  MAXMSGL (2000) +
  DEFPSIST (YES) +
  INITQ (MOTOR.INS.INIT.QUEUE) +
  TRIGGER +
  TRIGTYPE (DEPTH) +
  TRIGDPTH (100)+
  TRIGMPRI (5)
  ```

  where:

  QLOCAL (MOTOR.INSURANCE.QUEUE)
  Is the name of the application queue being defined.

  PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
  Is the name of the process definition that defines the application to be started by a trigger monitor program.

  MAXMSGL (2000)
  Is the maximum length of messages on the queue.

  DEFPSIST (YES)
  Specifies that messages on this queue are persistent by default.

  INITQ (MOTOR.INS.INIT.QUEUE)
  Is the name of the initiation queue on which the queue manager is to put the trigger message.

  TRIGGER
  Is the trigger attribute value.

  TRIGTYPE (DEPTH)
  Specifies that a trigger event is generated when the number of messages of the required priority (TRIGMPRI) reaches the number specified in TRIGDPTH.

  TRIGDPTH (100)
  Is the number of messages required to generate a trigger event.

  TRIGMPRI (5)
  Is the priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

- Define an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue MOTOR.INS.INIT.QUEUE for guidance:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
GET (ENABLED) +
NOSHARE +
NOTRIGGER +
MAXMSGL (2000) +
MAXDEPTH (1000)
```

- Define a process

  Use the DEFINE PROCESS command to create a process definition. A process definition defines the application to be used to process messages from the application queue. The application queue definition names the process to be used and thereby associates the application queue with the application to be used to process its messages. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

  ```
  DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
  DESCR ('Insurance request message processing') +
  APPLTYPE (UNIX) +
  APPLICID ('/u/admin/test/IRMP01') +
  USERDATA ('open, close, 235')
  ```

  Where:

  MOTOR.INSURANCE.QUOTE.PROCESS
  Is the name of the process definition.

  DESCR ('Insurance request message processing')
  Describes the application program to which this definition relates. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

  APPLTYPE (UNIX)
  Is the type of application to be started.

  APPLICID ('/u/admin/test/IRMP01')
  Is the name of the application executable file, specified as a fully qualified file name. In Windows systems, a typical APPLICID value would be `c:\appl\test\irmp01.exe`.

  USERDATA ('open, close, 235')
  Is user-defined data, which can be used by the application.

- Display attributes of a process definition

  Use the DISPLAY PROCESS command to examine the results of your definition. For example:

  ```
  DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)


  24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
  AMQ8407: Display Process details.
  DESCR ('Insurance request message processing')
  APPLICID ('/u/admin/test/IRMP01')
  USERDATA (open, close, 235)
  PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
  APPLTYPE (UNIX)
  ```

  You can also use the MQSC command ALTER PROCESS to alter an existing process definition, and the DELETE PROCESS command to delete a process definition.

CAPEXPRY: Limit message expiry time

Administrators can limit the expiry time of any message put to a queue or topic by using the **CAPEXPRY** attribute.

There are two ways to set **CAPEXPRY**:

- By specifying the **CAPEXPRY** attribute.
- By specifying the **CAPEXPRY** attribute in the **CUSTOM** attribute.

From IBM MQ 9.4.0, you should use the **CAPEXPRY** attribute without specifying it inside the **CUSTOM** attribute.

The following information applies regardless of how you have set the **CAPEXPRY** attribute:

- The value of **CAPEXPRY** is expressed in tenths of seconds, so a value of 600 represents one minute.
- If an application sets an expiry time in the MQMD structure that exceeds the **CAPEXPRY** value of the queue or topic, it is reduced to **CAPEXPRY**. If the expiry time is lower than **CAPEXPRY**, it remains unchanged.
- If more than one object is used on the resolution path, for example, when a message is put to an alias or remote queue, then the lowest of all the **CAPEXPRY** values is used as the upper limit for message expiry.
- Changes to the **CAPEXPRY** values take effect immediately. The expiry value is evaluated for each put to a queue or topic and so is sensitive to the object resolution, which might differ between each put operation.

  However, existing messages in the queue, prior to a change in **CAPEXPRY**, are not affected by the change (that is, their expiry time remains the same). Only new messages that are put into the queue or topic after the change in **CAPEXPRY** have the new expiry time.
- In a cluster where a put is performed to a queue opened with MQOO_BIND_NOT_FIXED, messages can be assigned different expiry values on each put, depending on the **CAPEXPRY** value set for the transmission queue, used by the channel, that sends the message to the selected target queue manager.

Attention: CAPEXPRY must not be used on any queues that hold IBM MQ internally-generated messages such as any SYSTEM.CLUSTER.* queue and the SYSTEM.PROTECTION.POLICY.QUEUE.

Interactions between the CAPEXPRY attribute and the CAPEXPRY attribute in the CUSTOM attribute

When migrating an object from an earlier version of the product, the **CAPEXPRY** value is set to the default value of NOLIMIT (for queues) and ASPARENT (for topics). Therefore, if set, the **CAPEXPRY** attribute inside the **CUSTOM** attribute takes precedence.

If a queue or topic already has the **CAPEXPRY** attribute set inside the **CUSTOM** attribute you must remove the **CAPEXPRY** attribute from the **CUSTOM** attribute before the new **CAPEXPRY** attribute is set to a non-default value. You can do this in a single command, for example:

```
ALTER QLOCAL(Q1) CAPEXPRY(1000) CUSTOM('')
```

The **CAPEXPRY** attribute that you set directly on queues or topics (not inside the **CUSTOM** attribute) is a clustered attribute. All instances of a clustered queue, or clustered topic, should use the same **CAPEXPRY** value. It is still possible that a transmission queue reduces the expiry time of a message if **CAPEXPRY** has been set on the transmission queue, and the value is lower than the **CAPEXPRY** attribute of the cluster queue or topic.

Related reference

DEFINE queues

DEFINE topic (define a new administrative topic)

Using the dmpmqmsg utility between two systems

The **dmpmqmsg** utility (formerly *qload*) allows you to copy or move the contents of a queue, or its messages, to a file.

Overview

The file that you create with **dmpmqmsg** can be saved away as required and used at some later point to reload the messages back onto the queue.

Important:

1. The file has a specific format understood by the utility. However, the file is human-readable, so that you can update it in an editor before you reload it. If you do edit the file you must not change its format.
2. The **dmpmqmsg** utility is shipped with the runtime fileset for AIX, Linux, and Windows, so is available in both the IBM MQ server and client.

Possible uses are:

- Saving the messages that are on a queue, to a file. Possibly for archiving purposes, and later reload back to a queue.
- Reloading a queue with messages you previously saved to a file.
- Removing old messages from a queue.
- 'Replaying' test messages from a stored location, even maintaining the correct time between the messages if required.

Attention: SupportPac MO03 used the **-l** parameter for specifying local or client binding. **-l** has been replaced by the **-c** parameter.

-P is now used for code page information instead of **-c**.

See dmpmqmsg for further information on the command and the available parameters.

Example of using the dmpmqmsg utility on Linux, using a Windows machine

You have a queue manager on a Linux machine that has messages on a queue (*Q1*) that you want to move into another queue (*Q2*) in the same queue manager. You want to initiate the **dmpmqmsg** utility from a Windows machine.

Queue (*Q1*) has four messages that have been added by using the sample **amqsput** (local queue manager) or **amqsputc** (remote queue manager) application.

On the Linux machine you see:

```
display ql(Q1) CURDEPTH
        2 : display ql(Q1) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q1)
TYPE(QLOCAL)
     CURDEPTH(4)
```

Set the MQSERVER environment variable to point to the queue manager in Linux. For example:

```
set MQSERVER=SYSTEM.DEF.SVRCONN/TCP/veracruz.x.com(1414)
```

where *veracruz* is the name of the machine.

Run the **dmpmqmsg** utility to read from the queue, *Q1*, and store the output in `c:\temp\mqqload.txt`.

Connect as a remote client to the queue manager, *QM_VER*, running in the Linux host and port established by MQSERVER. You achieve the connection as a remote client by using the attribute: `-c`.

```
dmpmqmsg -m QM_VER -i Q1 -f c:\temp\mqqload.txt -c
Read      - Files:    0   Messages:       4   Bytes:           22
Written - Files:    1   Messages:       4   Bytes:           22
```

The output file `c:\temp\mqqload.txt` contains text, using a format that the **dmpmqmsg** utility understands.

On the Windows machine, issue the **dmpmqmsg** command (using the `-o` option instead of the `-i` option) to load queue (*Q2*) on the Linux machine from a file on the Windows machine:

```
dmpmqmsg -m QM_VER -o Q2 -f c:\temp\mqqload.txt -c
Read      - Files:    1   Messages:       4   Bytes:           22
Written - Files:    0   Messages:       4   Bytes:           22
```

On the Linux machine, note that there are now four messages in the queue that have been restored from the file.

```
display ql(Q2) CURDEPTH
        6 : display ql(Q2) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q2)
TYPE(QLOCAL)
     CURDEPTH(4)
```

On the Linux machine,

Delete the messages from the original queue.

```
clear qlocal(Q1)
        4 : clear qlocal(Q1)
AMQ8022: IBM MQ queue cleared.
```

Confirm that there are no more messages on the original queue:

```
display ql(Q1) CURDEPTH
        5 : display ql(Q1) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q1)
TYPE(QLOCAL)
     CURDEPTH(0)
```

See dmpmqmsg for a description of the command and its parameters.

Related concepts

“Examples of using the dmpmqmsg utility” on page 198

Simple ways in which you can use the **dmpmqmsg** utility (formerly **qload**).

Examples of using the dmpmqmsg utility

Simple ways in which you can use the **dmpmqmsg** utility (formerly **qload**).

Unload a queue to a file

Use the following options on the command line to save the messages that are on a queue, into a file:

```
dmpmqmsg -m QM1 -i Q1 -f c:\myfile
```

This command takes a copy of the messages from the queue and saves them in the file specified.

Unload a queue to a series of files

You can unload a queue to a series of files by using an `insert` character in the file name. In this mode each message is written to a new file:

```
dmpmqmsg -m QM1 -i Q1 -f c:\myfile%n
```

This command unloads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Load a queue from a file

To reload a queue with the messages you saved in "Unload a queue to a file" on page 198, use the following options on the command line:

```
dmpmqmsg -m QM1 -o Q1 -f c:\myfile%n
```

This command unloads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Load a queue from a series of files

You can load a queue from a series of files by using an `insert` character in the file name. In this mode each message is written to a new file:

```
dmpmqmsg -m QM1 -o Q1 -f c:\myfile%n
```

This command loads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Copy the messages from one queue to another queue

Replace the file parameter in "Unload a queue to a file" on page 198, with another queue name and use the following options:

```
dmpmqmsg -m QM1 -i Q1 -o Q2
```

This command allows the messages from one queue to be copied to another queue.

Copy the first 100 messages from one queue to another queue

Use the command in the previous example and add the `-r#100` option:

```
dmpmqmsg -m QM1 -i Q1 -o Q2 -r#100
```

Move the messages from one queue to another queue

A variation on "Load a queue from a file" on page 199. Note the distinction between using **-i** (lowercase) which only browses a queue, and **-I** (uppercase) which destructively gets from a queue:

```
dmpmqmsg -m QM1 -I Q1 -o Q2
```

Move messages older than one day from one queue to another queue

This example shows the use of age selection. Messages can be selected that are older than, younger than, or within a range of ages.

```
dmpmqmsg -m QM1 -I Q1 -o Q2 -T1440
```

Display the ages of messages currently on a queue

Use the following options on the command line:

```
dmpmqmsg -m QM1 -i Q1 -f stdout -dT
```

Work with the message file

Having unloaded the message from your queue, as in "Unload a queue to a file" on page 198, you might want to edit the file.

You might also want to change the format of the file to use one of the display options that you did not specify at the time you unloaded the queue.

You can use the **dmpmqmsg** utility to reprocess the file into the required format even after the unload of the queue has taken place. Use the following options on the command line.

```
dmpmqmsg -f c:\oldfile -f c:\newfile -dA
```

See dmpmqmsg for a description of the command and its parameters.

Working with remote IBM MQ objects

You can administer IBM MQ objects on remote queue managers by using MQSC commands, PCF commands, or the administrative REST API. Before you can use any of these methods, you must define transmission queues and channels between the local queue manager and the remote queue manager so that commands can be sent to the remote queue manager, and responses received by the local queue manager. Alternatively, you can configure a queue manager cluster, and then use the same remote administration methods.

About this task

To prepare queue managers for remote administration, you must configure the following objects on the local queue manager:

- A listener.
- A transmission queue that has the name of the remote queue manager.
- A sender channel that has the connection details for the remote queue manager.
- A receiver channel that has the same name as the sender channel on the remote queue manager.

You must also configure the following objects on the remote queue manager:

- A listener.
- A transmission queue that has the name of the local queue manager.
- A sender channel that has the connection details for the local queue manager.
- A receiver channel that has the same name as the sender channel on the local queue manager.

For more information about configuring these objects, see "Configuring queue managers for remote administration" on page 201.

Alternatively, you can configure a queue manager cluster. A *cluster* is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network without complex transmission queue, channel, and queue definitions. Clusters can be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications. Even the smallest cluster reduces system administration costs.

Establishing a network of queue managers in a cluster involves fewer definitions than establishing a traditional distributed queuing environment. With fewer definitions to make, you can set up or change your network more quickly and easily, and reduce the risk of making an error in your definitions.

To set up a cluster, you need one cluster sender (CLUSSDR) and one cluster receiver (CLUSRCVR) definition for each queue manager. You do not need any transmission queue definitions or remote queue definitions. The principles of remote administration are the same when used within a cluster, but the definitions themselves are greatly simplified.

For more information about configuring a cluster, see Configuring a queue manager cluster.

Procedure

- For information on how to administer remote IBM MQ objects, see the following subtopics:
  - “Configuring queue managers for remote administration” on page 201
  - “Managing the command server for remote administration” on page 205
  - “Issuing MQSC commands on a remote queue manager” on page 206
  - “Data conversion between coded character sets ” on page 207

Configuring queue managers for remote administration

You can administer a remote queue manager from a local queue manager by using the administrative REST API, MQSC, or PCF commands. The remote queue manager might be on the same system, in a different installation, or on a different system with the same environment, or a different IBM MQ environment. Before you can remotely administer a queue manager from a local queue manager, you must create a sender and receiver channel, a listener, and a transmission queue on each queue manager. These channels and queues enable the commands to be sent to the remote queue manager and the responses to be received on the local queue manager. The procedure to create these queues and channels is the same whether you want to use the administrative REST API, MQSC, or PCF commands.

Before you begin

- The following procedure uses example queue managers `source.queue.manager`, and `target.queue.manager`. You must create and start these queue managers on your system to follow these steps, or substitute your own queue manager names in the relevant steps.
- The following procedure uses TCP/IP as the transport type. You must know the IP address of both systems to complete this task.
- The following procedure creates listeners that use the network ports 1818 on the local system and 1819 on the remote system. You can use other ports, but must substitute your port values in the appropriate steps.
- You must run the commands in the procedure locally or over a network facility such as Telnet.

About this task

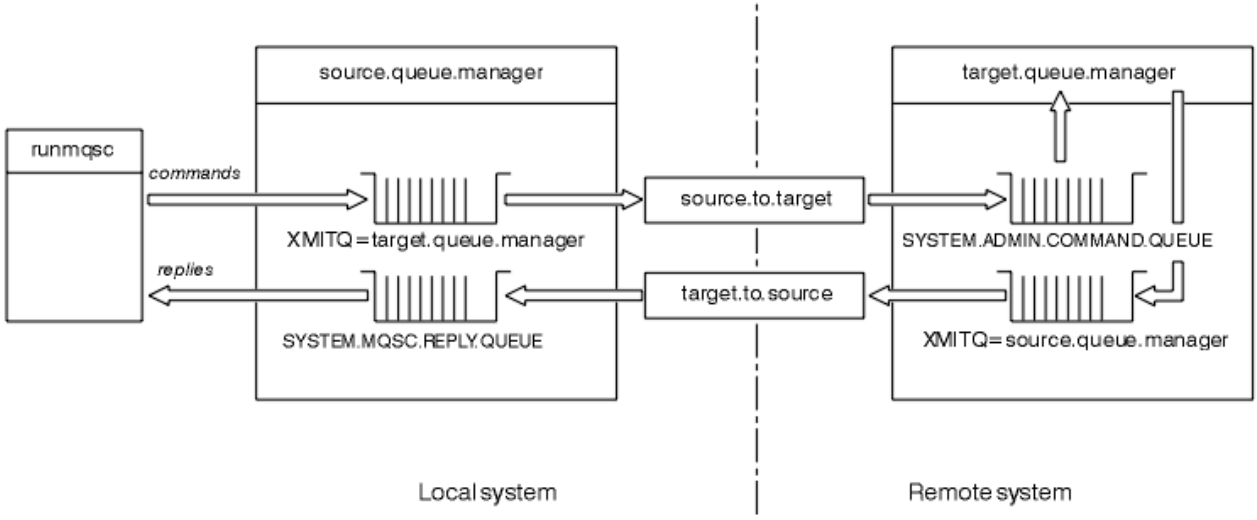


Figure 15. Setting up channels and queues for remote administration

Figure 15 on page 201 shows the configuration of queue managers, queues, and channels that you need for remote administration:

- The object `source.queue.manager` is the source queue manager from which you can issue administrative REST API, MQSC, or PCF commands, and to which the results of these commands are returned.
- The object `target.queue.manager` is the name of the target queue manager, which processes the commands and generates any operator messages.
- Commands get put onto the transmission queue that has the same name as the remote queue manager. In this case, `target.queue.manager`. A transmission queue is a specialized local queue that temporarily holds messages before the MCA picks them up and sends them to the remote queue manager.
- The commands get sent by the `source.to.target` channel to the `SYSTEM.ADMIN.COMMAND.QUEUE` on the remote queue manager. Each end of the channel has a separate definition. One end is a sender and the other end is a receiver. The two definitions must have the same name and together constitute a single message channel.
- The command output is put on the remote transmission queue that has the same name as the local queue manager from which the command was sent. In this case, `source.queue.manager`.
- The output is sent by the `target.to.source` channel to an appropriate reply queue, where it is taken and output by the original command.

Procedure

1. On the remote system queue manager, ensure that the command queue `SYSTEM.ADMIN.COMMAND.QUEUE` is present. This queue is created by default when a queue manager is created.
2. On the remote system, check that the command server is running on the queue manager. If the command server is not running, remote administration is not possible:
   a) Start **runmqsc** for the queue manager. For example, for queue manager `target.queue.manager`, enter the following command:

   ```
   runmqsc target.queue.manager
   ```

   b) Display the status of the command server by entering the following command:

   ```
   DISPLAY QMSTATUS CMDSERV
   ```

   c) Exit the **runmqsc** command prompt by entering the following command:

   ```
   end
   ```

   d) If the command server is not started, start it. For example, for queue manager `target.queue.manager`, enter the following command:

   ```
   strmqcsv target.queue.manager
   ```

3. Define the channels, listener, and transmission queue on the local queue manager:
   a) Start **runmqsc** for the queue manager. For example, for queue manager `source.queue.manager`, enter the following command:

   ```
   runmqsc source.queue.manager
   ```

   b) Define the sender channel. This sender channel must have the same name as the receiver channel on the remote queue manager. For example, enter the following MQSC command, replacing the value for **CONNAME** with the IP address for the remote queue manager and the port number of the listener:

   ```
   DEFINE CHANNEL ('source.to.target') +
   CHLTYPE(SDR) +
   ```

```
CONNAME (localhost:1819) +
XMITQ ('target.queue.manager') +
TRPTYPE(TCP)
```

c) Define the receiver channel. This receiver channel must have the same name as the sender channel on the remote queue manager. For example, enter the following command:

```
DEFINE CHANNEL ('target.to.source') +
CHLTYPE(RCVR) +
TRPTYPE(TCP)
```

d) Define the listener on the local queue manager. For example, enter the following command:

```
DEFINE LISTENER ('source.queue.manager') +
TRPTYPE (TCP) +
PORT (1818)
```

e) Define the transmission queue on the local queue manager. This transmission queue must have the same name as the remote queue manager. For example, enter the following command:

```
DEFINE QLOCAL ('target.queue.manager') +
USAGE (XMITQ)
```

f) Start the listener. For example, enter the following command:

```
START LISTENER ('source.queue.manager')
```

g) Exit the **runmqsc** command prompt by entering the following command:

```
end
```

4. Define the channels, listener, and transmission queue on the remote queue manager:

a) Start **runmqsc** for the queue manager. For example, for queue manager `target.queue.manager`, enter the following command:

```
runmqsc target.queue.manager
```

b) Define the sender channel. This sender channel must have the same name as the receiver channel on the local queue manager. For example, enter the following MQSC command, replacing the value for **CONNAME** with the IP address for the local queue manager and port number for the listener:

```
DEFINE CHANNEL ('target.to.source') +
CHLTYPE(SDR) +
CONNAME (localhost:1818) +
XMITQ ('source.queue.manager') +
TRPTYPE(TCP)
```

c) Define the receiver channel. This receiver channel must have the same name as the sender channel on the local queue manager For example, enter the following command:

```
DEFINE CHANNEL ('source.to.target') +
CHLTYPE(RCVR) +
TRPTYPE(TCP)
```

d) Define the listener. For example, enter the following command:

```
DEFINE LISTENER ('target.queue.manager') +
TRPTYPE (TCP) +
PORT (1819)
```

e) Define the transmission queue. This transmission queue must have the same name as the local queue manager. For example, enter the following command:

```
DEFINE QLOCAL ('source.queue.manager') +
USAGE (XMITQ)
```

f) Start the listener. For example, enter the following command:

```
START LISTENER ('target.queue.manager')
```

g) Exit **runmqsc** by entering the following command:

```
end
```

5. Start the sender channel on the local system:
   a) Start **runmqsc** for the queue manager. For example, for queue manager `source.queue.manager`, enter the following command:

```
runmqsc source.queue.manager
```

   b) Start the sender channel. For example, enter the following command:

```
START CHANNEL ('source.to.target')
```

   c) Exit **runmqsc** by entering the following command:

```
end
```

6. Start the sender channel on the remote system:
   a) Start runmqsc for the queue manager. For example, for queue manager `target.queue.manager`, enter the following command:

```
runmqsc target.queue.manager
```

   b) Start the sender channel. For example, enter the following command:

```
START CHANNEL ('target.to.source')
```

   c) Exit **runmqsc** by entering the following command:

```
end
```

7. Test that the configuration has completed successfully by sending an MQSC command from the local system to the remote queue manager:
   a) Start the **runmqsc** command prompt for the remote queue manager from the local system. For example, enter the following command:

```
runmqsc -w 30 -m source.queue.manager target.queue.manager
```

   b) Display the queues on the remote queue manager by entering the following command:

```
DISPLAY QUEUE (*)
```

   On success, a list of queues from the remote queue manager is displayed.
   c) If these steps do not work, check that the channels on both systems are in a running state. If the channels are not running, and do not start, check the channels and transmission queues are configured correctly, and that the command server is running. For example, check that the correct CONNAME is specified for the sender channels, and that the transmission queues have the correct names. Also, check the queue manager logs for security exceptions that might help resolve the issue.

Results

Your queue managers are configured to remotely administer the remote queue manager from the local system.

What to do next

- Learn more about remote administration using MQSC commands: “Issuing MQSC commands on a remote queue manager” on page 206

- Learn more about writing administration programs using PCF commands: "Using IBM MQ Programmable Command Formats" on page 26.
- Learn more about using the administrative REST API for remote administration: "Remote administration using the REST API" on page 78.

Managing the command server for remote administration

Each queue manager has a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message. You can start, stop, and display the status of the command server. A command server is mandatory for all administration involving PCF commands, the MQAI, and also for remote administration.

Before you begin

Depending on the value of the queue manager attribute, **SCMDSERV**, the command server is either started automatically when the queue manager starts, or must be started manually. If the command server is started automatically, you cannot use the strmqcsv or endmqcsv commands to start and stop the command server. You can change the value of the **SCMDSERV** attribute by using the MQSC command **ALTER QMGR**. By default, the command server is started automatically.

Stopping a queue manager also ends the command server associated with it.

Procedure

- Display the status of the command server:
  a) Start the **runmqsc** command prompt for the appropriate queue manager by entering the following command:

     ```
     runmqsc target.queue.manager
     ```

     where `target.queue.manager` is the queue manager for which the command server is being displayed.
  b) Display the command server status by entering the following MQSC command:

     ```
     DISPLAY QMSTATUS CMDSERV
     ```

  c) Exit the **runmqsc** command prompt by entering the following command:

     ```
     end
     ```

- If the command server is not set to start automatically, start the command server by entering the following command:

  ```
  strmqcsv target.queue.manager
  ```

  where `target.queue.manager` is the queue manager for which the command server is being started.
- If the command server is not set to start automatically, stop the command server by entering the following command:

  ```
  endmqcsv target.queue.manager
  ```

  where `target.queue.manager` is the queue manager for which the command server is being stopped.

  By default, the command server stops in a controlled way. You can stop the command server immediately by adding the `-i` flag to the command.

Issuing MQSC commands on a remote queue manager

After you configure queue managers for remote administration, you can use a particular form of the **runmqsc** command on a local system to run MQSC commands on a remote queue manager. Each command is sent as an Escape PCF to the command queue, SYSTEM.ADMIN.COMMAND.QUEUE, of the remote queue manager. Replies are received on the SYSTEM.MQSC.REPLY.QUEUE queue.

Before you begin

You must complete the steps in "Configuring queue managers for remote administration" on page 201 to configure channels, transmission queues, listeners, and the command server before you can remotely administer a queue manager by using MQSC commands.

Procedure

1. Ensure that the command server is running on the remote queue manager.

   For information on how to start the command server on a queue manager, see "Managing the command server for remote administration" on page 205.
2. On the source queue manager, you can then run MQSC commands in one of two ways:
   - Interactively, by starting **runmqsc** with the following commands:
     - If the remote queue manager is on z/OS, enter the following command:

       ```
       runmqsc -w 30 -x -m source.queue.manager target.queue.manager
       ```
     - If the remote queue manager is on Multiplatforms, enter the following command:

       ```
       runmqsc -w 30 -m source.queue.manager target.queue.manager
       ```
   - From a command file:
     a. Put the MQSC commands to be run on the remote system in a text file, one command per line.
     b. Verify your MQSC commands on the local queue manager by using the -v flag on the **runmqsc** command. The -v flag checks that the commands are valid, but does not run them. Be aware that some commands might fail if they are applicable to the remote queue manager but are not applicable to the local queue manager:

        ```
        runmqsc -v source.queue.manager < myCmdFile.in > results.out
        ```

        The `myCmdFile.in` contains the MQSC commands to check, and the `results.out` file contains the verification results for the commands.
     c. Run the command file on the remote queue manager by entering one of the following commands:
        - If the remote queue manager is on z/OS, enter the following command:

          ```
          runmqsc -w 30 -x -m source.queue.manager target.queue.manager < myCmdFile.in >
          results.out
          ```
        - If the remote queue manager is on Multiplatforms, enter the following command:

          ```
          runmqsc -w 30 -m source.queue.manager target.queue.manager < myCmdFile.in >
          results.out
          ```

   The parameters used are the following parameters:

   -w *seconds*
   Specifies that the MQSC commands are run in indirect mode, where commands are put on the command server input queue and executed in order.

   The variable *seconds* specifies the length of time to wait, in seconds, for a response from the remote queue manager. Any replies received after this time are discarded, but the MQSC

commands still run on the remote queue manager. The following message is generated on the local queue manager when the command times out:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

When you stop issuing MQSC commands, the local queue manager displays any timed-out responses that have arrived and discards any further responses.

-x
Specifies that the remote queue manager is a z/OS queue manager.

-m ***localQMgrName***
Specifies the name of the local queue manager that you want to use to submit commands to the remote queue manager

What to do next

If you have difficulty in running MQSC commands remotely:

- Check that the remote queue manager is running.
- Check that the command server is running on the remote system.
- Check that the channel disconnect interval has not expired. For example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.
- Ensure that the requests that are sent from the local queue manager make sense to the target queue manager. For example, requests that include parameters that are not supported on the remote queue manager.
- Also see Resolving problems with MQSC commands.

Data conversion between coded character sets

Message data in IBM MQ defined formats (also known as built-in formats) can be converted by the queue manager from one coded character set to another, provided that both character sets relate to a single language or a group of similar languages.

For example, conversion between coded character sets with identifiers (CCSIDs) 850 and 500 is supported, because both apply to Western European languages.

For EBCDIC newline (NL) character conversions to ASCII, see All queue managers stanza of the mqs.ini file and the **AMQ_CONVEBCDICNEWLINE** environment variable.

Supported conversions are defined in Data conversion processing.

Converting between CCSIDs 37 and 500 is supported on the IBM MQ Appliance, Windows, Linux, and macOS.

When a queue manager cannot convert messages in built-in formats

The queue manager cannot automatically convert messages in built-in formats if their CCSIDs represent different national-language groups. For example, conversion between CCSID 850 and CCSID 1025 (which is an EBCDIC coded character set for languages using Cyrillic script) is not supported because many of the characters in one coded character set cannot be represented in the other. If you have a network of queue managers working in different national languages, and data conversion among some of the coded character sets is not supported, you can enable a default conversion.

For platforms to which the `ccsid_part2.tbl` applies, see “Specifying default data conversion” on page 210 using `ccsid_part2.tbl` for further information. Default data conversion on platforms other than those to which the `ccsid_part2.tbl` file applies is described in “Default data conversion” on page 208.

ccsid_part2.tbl file

An additional file, `ccsid_part2.tbl`, is provided.

The `ccsid_part2.tbl` file takes precedence over the `ccsid.tbl` file and:

- Allows you to add or modify CCSID entries
- Specify default data conversion
- Specify data for different command levels

The `ccsid_part2.tbl` is applicable to the following platforms only:

- Linux Linux - all versions
- Windows Windows

Windows On IBM MQ for Windows, `ccsid_part2.tbl` is located in directory *MQDataRoot*`\conv\table` by default. Furthermore, on IBM MQ for Windows it records all the supported code sets.

Linux On IBM MQ for Linux, `ccsid_part2.tbl` is located in directory *MQDataRoot*`/conv/table`, and the supported code sets are held in conversion tables provided by IBM MQ.

Although the `ccsid_part2.tbl` file replaces the existing `ccsid.tbl` file used in previous versions of IBM MQ to supply additional CCSID information, the `ccsid.tbl` file continues to be parsed by IBM MQ and must therefore not be deleted.

For more information, see "The ccsid_part2.tbl file" on page 209.

`ccsid.tbl` file

On platforms other than those to which `ccsid_part2.tbl` applies, the file `ccsid.tbl` is used for the following purposes:

- AIX On AIX, the supported code sets are held internally by the operating system.
- It specifies any additional code sets. To specify additional code sets, you need to edit `ccsid.tbl` (guidance on how to do this is provided in the file).
- It specifies any default data conversion.

You can update the information recorded in `ccsid.tbl`; you might want to do this if, for example, a future release of your operating system supports additional coded character sets.

Default data conversion

If you set up channels between two machines on which data conversion is not normally supported, you must enable default data conversion for the channels to work.

On platforms other than those to which `ccsid_part2.tbl` applies, to enable default data conversion, edit the `ccsid.tbl` file to specify a default EBCDIC CCSID and a default ASCII CCSID. Instructions on how to do this are included in the file. You must do this on all machines that will be connected using the channels. Restart the queue manager for the change to take effect.

The default data-conversion process is as follows:

- If conversion between the source and target CCSIDs is not supported, but the CCSIDs of the source and target environments are either both EBCDIC or both ASCII, the character data is passed to the target application without conversion.
- If one CCSID represents an ASCII coded character set, and the other represents an EBCDIC coded character set, IBM MQ converts the data using the default data-conversion CCSIDs defined in ccsid.tbl.

Note: Try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. If you use only the set of characters that is valid for IBM MQ object names (as defined in Naming IBM MQ objects) you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

Converting messages in user-defined formats

The queue manager cannot convert messages in user-defined formats from one coded character set to another. If you need to convert data in a user-defined format, you must supply a data-conversion exit for each such format. Do not use default CCSIDs to convert character data in user-defined formats. For more information about converting data in user-defined formats and about writing data conversion exits, see the Writing data-conversion exits.

Changing the queue manager CCSID

When you have used the **CCSID** attribute of the **ALTER QMGR** command to change the CCSID of the queue manager, stop and restart the queue manager to ensure that all running applications, including the command server and channel programs, are stopped and restarted.

This is necessary because any applications that are running when the queue manager CCSID is changed continue to use the existing CCSID.

The `ccsid_part2.tbl` file

The `ccsid_part2.tbl` file is used to supply additional CCSID information. The `ccsid_part2.tbl` file replaces the `ccsid.tbl` file that was used before IBM MQ 9.0.

Note: The `ccsid.tbl` file, which was used before IBM MQ 9.0 to supply additional CCSID information, continues to be parsed by IBM MQ and should not be deleted. However, entries in `ccsid_part2.tbl` take precedence over other entries in `ccsid.tbl`.

You should use `ccsid_part2.tbl` rather than `ccsid.tbl` because `ccsid_part2.tbl`:

- Contains support for the Unicode encoding values. From IBM MQ 9.0, the product supports all Unicode characters defined in the Unicode 8.0 standard in data conversion, including full support for UTF-16. For more information, see “Data conversion between coded character sets ” on page 207.
- Allows you to specify the version of CCSID entries, so that the entries are only applicable to selected command levels.

You can use the `ccsid_part2.tbl` file to:

- Add or modify CCSID entries
- Specify default data conversion
- Specify data for different command levels

The `ccsid_part2.tbl` file is applicable to the following platforms only:

- Linux Linux - all versions
- Windows Windows

The location of the `ccsid_part2.tbl` file depends on your platform:

- Linux The *MQDataRoot*`/conv/table` directory on all versions of Linux.
- Windows The *MQDataRoot*`\conv\table` directory on Windows.

Adding or modifying CCSID entries

An entry in the `ccsid_part2.tbl` file has the following format:

```
<CCSID number> <Base CCSID> <DBCS CodePage> <SBCS CodePage>
<Type> <Encoding> <ACRI> <Name>
```

An example entry for CCSID 1200 (UTF-16) is:

```
1200 1200 1200 1200 3 8 0 UTF-16
```

Note: For more details on the value for ACRI see the comment in the `ccsid_part2.tbl` file.

In the `ccsid_part2.tbl` format:

Type can equal:

1=SBCS

2=DBCS

3=MBCS

Encoding can equal:

1=EBCDIC

2 = ASCII

3 = ISO

4 = UCS-2

5 = UTF-8

6 = Euc

7 = GB18030

8 = UTF-16

9 = UTF-32

When editing the file you:

- Can specify a comment by using the # symbol at the start of a line. This prevents IBM MQ from attempting to parse the line.
- Cannot supply in-line comments.
- Must ensure that you do not create blank lines.
- Must not add new entries at the end of the file.

  New CCSID entries should be added before the ACRI table information.

Specifying default data conversion

You can define default conversion CCSIDs, which are used to convert between ASCII or similar and EBCDIC CCSIDs, if no conversion is supported between two CCSIDs.

If you enable this function, the default conversion is used for transmission and message headers, and can also be used in user data conversion.

Default conversions are enabled by creating two lines similar to the following:

```
default       0      500     1     1    0
default       0      850     1     2    0
```

The first line sets the default for EBCDIC CCSIDs to 500 and the second line sets the default for ASCII and similar CCSIDs to 850.

Specifying data for different command levels

To specify CCSID entries for different command levels of IBM MQ you use a colon symbol followed by the command level (or command levels) of IBM MQ that you want the next section to be applicable to.

The number represents the minimum command level that the queue manager or client must be running in. For example, if the current queue manager is command level 900, and it encounters an 800 or 900 command level flag, the CCSIDs are read.

However, a queue manager at level 800 ignores any CCSIDs in the 900 section.

The command level specified is applicable to all CCSID entries encountered after a command level flag, until a new command level flag is found.

If you require to set the command level to all command levels, specify the number zero.

When first parsing `ccsid_part2.tbl`, IBM MQ treats all CCSIDs encountered as valid for all command levels of IBM MQ.

Versioning starts to be used only when IBM MQ encounters the first command level flag.

The following code snippet shows an example of using Versioning:

```
# Comment Block
# End of Comment Block
# Because no command level flag is specified and we're at the start of the file
# the following CCSIDs will be read on all versions
    819   819       0      819     1    3     0    ISO8859-1
    923   923       0      923     1    3     0    ISO8859-15
   1051  1051       0     1051     1    3     0    IBM-1051
# The colon :900 below shows that the CCSIDs after will only be for MQ cmd level 900 and above
:900
   8629   437       0      437     1    2     0    IBM-437
  12725   437       0      437     1    2     0    IBM-437
  16821   437       0      437     1    2     0    IBM-437
  20917   437       0      437     1    2     0    IBM-437
# The colon :0 below shows that the CCSIDs after will be for all version of MQ
:0
   4946   850       0      850     1    2     0    IBM-850
  33618   850       0      850     1    2     0    IBM-850
  61697   850       0      850     1    2     0    IBM-850
  61698   850       0      850     1    2     0    IBM-850
```

Administering Managed File Transfer

Use Managed File Transfer commands to administer Managed File Transfer. You can also use the IBM MQ Explorer for some of the administrative tasks.

Start transfer by placing a message in an agent command queue

You can also start a file transfer by putting a file transfer message on the command queue of the source agent. An example command queue name is `SYSTEM.FTE.COMMAND.AGENT01`. You must ensure that the message reaches the command queue of the correct source agent; if the message is received by an agent that does not match the source information in the XML, the message is rejected.

The transfer request XML must conform to the `FileTransfer.xsd` schema and use the <request> element as the root element. See File transfer request message format for information about the structure and content of a transfer request message. How you put the transfer request message on an agent command queue is task-specific. For example, you can use the IBM MQ Java API to put a message on the queue programmatically.

Related concepts

“Transferring data from files to messages” on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

“Transferring data from messages to files” on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

“The protocol bridge” on page 288
The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The

dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

"Working with MFT from IBM Integration Bus" on page 326
You can work with Managed File Transfer from IBM Integration Bus using the FTEOutput and FTEInput nodes.

"MFT recovery and restart" on page 327
If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

Related tasks

"Starting an MFT agent" on page 213
You can start a Managed File Transfer Agent from the command line. In this case, the agent process stops when you log off the system.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

"Monitoring file transfers that are in progress" on page 226
You can monitor a file transfer that is in progress using the **Managed File Transfer - Current Transfer Progress** tab in IBM MQ Explorer. This file transfer can be one started from either IBM MQ Explorer or the command line. The tab also displays the progress of scheduled transfers at the point the scheduled transfers start.

"Viewing the status of file transfers in the Transfer Log" on page 228
You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

"Monitoring MFT resources" on page 230
You can monitor Managed File Transfer resources; for example, a queue or a directory. When a condition on this resource is satisfied, the resource monitor starts a task, such as a file transfer. You can create a resource monitor by using the **fteCreateMonitor** command or the **Monitors** view in the Managed File Transfer plug-in for IBM MQ Explorer.

"Working with file transfer templates" on page 260
You can use file transfer templates to store common file transfer settings for repeated or complex transfers. Either create a transfer template from the command line by using the **fteCreateTemplate** command or use the IBM MQ Explorer to create a transfer template by using the **Create New Template for Managed File Transfer** wizard, or save a template while you are creating a file transfer by selecting the **Save transfer settings as a template** check box. The **Transfer Templates** window displays all of the transfer templates that you have created in your Managed File Transfer network.

"Listing MFT agents" on page 218
You can list the Managed File Transfer agents registered with a particular queue manager using the command line or the IBM MQ Explorer.

"Stopping an MFT agent" on page 218
You can stop a Managed File Transfer agent from the command line. When you stop an agent, you are quiescing the agent and allowing the agent to complete its current file transfer before stopping. You can also specify the **-i** parameter at the command line to stop an agent immediately. When the agent has stopped, you cannot use that agent to transfer files until you restart it.

Configuring an MFT logger

Related reference

Guidelines for transferring files

Starting an MFT agent

You can start a Managed File Transfer Agent from the command line. In this case, the agent process stops when you log off the system.

About this task

ALW On AIX, Linux, and Windows, you can configure an agent so that it continues running when you log off from the system and can continue to receive file transfers. The MFT process controller is responsible for starting an MFT agent, and restarting that process if it ends for any reason.

z/OS On z/OS, you can configure the agent to start as a started task from JCL without the need for an interactive session.

Note that, if an agent encounters an unrecoverable error when it is running, a first failure data capture (FDC) is generated and the agent is stopped.

Procedure

- To start an agent from the command line, use the **fteStartAgent** command.
  For more information, see **fteStartAgent**.
- ALW
  To configure an agent so that it continues running when you log off from the system:
  - Windows On Windows, configure the agent to run as a Windows service. For more information, see “Starting an MFT agent as a Windows service” on page 214.
  - Linux AIX On AIX and Linux, configure the agent to start automatically during a reboot by using a script file. For more information, see “Starting an MFT agent at AIX and Linux system startup” on page 215.
- z/OS
  On z/OS, configure the agent to start as a started task from JCL without the need for an interactive session.
  For more information, see “Starting an MFT agent on z/OS” on page 217.

Related tasks

“Listing MFT agents” on page 218
You can list the Managed File Transfer agents registered with a particular queue manager using the command line or the IBM MQ Explorer.

“Stopping an MFT agent” on page 218
You can stop a Managed File Transfer agent from the command line. When you stop an agent, you are quiescing the agent and allowing the agent to complete its current file transfer before stopping. You can also specify the **-i** parameter at the command line to stop an agent immediately. When the agent has stopped, you cannot use that agent to transfer files until you restart it.

Related reference

MFT agent status values
fteStartAgent

Windows Starting an MFT agent as a Windows service

You can start an agent as a Windows service so that when you log off Windows, your agent continues running and can receive file transfers.

About this task

On Windows, when you start an agent from the command line, the agent process runs using the user name you used to log on to Windows. When you log off the system, the agent process stops. To prevent the agent stopping, you can configure an agent to run as a Windows service. Running as a Windows service also allows you to configure agents to be started automatically when the Windows environment starts or is restarted.

Complete the following steps to start an agent that runs as a Windows service. You must be running Managed File Transfer on one of the supported Windows versions to run the agent as a Windows service. For the list of supported environments, refer to the System Requirements for IBM MQ.

The exact steps depend on whether you have already created an agent or whether you are creating an agent. Both options are described in the following steps.

Procedure

1. If you are creating a Managed File Transfer agent, use the **fteCreateAgent**, **fteCreateCDAgent**, or **fteCreateBridgeAgent** command. Specify the **-s** parameter to run the agent as a Windows service. In the following example, the agent AGENT1 is created, which has an agent queue manager QMGR1. The Windows service runs using a user name of `fteuser`, which has an associated password `ftepassword`.

   ```
   fteCreateAgent -agentName AGENT1 -agentQMgr QMGR1 -s -su fteuser -sp ftepassword
   ```

   You can optionally specify a name for the service after the **-s** parameter. If you do not specify a name, the service is named `mqmftAgent`*AGENTQMGR*, where *AGENT* is the agent name you specified and *QMGR* is your agent queue manager name. In this example, the default name for the service is `mqmftAgentAGENT1QMGR1`.

   Note: The Windows user account that you specify using the **-su** parameter must have the **Log on as a service** rights. For information about how to configure this, see Troubleshooting an MFT agent or logger running as a Windows service.

   For more information, see fteCreateAgent, fteCreateCDAgent: create a Connect:Direct® bridge agent, or fteCreateBridgeAgent (create and configure an MFT protocol bridge agent).
2. If you followed the previous step to create an agent, run the MQSC commands that are generated by the **fteCreateAgent**, **fteCreateCDAgent**, or **fteCreateBridgeAgent** command. These commands create the IBM MQ queues that are needed by the agent.

   For example, for an agent named *AGENT1*, an agent queue manager named *QMGR1* and a coordination queue manager named *COORDQMGR1*, run the following command:

   ```
   runmqsc QMGR1 MQ_DATA_PATH\mqft\config\COORDQMGR1\agents\AGENT1\AGENT1_create.mqsc
   ```
3. If you did not follow the previous steps to create an agent and instead want to configure an existing agent to run as a Windows service, first stop your agent if it is running, and then modify its configuration.
   a) The following example uses an agent named AGENT1. Run the following command:

      ```
      fteStopAgent AGENT1
      ```

   b) Use the **fteModifyAgent** command to configure the agent to run as a Windows service:

      ```
      fteModifyAgent -agentName AGENT1 -s Service1 -su fteuser -sp ftepassword
      ```

      For more information, see fteModifyAgent: run an MFT agent as a Windows service.

4. Start your agent using the **fteStartAgent** command. Alternatively, you can use the Windows Services tool, which is available from Administrative Tools in the Control Panel, selected from the Windows desktop start menu, to start the service.

   ```
   fteStartAgent AGENT1
   ```

   The service continues to run even if you log off Windows. To ensure that the service also restarts when Windows restarts after a shutdown, the **Startup Type** field in the Windows Services tool is set to **Automatic** by default. Change this to **Manual** if you do not want the service to restart when Windows restarts.
5. Optional: To stop the agent, either use the fteStopAgent command or use the Windows Services tool. For example, from the command line, run the following command:

   ```
   fteStopAgent AGENT1
   ```

   - When you run the **fteStopAgent** command as a service, the command always runs using the **-i** parameter regardless of whether you specified this parameter. The **-i** parameter stops the agent immediately without completing any transfers that are in progress. This is caused by a limitation of the Windows service.

What to do next

If you have problems starting your Windows service, see Troubleshooting an MFT agent or logger running as a Windows service. This topic also describes the location of the Windows service log files.

Related tasks

"Listing MFT agents" on page 218
You can list the Managed File Transfer agents registered with a particular queue manager using the command line or the IBM MQ Explorer.

"Stopping an MFT agent" on page 218
You can stop a Managed File Transfer agent from the command line. When you stop an agent, you are quiescing the agent and allowing the agent to complete its current file transfer before stopping. You can also specify the **-i** parameter at the command line to stop an agent immediately. When the agent has stopped, you cannot use that agent to transfer files until you restart it.

Related reference

fteCreateAgent (create an MFT agent)

fteCreateCDAgent (create a Connect:Direct bridge agent)

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

fteModifyAgent (run an MFT agent as a Windows service)

Related information

The MFT `agent.properties` file

Linux AIX Starting an MFT agent at AIX and Linux system startup

A Managed File Transfer Agent can be configured to start at system startup on AIX and Linux. When you log off, your agent continues running and can receive file transfers.

When you have created and configured an agent using one of these Managed File Transfer commands; **fteCreateAgent**, **fteCreateCDAgent**, or **fteCreateBridgeAgent**, you can configure it to start automatically during a reboot on AIX and Linux machines by using a script file that simply executes the following command:

```
su -l mqmft_user -c mq_install_root/bin/fteStartAgent agent_name
```

Where *mq_install_root* is the root directory of the required Managed File Transfer installation, the default is: `/opt/mqm` and *agent_name* is the name of the Managed File Transfer Agent to be started. The

use of this script file varies depending on the specific operating system. For example, there are additional options available under Linux.

Linux

Linux

For Linux systems there are multiple ways that you can start applications during the system boot process. In general, consider following these steps:

1. Create a file called `/etc/rc.mqmft` with contents:

   ```
   #!/bin/sh
   su -l mqmft_user"-c mq_install_root/bin/fteStartAgent agent_name"
   ```

   Where *mqmft_user* is the user ID under which the agent process is to run. This user ID must be a member of the mqm group.
2. Make the file executable, for example:

   ```
   chmod 755 /etc/rc.mqmft
   ```
3. Next add the following line to `/etc/inittab`:

   ```
   mqmft:5:boot:/etc/rc.mqmft
   ```

Other ways to start an agent during boot on Linux include adding the script lines to the `/etc/rc.d/rc.local` file, or on Linux SuSe, adding the script lines to the `/etc/init.d/boot.local` file. You should select the method that works best for your environment. Here is some more information on other ways to start an agent during startup on specific Linux distributions that are supported:

SLES 10 and 11

For SUSE Linux Enterprise Server (SLES) 10 and 11 systems, follow these steps:

1. As the system root user ID, create your own `/etc/init.d/rc.rclocal` file.
2. Add the following lines to the `rc.rclocal` file:

   ```
   #!/bin/sh
   ### BEGIN INIT INFO
   # Provides: rc.rclocal
   # Required-Start: $network $syslog
   # Required-Stop: $network $syslog
   # Default-Stop: 0 1 2 6
   # Description: MQMFT agent startup
   ### END INIT INFO
   su -l mqmft_user"-c mq_install_root/bin/fteStartAgent agent_name"
   ```
3. Run the following commands:

   ```
   chmod 755 rc.rclocal

   chkconfig --add rc.rclocal
   ```

Starting Managed File Transfer agents on Linux with `systemd`

Linux

Carry out the following procedure:

1. Create a file in the `/etc/systemd/` system folder and name it, for example, `<agentname>.service`. Add the following content, where `<agentname>` is `MFT_AGT_LNX_0`.

   ```
   # vi /etc/systemd/system/MFT_AGT_LNX_0.service
   [Unit]
   Description=IBM MQ MFT MFT_AGT_LNX_0
   [Service]
   ExecStart=/opt/mqm/bin/fteStartAgent MFT_AGT_LNX_0
   ExecStop=/opt/mqm/bin/fteStopAgent MFT_AGT_LNX_0
   ```

```
Type=forking
User=mqm
Group=mqm
KillMode=none
```

2. To enable the service run the following commands:

```
# systemctl enable MFT_AGT_LNX_0
# systemctl daemon-reload
```

3. To start the agent and check its status, run the following commands:

```
# systemctl start MFT_AGT_LNX_0
# systemctl status MFT_AGT_LNX_0
```

Related tasks

"Stopping an MFT agent" on page 218

You can stop a Managed File Transfer agent from the command line. When you stop an agent, you are quiescing the agent and allowing the agent to complete its current file transfer before stopping. You can also specify the **-i** parameter at the command line to stop an agent immediately. When the agent has stopped, you cannot use that agent to transfer files until you restart it.

Related reference

fteCreateAgent

fteCreateCDAgent: create a Connect:Direct bridge agent

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

z/OS Starting an MFT agent on z/OS

On z/OS, in addition to running the **fteStartAgent** command from a z/OS UNIX System Services session, you can start an agent as a started task from JCL without the need for an interactive session.

A started task is used because it runs under a specific user ID and is not affected by users logging off.

Note: Started tasks are typically run under an administrative user that might not have log-on privileges and so it is not possible to log on to the z/OS system as the user that the agent is running under. The **fteStartAgent**, **fteStopAgent**, **fteSetAgentTraceLevel** commands, and the **fteShowAgentDetails** command with the **-d** parameter specified, cannot be issued for that agent.

You can use the agent property **adminGroup** with Managed File Transfer agents on z/OS. You can define a security manager group, for example MFTADMIN and then add the started task userid and administrator TSO ids to this group. Edit the agent properties file and set the **adminGroup** property to be the name of this security manager group.

```
adminGroup=MFTADMIN
```

Members of this group can then issue the **fteStartAgent**, **fteStopAgent**, and **fteSetAgentTraceLevel** commands, and the **fteShowAgentDetails** command with the **-d** parameter specified, for the agent that is running as a started task.

For more information, see the **adminGroup** property in The MFT `agent.properties` file.

As a Java application, an agent is a z/OS UNIX System Services application that you can run from JCL by using the BFGAGSTP member, from a generated Managed File Transfer command PDSE library data set for an agent. For more information about how to create an MFT command PDSE library data set, and customize it for the required agent, see Creating an MFT Agent or Logger command data set.

Related concepts

Enabling MFT agents to connect to remote z/OS queue managers

Related reference

"Stopping an MFT agent on z/OS" on page 219

If you are running a Managed File Transfer Agent on z/OS as a started task from JCL, the agent accepts the z/OS operator commands **MODIFY** and **STOP**, in addition to the **fteStopAgent** command.

The MFT `agent.properties` file

Listing MFT agents

You can list the Managed File Transfer agents registered with a particular queue manager using the command line or the IBM MQ Explorer.

About this task

To list agents using the command line, see fteListAgents command.

To list agents using the IBM MQ Explorer, in the Navigator view click **Agents** under the coordination queue manager name.

If an agent is not listed by the **fteListAgents** command or is not displayed in the IBM MQ Explorer, use the diagnosis flowchart in the following topic to locate and fix the problem: What to do if your MFT agent is not listed by the **fteListAgents** command.

Related reference

fteListAgents: list the MFT agents for a coordination queue manager

MFT agent status values

fteShowAgentDetails

Stopping an MFT agent

You can stop a Managed File Transfer agent from the command line. When you stop an agent, you are quiescing the agent and allowing the agent to complete its current file transfer before stopping. You can also specify the **-i** parameter at the command line to stop an agent immediately. When the agent has stopped, you cannot use that agent to transfer files until you restart it.

Before you begin

If you want to check the names of the agents associated with a queue manager, you can list agents by using the IBM MQ Explorer or the command line, see fteListAgents command.

About this task

To stop an agent from the command line, see fteStopAgent.

If an agent is stopped in a controlled manner using the **fteStopAgent**, then the agent does not accept any new managed transfer requests and waits for any in-progress transfers to complete before it actually shuts itself down. From IBM MQ 9.3.0, to show that the agent is still in a transient state, and therefore has not yet shut down and cannot yet be restarted, the agent enters the STOPPING state until any ongoing transfers are complete. This status appears in the output of the **fteListAgents** and **fteShowAgentDetails** commands, and in MFT REST API queries, and in the MFT plugin's **Agents** view of IBM MQ Explorer.

Windows If you have configured your agent to run as a Windows service, running the **fteStopAgent** command also stops the Windows service. Alternatively, you can stop the agent by stopping the service by using the Windows Services tool. For more information, see the topic “Starting an MFT agent as a Windows service” on page 214.

Related tasks

“Starting an MFT agent” on page 213

You can start a Managed File Transfer Agent from the command line. In this case, the agent process stops when you log off the system.

Related reference

MFT agent status values

fteStopAgent

“Stopping an MFT agent on z/OS” on page 219

If you are running a Managed File Transfer Agent on z/OS as a started task from JCL, the agent accepts the z/OS operator commands **MODIFY** and **STOP**, in addition to the **fteStopAgent** command.

z/OS Stopping an MFT agent on z/OS

If you are running a Managed File Transfer Agent on z/OS as a started task from JCL, the agent accepts the z/OS operator commands **MODIFY** and **STOP**, in addition to the **fteStopAgent** command.

A started task is used because it runs under a specific user ID and is not affected by users logging off.

Note: Started tasks are typically run under an administrative user that might not have log-on privileges and so it is not possible to log on to the z/OS system as the user that the agent is running under. The **fteStartAgent**, **fteStopAgent**, **fteSetAgentTraceLevel** commands, and the **fteShowAgentDetails** command with the **-d** parameter specified, cannot be issued for that agent.

You can use the agent property **adminGroup** with Managed File Transfer agents on z/OS. You can define a security manager group, for example MFTADMIN and then add the started task userid and administrator TSO ids to this group. Edit the agent properties file and set the **adminGroup** property to be the name of this security manager group.

```
adminGroup=MFTADMIN
```

Members of this group can then issue the **fteStartAgent**, **fteStopAgent**, and **fteSetAgentTraceLevel** commands, and the **fteShowAgentDetails** command with the **-d** parameter specified, for the agent that is running as a started task.

For more information, see the **adminGroup** property in The MFT `agent.properties` file .

Controlled agent shutdown by using the z/OS MODIFY command (F)

The **MODIFY** command allows you to stop an agent in a controlled way as an alternative to the **fteStopAgent** command. The agent completes any transfers currently in progress but the agent does not start any new transfers.

For example:

```
F job_name,APPL=STOP
```

where *job_name* is the job that the agent process is running under.

Immediate agent shutdown by using the z/OS STOP command (P)

The **STOP** command is equivalent to an immediate stop by using the **fteStopAgent** command with the **-i** parameter. The agent is stopped immediately even if the agent is currently transferring a file.

For example:

```
P job_name
```

where *job_name* is the job that the agent process is running under.

Related concepts

Enabling MFT agents to connect to remote z/OS queue managers

Related reference

"Starting an MFT agent on z/OS" on page 217
On z/OS, in addition to running the **fteStartAgent** command from a z/OS UNIX System Services session, you can start an agent as a started task from JCL without the need for an interactive session.

The MFT `agent.properties` file

Starting a new file transfer

You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

About this task

To start a new file transfer from the command line, see fteCreateTransfer command.

To start a new file transfer by using the **Create New Managed File Transfer** wizard in IBM MQ Explorer, use the following steps:

Procedure

1. In the Navigator view, click **Managed File Transfer**. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are displayed in the Navigator view. Expand the name of the coordination queue manager that the agent you want to use for the transfer is registered against. If you are currently connected to a coordination queue manager other than the one you want to use for the transfer, right-click that coordination queue manager name in the Navigator view and click **Disconnect**. Then right-click the name of the coordination queue manager you want to use and click **Connect**.
3. Start the **Create New Managed File Transfer** wizard by using either of the following methods:
   a) Right-click the name of any of the following nodes in the Navigator view: the relevant coordination queue manager, **Transfer Templates**, **Transfer Log**, or **Pending Transfers**. Then click **New Transfer** to start the wizard.
   b) Click **File** > **New** > **Other** > **Managed File Transfer Wizards** > **New Transfer Wizard**
4. Follow the instructions on the wizard panels. There is also context-sensitive help provided for each panel. To access the context-sensitive help on Windows, press F1. On Linux, press `Ctrl`+F1 or `Shift`+F1.

Related concepts

"Using transfer definition files" on page 221
You can specify a transfer definition file which can be used to create a file transfer. The transfer definition file is an XML file that defines some or all of the information required to create the transfer.

Related tasks

"Creating a scheduled file transfer" on page 223
You can schedule a new file transfer either from the IBM MQ Explorer, or from the command line. The scheduled transfer can contain single files or multiple files in a group. You can perform a scheduled file transfer once or repeat the transfer multiple times.

"Triggering a file transfer" on page 225
You can set certain trigger conditions on a file transfer that must be true before that transfer can take place. If the triggering conditions are not true, the file transfer does not take place and a log message is optionally submitted to record the fact the transfer did not happen. The file transfer request is then discarded. For example, you can set up a file transfer that takes place only if a named file on the system where the source agent is located is over a specified size, or if a particular named file exists on the system where the source agent is located. You can set up a triggered file transfer from either the IBM MQ Explorer or from the command line.

"Setting a timeout for recovery of stalled transfers" on page 327

You can set a transfer recovery timeout for stalled file transfers that applies to all the transfers for a source agent. You can also set a transfer recovery timeout for an individual transfer. If you set a specific amount of time, in seconds, during which a source agent keeps trying to recover a stalled file transfer and the transfer is not successful when the agent reaches the timeout, the transfer fails.

Related reference

fteCreateTransfer: start a new file transfer

File transfer request message format

Guidelines for transferring files

Using transfer definition files

You can specify a transfer definition file which can be used to create a file transfer. The transfer definition file is an XML file that defines some or all of the information required to create the transfer.

Transfer definition files are useful when you want to specify multiple source files and multiple destination files in a single transfer operation. You can use a transfer definition file to submit a complex file transfer. You can reuse and share the transfer definition file.

You can use two formats for a transfer definition file, and while these formats vary slightly, both conform to the `FileTransfer.xsd` schema. You can find this schema in the `samples\schema` directory of the Managed File Transfer installation.

The following two formats of transfer definition files are supported:

- A definition of the source and destination files for a transfer. This definition uses a **transferSpecifications** element as the root.
- A definition of the entire transfer, including source and destination files and the source and destination agents. This definition uses a **request** element as the root.
  - Files with this format can be generated from the **fteCreateTransfer** command by using the **-gt** parameter.

The following example shows a transfer definition file format that specifies only the source and destination files for a transfer:

```
<?xml version="1.0" encoding="UTF-8"?>
<transferSpecifications xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
  <item checksumMethod="MD5" mode="text">
    <source recursive="false" disposition="leave">
      <file>textTransferTest.txt</file>
    </source>
    <destination type="directory" exist="overwrite">
      <file>c:\targetfiles</file>
    </destination>
  </item>
</transferSpecifications>
```

To submit this format of transfer definition file you must specify the source and destination agents on the command line:

```
fteCreateTransfer -sa AGENT1 -sm agent1qm -da AGENT2 -dm agent2qm -td
  c:\definitions\example1.xml
```

The following example is a transfer definition file format that specifies all information required for a transfer:

```
<?xml version="1.0" encoding="UTF-8"?>
<request version="3.00" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
  <managedTransfer>
    <originator>
      <hostName>example.com.</hostName>
      <userID>fteuser</userID>
    </originator>
    <sourceAgent agent="AGENT1" QMgr="agent1qm"/>
```

```
    <destinationAgent agent="AGENT2" QMgr="agent2qm"/>
    <transferSet>
      <item mode="binary" checksumMethod="MD5">
        <source recursive="false" disposition="leave">
          <file>c:\sourcefiles\*.jpg</file>
        </source>
        <destination type="directory" exist="error">
          <file>/targetfiles/images</file>
        </destination>
      </item>
    </transferSet>
  </managedTransfer>
</request>
```

You can generate a file with this format by using the **-gt** parameter on the **fteCreateTransfer** command. When you submit a transfer definition file with this format, you do not need to specify anything else on the command line:

```
fteCreateTransfer -td c:\definitions\example2.xml
```

You can override the source and destination agent information about the command line by passing in the normal parameters in addition to the transfer definition file. For example:

```
fteCreateTransfer -da AGENT9 -dm agent9qm -td c:\definitions\example2.xml
```

This example uses the command-line options to override the destination agent defined inside the transfer definition file with **AGENT9** and the destination queue manager defined in the transfer definition file as **agent9qm**.

Both of the formats described can contain one or more <item> elements. For further information about the <item> element, see File transfer request message format. Each of these transfer items defines a source and destination file pair with additional attributes to control the behavior of the transfer. For example, you can specify the following behavior:

- Whether the transfer uses a checksum
- Whether the transfer is text or binary
- Whether to delete the source file after the transfer has completed
- Whether to overwrite the destination file if the file exists

An advantage of using transfer definition files is that you can specify additional options that are not available from the command line. For example, when you are carrying out message-to-file transfers, you can specify the groupId attribute by using a transfer definition file. This attribute specifies the IBM MQ group ID of the messages that are read from the queue. Another advantage of transfer definition files is that you can specify different options for each file pair. For example, you can specify whether a checksum is used, or whether the file is transferred in text or binary mode, on a file-by-file basis. If you use the command line, the same options apply for every file in a transfer.

For example:

```
  <item checksumMethod="none" mode="binary">
    <source disposition="leave">
      <file>c:\sourcefiles\source1.doc</file>
    </source>
    <destination type="file" exist="error">
      <file>c:\destinationfiles\destination1.doc</file>
    </destination>
  </item>

  <item checksumMethod="MD5" mode="text">
    <source disposition="delete">
      <file>c:\sourcefiles\source2.txt</file>
    </source>
    <destination type="file" exist="overwrite">
      <file encoding="UTF8" EOL="CRLF">c:\destinationfiles\destination2.txt</file>
    </destination>
  </item>

  <item checksumMethod="none" mode="text">
```

```
    <source recursive="false" disposition="leave">
      <file>c:\originfiles\source3.txt</file>
    </source>
    <destination type="file" exist="overwrite">
      <file>c:\targetfiles\destination3.txt</file>
    </destination>
  </item>
```

z/OS You can use items to transfer a file from a distributed system to a z/OS system:

z/OS

```
  <item checksumMethod="none" mode="text">
    <source recursive="false" disposition="leave">
      <file>textTransferTest.txt</file>
    </source>
    <destination type="dataset" exist="overwrite">
      <file encoding="IBM-1047">//TEXT.TRANS.TEST</file>
    </destination>
  </item>
```

z/OS This example transfers the file `textTransferTest.txt` from the source agent to the data set `//TEXT.TRANS.TEST` on the destination agent in text mode. This transfer converts the source data from the default encoding of the source agent (no source encoding attribute is specified) to code page: IBM-1047.

Creating a scheduled file transfer

You can schedule a new file transfer either from the IBM MQ Explorer, or from the command line. The scheduled transfer can contain single files or multiple files in a group. You can perform a scheduled file transfer once or repeat the transfer multiple times.

About this task

You can set up a file transfer schedule to occur once, or to occur at the following intervals:

- Every minute
- Hourly
- Daily
- Weekly
- Monthly
- Yearly

You can then specify the occurrences to stop at the following points:

- At a defined time and date
- After a defined number of occurrences

Alternatively, you can specify that the occurrences continue forever.

If a scheduled transfer runs at the same time each day, use the **adjustScheduleTimeForDaylightSaving** attribute in the agent properties file to adjust the time the schedule takes place when the clocks change.

The default value of **adjustScheduleTimeForDaylightSaving** is `True`. Set **adjustScheduleTimeForDaylightSaving** to `False` when you want a scheduled transfer to run at the exact specified clock time and not shift during daylight saving changes. This change prevents the transfer from moving an hour earlier or later after Daylight Saving Time adjustments, which is important when such timing shifts are not needed and might disrupt dependent processes. See The MFT `agent.properties` file for more information.

To create a new scheduled file transfer using the command line, use the scheduling parameters (**-tb**, **-ss**, **-oi**, **-of**, **-oc**, and **-es**) for the fteCreateTransfer command.

To create a new scheduled file transfer using the **Create New Managed File Transfer** wizard in IBM MQ Explorer, use the following steps:

Procedure

1. In the Navigator view, click **Managed File Transfer**. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are displayed in the Navigator view. Expand the name of the coordination queue manager that the agent you want to use for the transfer is registered against. If you are currently connected to a coordination queue manager other than the one you want to use for the transfer, right-click that coordination queue manager name in the Navigator view and click **Disconnect**. Then right-click the name of the coordination queue manager you want to use and click **Connect**.
3. Start the **Create New Managed File Transfer** wizard using either of the following methods:
   a) Right-click the name of any of the following nodes in the Navigator view: the relevant coordination queue manager, **Transfer Templates**, **Transfer Log**, or **Pending Transfers**. Then click **New Transfer** to start the wizard.
   b) Click **File** > **New** > **Other** > **Managed File Transfer Wizards** > **New Transfer Wizard**
4. Follow the instructions on the wizard panels. Ensure that you select the **Enable schedule transfer** check box and enter your schedule details on the **Schedule** tab. Scheduled file transfers start within a minute of the schedule start time, if there are no problems that might affect the transfer. For example, there might be issues with your network or agent that prevent the scheduled transfer starting. There is context-sensitive help provided for each panel. To access the context-sensitive help on Windows, press F1. On Linux, press `Ctrl`+`F1` or `Shift`+`F1`.

Results

For information about the messages involved in scheduled file transfers, see Scheduled file transfer log message formats.

Working with pending file transfers

You can view scheduled file transfers that are pending from the IBM MQ Explorer. The **Pending Transfers** window displays all of the pending transfers registered with the coordination queue manager that you are currently connected to.

About this task

To view the status of a scheduled file transfer that has not yet started, use the following steps:

Procedure

1. Expand **Managed File Transfer** in the Navigator view. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are displayed in the Navigator view. Expand the name of the coordination queue manager that you have used for the scheduled transfer. If you want to change which coordination queue manager you are connected to, right-click the name of the coordination queue manager you want to use in Navigator view and click **Connect**.
3. Click **Pending Transfers**. The **Pending Transfers** window is displayed in the Content view.
4. The **Pending Transfers** window displays the following details about your scheduled file transfers:
   a) **Name** The number of the scheduled file transfer. This number is automatically assigned.
   b) **Source** The name of the source agent.
   c) **Source File** The name of the file to be transferred on its host system.
   d) **Destination** The name of the destination agent.
   e) **Destination File** The name of the file after it is transferred to the destination system.

f) **Scheduled Start (selected time zone)** The time and date that the file transfer is scheduled to start in the administrator's selected time zone. To change the time zone displayed, click **Window** > **Preferences** > **IBM MQ Explorer** > **Managed File Transfer** and select an alternative time zone from the **Time zone:** list. Click **OK**.

g) **Repeat Every** If you have chosen to repeat the scheduled transfer, the specified interval that you want to repeat the transfer, expressed as a number.

h) **Repeat Type** If you have chosen to repeat the scheduled transfer, the type of repeat interval you have specified for the file transfer. The type can be one of the following values: `minutes`, `hours`, `days`, `weeks`, `months`, or `years`.

i) **Repeat Until** If you have chosen to repeat the scheduled transfer, the details of when you want the repeating file transfer to stop. For example, a specified date and time, or after a specified number of occurrences.

Results

To refresh what is displayed in the **Pending Transfers** window, click the Refresh button on the Content view toolbar.

To cancel a pending file transfer, right-click the particular transfer and click **Cancel**. Canceling a transfer completely discards the file transfer request.

Triggering a file transfer

You can set certain trigger conditions on a file transfer that must be true before that transfer can take place. If the triggering conditions are not true, the file transfer does not take place and a log message is optionally submitted to record the fact the transfer did not happen. The file transfer request is then discarded. For example, you can set up a file transfer that takes place only if a named file on the system where the source agent is located is over a specified size, or if a particular named file exists on the system where the source agent is located. You can set up a triggered file transfer from either the IBM MQ Explorer or from the command line.

About this task

You can monitor a resource continually for a trigger condition to be satisfied. For further information about resource monitoring see: “Monitoring MFT resources” on page 230.

There are three different triggering conditions that you can set. The conditions are as follows:

- If a particular file exists on the same system as the source agent
- If a particular file does not exist on the same system as the source agent
- If a particular file is over a certain size on the system where the source agent is located (the size can be expressed in bytes, KB, MB, or GB). These units of measurement use the 2^{10} convention, for example 1 KB equals 1024 bytes and 1 MB equals 1024 KB.

The triggering types in the preceding list can be combined in two ways:

- For a single condition, you can specify more than one file on the system where the source agent is located. This triggers the transfer if any one of the specified files meets the condition (Boolean operator OR).
- You can specify multiple conditions. This triggers the transfer only if all of the conditions are met (Boolean operator AND).

You can also combine a triggered transfer with a scheduled transfer. See Creating a scheduled file transfer for more information. In this case the trigger conditions are evaluated at the time the schedule is due to start, or for a repeating schedule every time the schedule is due to start.

Triggered transfers are not supported on protocol bridge agents.

To create a triggered file transfer by using the command line, use the **-tr** parameter on the fteCreateTransfer command.

To create a scheduled file transfer by using the **Create New Managed File Transfer** wizard in IBM MQ Explorer, use the following steps:

Procedure

1. In the Navigator view, click **Managed File Transfer**. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are displayed in the Navigator view. Expand the name of the coordination queue manager that you have used for the scheduled transfer. If you want to change which coordination queue manager you are connected to, right-click the name of the coordination queue manager you want to use in Navigator view and click **Connect**.
3. Start the **Create New Managed File Transfer** wizard by using either of the following methods:
   a) Right-click the name of any of the following nodes in the Navigator view: the relevant coordination queue manager, **Transfer Templates**, **Transfer Log**, or **Pending Transfers**. Then click **New Transfer** to open the wizard.
   b) Click **File** > **New** > **Other** > **Managed File Transfer Wizards** > **New Transfer Wizard**
4. Follow the instructions on the wizard panels. Ensure that you select the **Enable triggered transfer** check box on the **Triggers** tab and complete the fields on that tab to set up triggering. There is context-sensitive help provided for each panel. To access the context-sensitive help on Windows, press F1. On Linux, press **Ctrl+F1** or Shift+F1.

Monitoring file transfers that are in progress

You can monitor a file transfer that is in progress using the **Managed File Transfer - Current Transfer Progress** tab in IBM MQ Explorer. This file transfer can be one started from either IBM MQ Explorer or the command line. The tab also displays the progress of scheduled transfers at the point the scheduled transfers start.

About this task

If you want to use IBM MQ Explorer to monitor transfers associated with a coordination queue manager on a remote system, follow the instructions in the "Configuring IBM MQ Explorer to monitor a remote coordination queue manager" on page 227 topic.

Previous file transfer information is not retained after you stop and restart IBM MQ Explorer. At restart, the information about past transfers is cleared from the **Current Transfer Progress** tab. You can clear completed transfers using **Remove completed transfers** at any point when IBM MQ Explorer is open.

Procedure

After you have started a new file transfer using IBM MQ Explorer or the command line, you can monitor the progress of your transfer in the **Current Transfer Progress** tab. The following information is displayed for each transfer in progress:

a) **Source**. The name of the agent used to transfer the file from the source system.

b) **Destination**. The name of the agent used to receive the file at the destination system.

c) **Current file**. The name of the file currently being transferred. The part of the individual file that has already been transferred is displayed in B, KiB, MiB. GiB, or TiB along with total size of the file in parentheses. The unit of measurement displayed depends on the size of the file.

   B is bytes per second. KiB/s is kibibytes per second, where 1 kibibyte equals 1024 bytes. MiB/s is mebibytes per second, where 1 mebibyte equals 1 048 576 bytes. GiB/s is gibibytes per second where 1 gibibyte equals 1 073 741 824 bytes. TiB/s is tebibytes per second where 1 tebibyte equals 1 099 511 627 776 bytes.

d) **File number**. If you are transferring more than one file, this number represents how far through the total group of files the transfer is.

e) **Progress**. The progress bar shows how complete the current file transfer is as a percentage.

f) **Rate**. The rate the file is being transferred in KiB/s (kibibytes per second, where 1 kibibyte equals 1024 bytes.)

g) **Started (selected time zone)**. The time that the file transfer started, presented in the selected time zone of the administrator. To change the time zone displayed, click **Window** > **Preferences** > **IBM MQ Explorer** > **Managed File Transfer** and select an alternative time zone from the **Time zone:** list. Click **OK**.

If the transfer enters a recovery state while transferring the file, the started time updates to reflect the time that the file transfer resumed.

Results

This tab regularly refreshes its information automatically, but to force a refreshed view of what is displayed in the **Current Transfer Progress** tab, click **Refresh** on the Content view toolbar.

To delete file transfers from the **Current Transfer Progress** tab, click **Remove completed transfers** on the Content view toolbar. Clicking this button removes file transfer details from the tab only; it does not stop or cancel a current or scheduled transfer.

If you want to return to the **Current Transfer Progress** tab after closing it, you can display the tab by clicking **Window** > **Show View** > **Other** > **Other** > **Managed File Transfer - Current Transfer Progress**. Click **OK**.

What to do next

Additionally, it is possible to develop applications for custom file transfer monitoring. This can be accomplished by creating a subscription to the appropriate Managed File Transfer administrative topic (either programmatically or administratively), and the monitor application can then receive Managed File Transfer file transfer activity publications on the topic. For more information on the subscription topic and publication message format, see File transfer progress message examples.

Related tasks

“Configuring IBM MQ Explorer to monitor a remote coordination queue manager” on page 227
Use IBM MQ Explorer to monitor file transfers associated with a coordination queue manager running on a remote system. You require a system that is capable of running the IBM MQ Explorer. The IBM MQ Explorer component needs to be installed to be able to connect to the remote coordination queue manager.

“Viewing the status of file transfers in the Transfer Log” on page 228
You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

Configuring IBM MQ Explorer to monitor a remote coordination queue manager

Use IBM MQ Explorer to monitor file transfers associated with a coordination queue manager running on a remote system. You require a system that is capable of running the IBM MQ Explorer. The IBM MQ Explorer component needs to be installed to be able to connect to the remote coordination queue manager.

About this task

Assumptions: Authority to connect to the remote coordination queue manager by configuring the queue manager to allow for remote connections.

For more information on how to configure this, see Connecting to a queue manager in client mode with channel authentication and Managing authorities for MFT-specific resources.

To monitor queue managers and file transfers between agents on a system that is not running Windows or Linux, configure the IBM MQ Explorer to connect to the remote system using the following steps:

Procedure

1. Start the local IBM MQ Explorer.
2. When IBM MQ Explorer is loaded, right-click on the **Managed File Transfer** folder and select **New configuration**.
3. Proceed through the wizard, selecting the Coordination and Commands queue manager, then define a name for the configuration.
4. Click **finish** to complete the definition.
5. When the definition is finished, right-click on the definition and select **Connect**.

Results

Now start IBM MQ Explorer and use it to monitor transfer activity for the Managed File Transfer network associated with the coordination queue manager.

Related tasks

"Monitoring file transfers that are in progress" on page 226
You can monitor a file transfer that is in progress using the **Managed File Transfer - Current Transfer Progress** tab in IBM MQ Explorer. This file transfer can be one started from either IBM MQ Explorer or the command line. The tab also displays the progress of scheduled transfers at the point the scheduled transfers start.

"Viewing the status of file transfers in the Transfer Log" on page 228
You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

Viewing the status of file transfers in the Transfer Log

You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

Procedure

1. Expand **Managed File Transfer** in the Navigator view and then expand the name of the coordination queue manager that you want to view the transfer log for.
2. Click **Transfer Log** in the Navigator view. The **Transfer Log** is displayed in the Content view.
3. The **Transfer Log** window displays the following details about your file transfers:
   a) **Source** The name of the agent on the system where the source file is located.
   b) **Destination** The name of the agent on the system you want to transfer the file to.
   c) **Completion State** The status of the file transfer. The state can be one of the following values: "Started", "In progress", "Successful", "Partially Successful", "Cancelled", or "Failed".
   d) **Owner** The user ID on the host that submitted the transfer request.
   e) **Started (selected time zone)** The time and date that the file transfer request was accepted by the Managed File Transfer agent, presented in the selected time zone of the administrator. To change the time zone displayed, click **Window** > **Preferences** > **IBM MQ Explorer** > **Managed File Transfer** and select an alternative time zone from the **Time zone:** list. Click **OK**.

f) **State Recorded (selected time zone)** (This column is not displayed by default. You can choose to display the column by using the **Configure Transfer Log Columns** window.) The time and date that the completion state was recorded, in the time zone selected by the administrator.

g) **Job Name** An identifier specified by the user by using the **-jn** parameter of **fteCreateTransfer** or in an Ant script

h) **Transfer ID** The unique identifier for the file transfer.

i) **Connect: Direct** Details about **Process Number**, **Process Name**, **Primary Node**, **Secondary Node**, **Source Type** and **Destination Type** are listed.

Results

Note: The internal format of the Transfer Log was changed in IBM MQ 8.0.0 Fix Pack 1 for APAR IC99545. As a result, if an IBM MQ Explorer is upgraded to V8.0.0.1 or later, and then restored to V8.0.0.0, no audit XML is displayed for transfers that took place while IBM MQ Explorer was at V8.0.0.1. The XML panel in the **Properties** window for these transfers will contain an empty text box.

To view further details about a completed transfer, expand the transfer that you are interested in by clicking the plus sign (+). You can then see all of the source and destination file names included in that transfer. However, if the transfer is currently in progress and consists of many files, you can view only the files that have already been transferred so far.

To refresh what is displayed in the **Transfer Log**, click the **Refresh** button on the Content view toolbar. The file transfer information in the Transfer Log remains in the log after you stop and restart the IBM MQ Explorer. If you want to delete all completed file transfers from the log, click **Remove Completed Transfers** on the Content view toolbar.

To delete an individual completed file transfer from the log, right-click the transfer and click **Delete**. If you delete a transfer, it does not stop or cancel a transfer that is in progress or that has been scheduled; you are deleting only the stored historical data.

To copy the unique identifier of a transfer to the clipboard, right-click that transfer and click **Copy ID**.

The metadata and the complete audit XML for the transfer are available from the pop-up menu, under the **Properties** action.

Related tasks

“Monitoring file transfers that are in progress” on page 226
You can monitor a file transfer that is in progress using the **Managed File Transfer - Current Transfer Progress** tab in IBM MQ Explorer. This file transfer can be one started from either IBM MQ Explorer or the command line. The tab also displays the progress of scheduled transfers at the point the scheduled transfers start.

“Configuring the Transfer Log” on page 230
You can configure what information is displayed and how information is displayed in the **Transfer Log** in the IBM MQ Explorer.

“Setting a timeout for recovery of stalled transfers” on page 327
You can set a transfer recovery timeout for stalled file transfers that applies to all the transfers for a source agent. You can also set a transfer recovery timeout for an individual transfer. If you set a specific amount of time, in seconds, during which a source agent keeps trying to recover a stalled file transfer and the transfer is not successful when the agent reaches the timeout, the transfer fails.

Configuring the Transfer Log

You can configure what information is displayed and how information is displayed in the **Transfer Log** in the IBM MQ Explorer.

About this task

To rearrange the order of the columns in the **Transfer Log**, click the title of the column you want to move and drag the column to its new position. The new column order is retained only until you next stop and restart the IBM MQ Explorer.

To filter entries in the **Transfer Log**, enter a string in the **Filter the displayed log entries** field. To restore all of the entries to the log, delete the string you entered from the field. You can use any valid Java regular expression in this field. For more information, see Regular expressions used by MFT.

To customize which columns are displayed in the Transfer Log, use **Configure Transfer Log Columns** . Use the following steps to start and use the **Configure Transfer Log Columns** window.

Procedure

1. Ensure that you have the **Transfer Log** open in the Content view. Click **Configure Transfer Log Columns** on the Content view toolbar. The **Configure Transfer Log Columns** window opens.
2. To customize your view of the **Transfer Log**, select or clear individual check boxes for the columns you want to show or hide. You can click **Select All**, then **OK** to select all of the check boxes or **Deselect All**, then **OK** to clear all of the check boxes.

Related tasks

"Monitoring file transfers that are in progress" on page 226
You can monitor a file transfer that is in progress using the **Managed File Transfer - Current Transfer Progress** tab in IBM MQ Explorer. This file transfer can be one started from either IBM MQ Explorer or the command line. The tab also displays the progress of scheduled transfers at the point the scheduled transfers start.

"Viewing the status of file transfers in the Transfer Log" on page 228
You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

Monitoring MFT resources

You can monitor Managed File Transfer resources; for example, a queue or a directory. When a condition on this resource is satisfied, the resource monitor starts a task, such as a file transfer. You can create a resource monitor by using the **`fteCreateMonitor`** command or the **Monitors** view in the Managed File Transfer plug-in for IBM MQ Explorer.

About this task

Managed File Transfer resource monitoring uses the following terminology:

Resource monitor
: A resource monitor is process that polls a resource (such as a directory or queue) at a predefined regular interval to see if the resource contents have changed. If they have, the contents are compared with the set of conditions for this monitor. If there is a match, the task for this monitor is started.

Resource
: The system resource that the resource monitor examines at every poll interval to be compared with the trigger conditions. Queues, directories, or nested directory structures can be the monitored resource.

Condition and Trigger condition
A condition is an expression that is evaluated (typically against the content of the monitored resource). If the expression evaluates to true, the condition contributes to the overall trigger condition.

The trigger condition is the overall condition, which is satisfied when all conditions are satisfied. When the trigger condition is satisfied the task can proceed.

Task
A task is the operation that is started when the trigger condition or set of conditions is satisfied. Supported tasks are file transfer and command call.

Trigger file
A trigger file is a file that is placed in a monitored directory to indicate that a task (typically a transfer) can begin. For example, it might indicate that all the files to be processed have arrived in a known location and can be transferred or otherwise acted upon. The name of the trigger file can be used to specify the files to be transferred by using variable substitution. For more information, see "Customizing MFT resource monitor tasks with variable substitution" on page 242.

The trigger file is also known as ready file or go file. However, in this documentation it is usually referred to as the trigger file.

Resource monitoring is not supported on protocol bridge agents or Connect:Direct bridge agents.

Related concepts
Guidance for configuring an MFT resource monitor to avoid overloading an agent
Related reference
`fteCreateMonitor`: create an MFT resource monitor
fteListMonitors: list MFT resource monitors
fteDeleteMonitor: delete an MFT resource monitor
MFT monitor request message formats

MFT resource monitoring concepts

An overview of the key concepts of the Managed File Transfer resource monitoring feature.

Resource monitors

You create a resource monitor by using the **`fteCreateMonitor`** command, which creates and starts a new resource monitor from the command line. The resource monitor is associated with a Managed File Transfer agent, and is only active when that agent is running. When the monitoring agent stops, so does the resource monitor. If the agent is already running when the resource monitor is created, the resource monitor starts immediately. The monitoring agent must also be the source agent of the task that is initiated by the resource monitor.

Resource monitor names must be unique within their agent. The resource monitor name must be a minimum of one character in length and must not contain asterisk (*), percent (%) or question mark (?) characters. The case in which a resource monitor name is supplied is ignored and the resource monitor name is converted to uppercase. If you try to create a resource monitor with a name that is already present, the request is ignored and the attempt is logged to the resource monitor log topic.

Note: You cannot create a resource monitor with a task definition that contains scheduled transfers.

Before IBM MQ 9.3.0, the only way to stop a resource monitor is by stopping the agent that is running the monitor operation. To restart a resource monitor, you must restart the agent altogether. From IBM MQ 9.3.0, you can start and stop resource monitors without needing to stop or restart an agent. For more information, see "Starting and stopping resource monitors" on page 234.

There is no restriction on the number of resource monitors that can be created on an agent, and all run with the same priority. Consider the implications of overlapping monitored resources, conflicting trigger conditions and how frequently the resources are polled.

Overlapping resource monitors can cause:

- Possible contention on the source location/items.
- Possible duplicate transfer requests for same source items.
- Unexpected errors or failures for transfers due to conflicts of source items.

If multiple monitors scan the same location and can trigger on the same items, you can potentially end up with the problem of two different monitors submitting managed transfer requests for the same item.

Resource monitors look at the contents of resources after every poll interval period. The contents of the resource are compared with the trigger conditions and if those conditions are satisfied, the task associated with the resource monitor is called.

The task is started asynchronously. If there is a condition match, and the task is started, the resource monitor continues to poll for further changes to the resource contents. So for example, if a match occurred because a file called `reports.go` arrived in a monitored directory, the task would be started once. At the next poll interval, even if the file still exists, the task is not started again. However, if the file is deleted and then placed in the directory again, or the file is updated (such that the last modified date attribute is changed), the next trigger condition check causes the task to be called again.

Before IBM MQ 9.1.5, if a resource monitor performs a poll that takes longer than the polling interval, this means that the next poll starts as soon as the current one finishes with no gap in between, which could have an effect on how quickly resource monitors submit work to an agent. This could cause performance issues if the items that are found during the first poll are still there when the second one takes place.

The resource monitor uses the ScheduledExecutorService and initiates the next poll only after the completion of the previous poll plus the configured poll interval time. This means that there will always be a gap in between the poll intervals, rather than having another poll starting straight away after the previous poll if the poll time was longer than the poll interval.

If a file has failed to transfer, you can clear the resource monitor history, which allows another transfer request to be submitted without the need to either delete the file and place it in the directory again, or update the file to change its last modified date attribute. Clearing the history is useful, for example, in situations where there is need for the file to be transferred but modifying the file is not possible. For more information, see “Clearing resource monitor history” on page 259.

Resources

Resource monitors in Managed File Transfer can poll the contents of the following two types of resource:

Directories or nested directory structures

A common scenario is to monitor a directory for the presence of a trigger file. An external application might be processing multiple files and placing them in a known source directory. When the application has completed its processing, it indicates that the files are ready to be transferred, or otherwise acted upon, by placing a trigger file into a monitored location. The trigger file can be detected by a Managed File Transfer resource monitor and the transfer of those files from the source directory to another Managed File Transfer Agent is initiated.

By default, the specified directory is monitored. To also examine sub-directories set the recursion level in the **`fteCreateTransfer`** command.

Two examples of monitoring a directory are as follows:

- Monitor for a trigger file (for example `trigger.file`) and then transfer a wildcard (for example, `*.zip`).
- Monitor for `*.zip` and then transfer `${FilePath}` (for example, the file that triggered the transfer). For more information about variable substitution, see “Customizing MFT resource monitor tasks with variable substitution” on page 242.

Note: Do not create a monitor that monitors for `*.zip`, and then transfers `*.zip`. The monitor tries to start a transfer of `*.zip` for every .zip file on your system. That is, the monitor generates * number of transfers for `*.zip`.

For an example of creating a resource monitor to monitor a directory, see “Monitoring a directory and using variable substitution” on page 239.

IBM MQ queues

An example of monitoring a queue is that an external application might be generating messages and placing them on a known queue with the same group ID. When the application has completed putting messages on the queue, it indicates that the group is complete. The complete group of messages can be detected by a Managed File Transfer resource monitor and the transfer of the group of messages from the source queue to a file is initiated. For an example of creating a resource monitor to monitor a queue, see "Example: Configuring an MFT resource" on page 241.

Note: You can specify only one monitor per queue. If you specify more than one monitor to poll an IBM MQ queue, unpredictable behavior occurs.

Monitoring data sets is not supported.

Conditions and trigger conditions

The condition is met when the resource contains a value that matches some other string or pattern. Conditions can be one of the following:

- Match on file name (pattern)
- No match on file name (pattern)
- File size
- Match if file size remains the same for a number of polls

File name matching can be expressed as:

- Exact string match
- Simple wildcard match as described in Using wildcard characters with MFT
- Regular expression match

File names can also be excluded from file name matching by using a wildcard or Java regular expression that identifies file names that are never matched.

When a matching file is detected, its last modified time stamp is retained. If subsequent polls detect that the file has been changed, the trigger condition is satisfied again, and the task is started. If the condition is to detect when a file does not exist, if no file in the monitored directory matches the file name pattern, the task is started. If a file is then added to the directory that does match the file name pattern, the task is only started if the file is then deleted.

Tasks

Managed File Transfer supports the following two types of task that you can configure to be started by resource monitors:

File transfer tasks

File transfer tasks are defined in the same way as any other file transfer. A useful way to generate the task XML required by a monitor is to run the fteCreateTransfer command with the **-gt** parameter. This command generates a task definition as an XML document, including the transfer specification. You then pass the name of the task XML document as the value for the **-mt** parameter on the fteCreateMonitor command. When the **fteCreateMonitor** is run, it reads the task XML document. After the **fteCreateMonitor** is run, any changes that are made to the task XML file are not used by the monitor.

When using a file transfer task, you can select how many trigger conditions are batched into a task. The default is for one trigger condition to start one task. You can run the fteCreateMonitor command with the **-bs** option to select the number of trigger conditions that are batched together into one task.

Command tasks

Command tasks can run Ant scripts, call executable programs, or run JCL jobs. For more information, see "Configuring MFT monitor tasks to start commands and scripts" on page 235.

Trigger files

You can use the contents of a trigger file in a resource monitor to define a set of files to transfer in a single transfer request. Each time a matching trigger file is detected, its contents are parsed for source file paths and optionally for destination file paths. These file paths are then used to define file items in the task transfer XML file that you specify, which is submitted as single transfer request to the agent. The definition of the resource monitor determines whether trigger content is enabled.

The format of each trigger file is a single file path to transfer on each line of text. The default format for the line is either a single source file path or a source and destination file path separated by a comma.

For more information and examples, see "Using a trigger file" on page 250.

Starting and stopping resource monitors

Before IBM MQ 9.3.0, the only way to stop a resource monitor is by stopping the agent that is running the monitor operation. To restart a resource monitor, you must restart the agent altogether. For more information, see "Starting an MFT agent" on page 213 and "Stopping an MFT agent" on page 218.

From IBM MQ 9.3.0, you can start and stop resource monitors without needing to stop or restart an agent by using the **fteStartMonitor** and **fteStopMonitor** commands. This is useful, for example, in the following situations:

- If an agent has multiple resource monitors, and only some of them have encountered errors but the remaining resource monitors are still working fine so you just want to restart the failed resource monitors.
- If you want to stop a resource monitor to carry out some maintenance work, or if the resource monitor is not required for certain time and you do not want it to run unnecessarily, thus consuming precious system resources.

For more information, see "Starting an MFT resource monitor" on page 255 and "Stopping an MFT resource monitor" on page 256.

Table 9. Behavior of a resource monitor depending on which command is run

Command	Behavior of resource monitor
fteStartMonitor	If the agent is running, then the resource monitor is started if it is currently stopped.
fteStopMonitor	If the agent is running, then the resource monitor is stopped if it is currently started.
fteStartAgent	The resource monitor is started as part of agent start up, regardless of earlier calls to **fteStopMonitor**.
fteStopAgent	Any resource monitors that are running are stopped.

Backing up and restoring resource monitors

You can back up the resource monitors that you have already defined so that you can re-use them in the future. There are various options that you can use as follows:

- Use the **fteCreateMonitor** command with the **-ox** parameter to export a resource monitor configuration to an XML file, and with the **-ix** parameter to restore a resource monitor by importing the resource monitor configuration from an XML file.
- Use the **fteListMonitors** command with the **-ox** to export the definition for a single resource monitor to an XML file.

- Use the **fteListMonitors** command with the **-od** to export multiple resource monitor definitions to a specified directory. Each resource monitor definition is saved to separate XML file. You can also use the **-od** option to export a single resource monitor definition to a specified directory.

For more information, see "Backing up and restoring MFT resource monitors" on page 257.

Resource monitor logging

Managed File Transfer includes resource monitor logging. For more information, see "Logging MFT resource monitors" on page 253.

Related concepts

"Customizing MFT resource monitor tasks with variable substitution" on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related tasks

"Configuring MFT monitor tasks to start commands and scripts" on page 235
Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

"Example: Configuring an MFT resource" on page 241
You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **-mq** parameter with the **fteCreateMonitor** command.

"Monitoring a queue and using variable substitution" on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

Related reference

fteCreateMonitor: create an MFT resource monitor

fteListMonitors: list MFT resource monitors

fteDeleteMonitor: delete an MFT resource monitor

Configuring MFT monitor tasks to start commands and scripts

Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

About this task

The file path to the executable program, Ant script, or JCL job that you want the monitoring agent to call must be included in the `commandPath` of the monitoring agent. For information about the command path property, see commandPath MFT property.

You can create the task definition XML document in one of the following ways:

- Create the task definition XML document manually according to the `FileTransfer.xsd` schema.
- Use a generated XML document as the basis for your task definition.

Whether you want a transfer task or a command task, the task definition must start with a `<request>` root element. The child element of `<request>` must be either `<managedTransfer>` or

<managedCall>. You would typically choose <managedCall> when there is a single command or script to run, and <managedTransfer> if you want the task to include a file transfer and optionally up to four command calls.

Procedure

- To create the task definition XML document manually according to the FileTransfer.xsd schema, see "Creating a task definition XML manually according to the schema" on page 236.
- To create a task definition by modifying a generated document, edit the XML document generated by the **fteCreateTransfer -gt** parameter. For more information, see "Creating a task definition document by modifying a generated document" on page 238.

Creating a task definition XML manually according to the schema

You can manually create a task definition XML file according to the schema FileTransfer.xsd.

About this task

The schema FileTransfer.xsd can be found in the *MQ_INSTALLATION_PATH*/mqft/samples/schema. For more information about this schema, see File transfer request message format.

Example

The following example shows an example task definition XML document saved as cleanuptask.xml, which uses the <managedCall> element to call an Ant script called RunCleanup.xml. The RunCleanup.xml Ant script must be located on the commandPath of the monitoring agent.

```
<?xml version="1.0" encoding="UTF-8"?>
<request version="4.00" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedCall>
     <originator>
       <hostName>hostName</hostName>
       <userID>userID</userID>
       <mqmdUserID>mqmdUserID</mqmdUserID>
     </originator>
     <agent QMgr="QM1" agent="AGENT1"/>
     <reply QMGR="QM1">reply</reply>
     <transferSet priority="1">
       <metaDataSet>
         <metaData key="name1">value1</metaData>
       </metaDataSet>
       <call>
           <command name="RunCleanup.xml" type="antscript" retryCount="2"
          retryWait="30" successRC="0">
           <target>check_exists</target>
           <target>copy_to_archive</target>
           <target>rename_temps</target>
             <target>delete_files</target>
           <property name="trigger.filename" value="${FileName}"/>
           <property name="trigger.path" value="${FilePath}"/>
           </command>
       </call>
     </transferSet>
     <job>
       <name>JOBCLEAN1</name>
     </job>
   </managedCall>
</request>
```

The <agent> element specifies the Managed File Transfer Agent that is configured with the named Ant script on its commandPath.

The <call><command>... structure defines the executable or script you want to run. The command takes an optional type attribute that can have one of the following values:

antscript
Run an Ant script in a separate JVM.

executable
Invoke an executable program.

jcl
Invoke a JCL job.

os4690background
This value must be for an IBM MQ 8.0 or earlier Managed File Transfer agent, running on the 4690 operating system. This operating system is no longer supported and the value is no longer used.

If you omit the `type` attribute, the default value executable is used.

The `name` attribute specifies the name of the Ant script, executable, or JCL job you want to run, without any path information. The agent searches for the script or program in the locations specified by the commandPath property in the agent's `agent.properties` file.

The `retrycount` attribute specifies the number of times to try calling the program again if the program does not return a success return code. The value assigned to this attribute must not be negative. If you do not specify the `retrycount` attribute, a default value of zero is used.

The `retrywait` attribute specifies the time to wait, in seconds, before trying the program invocation again. The value assigned to this attribute must not be negative. If you do not specify the `retrywait` attribute, a default value of zero is used.

The `successrc` attribute is an expression used to determine when the program invocation successfully runs. The process return code for the command is evaluated using this expression. The value can be composed of one or more expressions combined with a vertical bar (|) character to signify Boolean OR, or an ampersand (&) character to signify Boolean AND. Each expression can be one of the following types of expression:

- A number to indicate an equality test between the process return code and the number.
- A number prefixed with a greater than character (>) to indicate a greater-than test between the number and the process return code.
- A number prefixed with a less than character (<) to indicate a less-than test between the number and the process return code.
- A number prefixed with an exclamation point character (!) to indicate a not-equal-to test between the number and the process return code. For example: >2&<7&!5|0|14 is interpreted as the following return codes being successful: 0, 3, 4, 6, 14. All other return codes are interpreted as being unsuccessful.

If you do not specify the `successrc` attribute, a default value of zero is used. This means that the command is judged to have successfully run if, and only if, it returns a code of zero.

For an Ant script, you would typically specify <target> and <property> elements. The <target> element values must match the target names in the Ant script.

For executable programs, you can specify <argument> elements. Nested argument elements specify arguments to pass to the program that is being called as part of the program invocation. The program arguments are built from the values specified by the argument elements in the order that the argument elements are encountered. You can specify zero or more argument elements as nested elements of a program invocation.

The administrator defines and starts the monitor as normal using the task definition XML document that includes the <managedCall> element. For example:

```
fteCreateMonitor -ma AGENT1 -mm QM1 -md /monitored -mn MONITOR01 -mt
   /tasks/cleanuptask.xml -pi 30 -pu seconds -tr match,*.go
```

The path to the transfer definition XML document must be on the local file system that you run the **fteCreateMonitor** command from (in this example `/tasks/cleanuptask.xml`). The `cleanuptask.xml` document is used to create the resource monitor only. Any tasks that the `cleanuptask.xml` document references (Ant scripts or JCL jobs) must be in the command path of the monitoring agent. When the monitor trigger condition is satisfied, any variables in the task definition XML are substituted with actual values from the monitor. So for example ${FilePath} is replaced in the

request message sent to the agent with `/monitored/cleanup.go`. The request message is put on the agent command queue. The command processor detects that the request is for a program call and starts the specified program. If a command of type `antscript` is called, a new JVM is started and the Ant task runs under the new JVM. For more information about using variable substitution, see Customizing tasks with variable substitution.

Related concepts

“Customizing MFT resource monitor tasks with variable substitution” on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related reference

File transfer request message format

commandPath MFT property

Creating a task definition document by modifying a generated document

You can create the monitor task definition document by modifying the XML document generated by the **`-gt`** option of **`fteCreateTransfer`**.

About this task

The generated document has a `<request>` followed by `<managedTransfer>` element. To convert this task definition to a valid `<managedCall>` structure, follow these steps:

Procedure

1. Replace the `<managedTransfer>` start and end tags with `<managedCall>` tags.
2. Remove any `<schedule>` element and child nodes.
3. Replace the `<sourceAgent>` start and end tags with `<agent>` to match the monitoring agent configuration details.
4. Remove `<destinationAgent>` and `<trigger>` elements.
5. Remove `<item>` elements.
6. Remove any `preSourceCall`, `postSourceCall`, `preDestinationCall`, or `postDestinationCall` elements.
7. Insert a new `<call>...</call>` structure within the `<transferSet>` element. This structure contains the command definition as shown in the following example:

```
<call>
          <command name="RunCleanup.xml" type="antscript" retryCount="2"
         retryWait="30" successRC="0">
          <target>check_exists</target>
          <target>copy_to_archive</target>
          <target>rename_temps</target>
            <target>delete_files</target>
          <property name="trigger.filename" value="${FileName}"/>
          <property name="trigger.path" value="${FilePath}"/>
          </command>
      </call>
```

Example

You can also retain the `<managedTransfer>` element including all the file transfer details, and insert up to four command calls. In this case you insert any selection of the following call elements between the `<metaDataSet>` and `<item>` elements:

preSourceCall
Call a program on the source agent before starting the transfer.

postSourceCall
Call a program on the source agent after completing the transfer.

preDestinationCall
Call a program on the destination agent before starting the transfer.

postDestinationCall
Call a program on the destination agent after completing the transfer.

Each of these elements takes the <command> element structure as described in the earlier example. The FileTransfer.xsd schema defines the types used by the various call elements.

The following example shows preSourceCall, postSourceCall, preDestinationCall, and postDestinationCall in a task definition document:

```
⋮
    <transferSet priority="1">
       <metaDataSet>
         <metaData key="key1">value1</metaData>
       </metaDataSet>
       <preSourceCall>
         <command name="send.exe" retryCount="0" retryWait="0" successRC="0"
           type="executable">
           <argument>report1.pdf</argument>
           <argument>true</argument>
         </command>
       </preSourceCall>
       <postSourceCall>
         <command name="//DO_IT.JCL" retryCount="0" retryWait="0" successRC="0"
           type="jcl">
           <argument>argument</argument>
         </command>
       </postSourceCall>
       <preDestinationCall>
         <command name="ant_script.xml" retryCount="0" retryWait="0" successRC="0"
           type="antscript">
           <target>step1</target>
           <property name="name" value="value"/>
         </command>
       </preDestinationCall>
       <postDestinationCall>
         <command name="runit.cmd" retryCount="0" retryWait="0" successRC="0" />
       </postDestinationCall>
       <item checksumMethod="none" mode="binary">
⋮
```

You can mix different types of command into the transfer. Argument, target, and property elements are optional.

Monitoring a directory and using variable substitution

You can monitor a directory using the **fteCreateMonitor** command. The value of a substitution variable can be substituted in the task XML definition and used to define the transfer behavior.

About this task

In this example, the source agent is called AGENT_HOP. The directory that AGENT_HOP monitors is called `/test/monitored`. The agent polls the directory every 5 minutes.

After a `.zip` file is written to the directory, the application that writes the file to the directory writes a trigger file to the same directory. The name of the trigger file is the same as the name of the `.zip` file, but has a different file extension. For example, after the file `file1.zip` is written to the directory, the file `file1.go` is written to the directory. The resource monitor monitors the directory for files that match the pattern `*.go` then uses variable substitution to request a transfer of the associated `.zip` file.

Procedure

1. Create the task XML that defines the task that the monitor performs when it is triggered.

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>blue.example.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_HOP" QMgr="QM_HOP" />
     <destinationAgent agent="AGENT_SKIP" QMgr="QM_SKIP" />
     <transferSet>
       <item mode="binary" checksumMethod="none">
         <source>
           <file>/test/monitored/${fileName{token=1}{separator=.}}.zip</file>
         </source>
         <destination type="file" exist="overwrite">
           <file>/out/${fileName{token=1}{separator=.}}.zip</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

The variables that are replaced with the values associated with the trigger file are highlighted in **bold**. This task XML is saved to the file `/home/USER1/task.xml`

2. Create a resource monitor to monitor the directory `/test/monitored`.

   Submit the following command:

```
fteCreateMonitor -ma AGENT_HOP -mm QM_HOP -md /test/monitored
                 -mn myMonitor -mt /home/USER1/task.xml
                 -tr match,*.go -pi 5 -pu minutes
```

3. A user or program writes the file `jump.zip` to the directory `/test/monitored`, then writes the file `jump.go` to the directory.
4. The monitor is triggered by the existence of the file `jump.go`. The agent substitutes the information about the trigger file into the task XML.

   This results in the task XML being transformed to:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>blue.example.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_HOP" QMgr="QM_HOP" />
     <destinationAgent agent="AGENT_SKIP" QMgr="QM_SKIP" />
     <transferSet>
       <item mode="binary" checksumMethod="none">
         <source>
           <file>/test/monitored/jump.zip</file>
         </source>
         <destination type="file" exist="overwrite">
           <file>/out/jump.zip</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

Results

The transfer defined by the task XML is performed. The `jump.zip` file is read from the `/test/monitored` directory by AGENT_HOP and is transferred to a file called `/out/jump.zip` located on the system where AGENT_SKIP is running.

Related concepts

“Customizing MFT resource monitor tasks with variable substitution” on page 242

When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related tasks

"Configuring MFT monitor tasks to start commands and scripts" on page 235
Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

Related reference

`fteCreateMonitor`: create an MFT resource monitor

Example: Configuring an MFT resource

You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **`-mq`** parameter with the **`fteCreateMonitor`** command.

About this task

In this example, the resource to be monitored is the queue *MONITORED_QUEUE*. This queue must be on the monitoring agent's queue manager, *QM_NEPTUNE*. The condition that the queue is monitored for is the presence of a complete group of messages. The task to be performed if the condition is satisfied is defined in the file `task.xml`.

Note: Do not create more than one resource monitor to monitor an individual queue. If you do then unpredictable behavior occurs.

Procedure

Type the following command:

```
fteCreateMonitor -ma AGENT_NEPTUNE -mn myMonitor -mm QM_NEPTUNE -mq MONITORED_QUEUE
  -mt task.xml -tr completeGroups -pi 5 -pu minutes
```

The monitor checks the queue every five minutes to see if the condition `completeGroups` is true. If there are one or more complete groups on the queue, the monitor runs the task defined in the `task.xml` file once for each complete group.

Related concepts

"Customizing MFT resource monitor tasks with variable substitution" on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related tasks

"Configuring MFT monitor tasks to start commands and scripts" on page 235
Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

"Monitoring a queue and using variable substitution" on page 247

You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

Related reference
fteCreateMonitor: create an MFT resource monitor

Customizing MFT resource monitor tasks with variable substitution

When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Attention: Variable names are not case-sensitive.

The variables that are used for substitution are only available for positive trigger conditions. Only `match` and `fileSize` trigger conditions cause variables to be substituted. If a `noMatch` condition is used, and there are substitution variable names in the task definition, the task is not called, and the monitor raises a return code of 110 and error message BFGDM0060E.

If the monitored resource is a queue

The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition.

User-defined message properties are prefixed with `usr.` but do not include this prefix in the variable name. Variable names must be preceded by a dollar sign ($) character and enclosed in braces {}.

For example, `${destFileName}` is replaced with the value of the `usr.destFileName` message property of the first message to be read from the source queue. For more information, see MQ message properties read by MFT from messages on source queues and "Monitoring a queue and using variable substitution" on page 247.

If a variable is not defined as a message property the monitor reports a BFGDM0060E error and returns return code 110 (Monitor task variable substitution failed). In addition to this, the agent writes the following error message to its event log (`outputN.log`):

```
BFGDM0113W: Trigger failure for <monitor name> for reason BFGDM0060E: A monitor task could
not complete as a variable substitution <variable name> was not present.
```

If moderate or verbose resource monitor logging is enabled for the monitor, the monitor writes the following message to the resource monitor event log of the agent (`resmoneventN.log`):

```
BFGDM0060E: A monitor task could not complete as a variable substitution
<variable name> was not present.
```

See "Logging MFT resource monitors" on page 253 for more information on resource monitor logging.

The following table shows which substitution variables are provided by default. For example, `${AGENTNAME}` is replaced with the name of the resource monitor agent.

Table 10. Substitution variables provided by default

Variable	Description
AGENTNAME	The name of the resource monitor agent.
QUEUENAME	The name of the queue that is being monitored.

Table 10. Substitution variables provided by default (continued)	
Variable	**Description**
ENCODING	The character encoding of the first message on the queue or the first message in a group.
MESSAGEID	The IBM MQ message ID of the first message on the queue or the first message in the group.
GROUPID	The IBM MQ group ID of the group or the message ID if only a single message is found. This variable is only set if you are monitoring for complete groups.
CurrentTimeStamp	A time stamp based on the local time that the monitor triggered at. The time stamp value is unique for the agent.
CurrentTimeStamp UTC	A time stamp based on the time, in the UTC time zone, that the monitor triggered at. The time stamp value is unique for the agent.

If the monitored resource is a directory

The following table shows the set of variable names that can be substituted in the task XML definition.

Table 11. Variables that can be substituted	
Variable	**Description**
FilePath	The complete path name of the trigger file.
FileName	The file name part of the trigger.
LastModifiedTime	The time that the trigger file was last modified. This time is expressed as the local time of the time zone that the agent is running in and is formatted as an ISO 8601 time.
LastModifiedDate	The date that the trigger file was last modified. This date is expressed as the local date of the time zone that the agent is running in and is formatted as an ISO 8601 date.
LastModifiedTimeUTC	The time that the trigger file was last modified. This time is expressed as the local time converted to the UTC time zone and is formatted as an ISO 8601 time
LastModifiedDateUTC	The date that the trigger file was last modified. This date is expressed as the local date converted to the UTC time zone and is formatted as an ISO 8601 date.
AgentName	The name of the resource monitor agent.
CurrentTimeStamp	A time stamp that is based on the local time that the monitor triggered at. The time stamp value is unique for the agent.
CurrentTimeStampUTC	A time stamp that is based on the time in the UTC time zone that the monitor triggered at. The time stamp value is unique for the agent.

If the monitored resource is a trigger file

The following table shows the set of variable names that can be substituted when a resource monitor is using the contents of a trigger file to determine the files that need to be transferred.

Table 12. Variables that can be substituted when using a trigger file	
Variable	**Description**
contentSource	The complete path name of the source file.
contentDestination	The complete path name of the destination file.

Variable names must be preceded by a dollar sign ($) character and enclosed in braces, {}. For example, `${FilePath}` is replaced with the fully qualified file path of the matching trigger file.

There are two special keywords that can be applied to variable names to provide further refinement. These are:

token
The token index to substitute (starting at 1 from the left and starting at -1 from the right)

separator
A single character to tokenize the variable value. The default is the forward slash character (/) on AIX and Linux platforms or backward slash character(\) on Windows platforms, but the separator can be any valid character that can appear in the variable value.

If the separator keyword is specified in a variable name, the variable value is split into tokens according to the separator character.

The value that is assigned to the token keyword is used as an index to select which token to use to replace the variable name. The token index is relative to the first character in the variable, and starts at 1. If the token keyword is not specified, the entire variable is inserted.

Any values that are substituted into an agent name in the message XML are treated in a not case-sensitive way. All Managed File Transfer Agent names are uppercase. If the value `Paris` is substituted into an agent attribute in the message XML, this value is interpreted as a reference to the agent `PARIS`.

Related concepts
“Examples: Variable substitution for resource monitor definitions” on page 244
Examples of variable substitution for resource monitor definitions using XML and IBM MQ Explorer.

Related tasks
What to do if variable substitution causes multiple files to go to a single file name

Examples: Variable substitution for resource monitor definitions

Examples of variable substitution for resource monitor definitions using XML and IBM MQ Explorer.

Examples showing how variable substitution works

Assuming that the file path to the matching trigger file is `c:\MONITOR\REPORTS\Paris\Report2009.doc` on Windows and `/MONITOR/REPORTS/Paris/Report2009.doc` on AIX and Linux platforms, the variables are substituted as shown in the following table.

Table 13. How the variables are substituted

Variable specification	After variable substitution
${FilePath}	Windows :`c:\MONITOR\REPORTS\Paris\Report2009.doc` AIX and Linux : `/MONITOR/REPORTS/Paris/Report2009.doc`
${FilePath{token=1}{separator=.}}	Windows :`c:\MONITOR\REPORTS\Paris\Report2009` AIX and Linux : `/MONITOR/REPORTS/Paris/Report2009`
${FilePath{token=2}{separator=.}}	Windows: doc AIX and Linux : doc

Table 13. How the variables are substituted (continued)	
Variable specification	**After variable substitution**
${FilePath{token=3}}	Windows : REPORTS AIX and Linux : Paris

You can also specify a negative token index to select tokens relative to the last character of the variable, as shown in the following table. The examples in the table use the same variable value, `c:\MONITOR\REPORTS\Paris\Report2009.doc` on Windows and `/MONITOR/REPORTS/Paris/Report2009.doc` on AIX and Linux.

Table 14. Examples of using a negative token index	
Variable specification	**After variable substitution**
${FilePath}	Windows :`c:\MONITOR\REPORTS\Paris\Report2009.doc` AIX and Linux : `/MONITOR/REPORTS/Paris/Report2009.doc`
${FilePath{token=-2}{separator=.}}	Windows :`c:\MONITOR\REPORTS\Paris\Report2009` AIX and Linux : `/MONITOR/REPORTS/Paris/Report2009`
${FilePath{token=-2}{separator=\}}	Windows : Paris AIX and Linux : Paris
${FilePath{token=-4}}	Windows : MONITOR AIX and Linux : MONITOR

The variables that are used for substitution are only available for the following positive trigger conditions and the `noSizeChange` option, which is an exception to the positive trigger condition rule:

- `match`
- `fileSize`
- `noSizeChange`

If a `noMatch` condition is used, and there are substitution variable names in the task definition, the task is not called, and the monitor raises a return code of 110 and error message `BFGDM0060E`.

Example using XML

The following example task definition XML uses the monitor agent name as the source agent for the transfer (`Paris`), uses the penultimate directory name in the file path as the destination agent name for the transfer (`Report2009`), and renames the transferred file to be the root of the trigger file name with an extension of `.rpt`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="${AgentName}" QMgr="QM1" />
     <destinationAgent agent="${FilePath{token=-2}}" QMgr="QMD" />
     <transferSet>
```

```
       <item mode="binary" checksumMethod="MD5">
         <source recursive="false" disposition="leave">
           <file>c:/incoming/reports/summary/report.doc</file>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/${FileName{token=1}{separator=.}}.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

This results in the task XML being transformed to:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT1" QMgr="QM1" />
     <destinationAgent agent="Paris" QMgr="QMD" />
     <transferSet>
       <item mode="binary" checksumMethod="MD5">
         <source recursive="false" disposition="leave">
           <file>c:/incoming/reports/summary/report.doc</file>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/Report2009.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

The variable `${FilePath{token=-2}}` in the `<destinationAgent>` element's `agent` attribute is replaced with the value `Paris`. This value is treated in a not case-sensitive way and interpreted as a reference to the agent PARIS.

Examples using IBM MQ Explorer

When creating a resource monitor through the IBM MQ Explorer, and once the monitor properties and trigger conditions have been specified, the option is given to add transfer items to the monitor. The following examples demonstrate how the ${FilePath} and ${FileName} variables can be used in the "**Add a transfer item panel**" to customize transfers resulting from a resource monitor match.

Example 1

In order to simply transfer the source file to another location when a trigger condition is met, the ${FilePath} variable can be used:

- Set the source **File name** to be `${FilePath}`.
- From the dropdown menu of **Type** for the destination, select **Directory**.
- Set the destination **File name** to be the location to which you wish the source file to be transferred, for example, this could be `C:\MFT\out\`.

Example 2

In order to transfer the source file to another location and change the extension of the file, the `${FileName}` variable can be used in conjunction with the `${FilePath}` variable:

In the following example it is assumed that the file path of the source file is equal to `C:\MONITOR\REPORTS\Paris\Report2009.doc`:

- Set the source **File name** to be `${FilePath}`.

- Set the destination **File name** to be the location to which you wish the source file to be transferred, followed by `${FileName{token=1}{separator=.}}`, followed by the new extension of the file. For example, this could be `C:\MFT\out\${FileName{token=1}{separator=.}}.rpt`, which would equate to `C:\MFT\out\Report2009.rpt` with the source file name.

Example 3

In order to use part of the file path of the source file to determine the destination of the transfer, the `${FilePath}` variable can be used in conjunction with token and separator specifications.

In the following example it is assumed that the file path of the source file is equal to `C:\MONITOR\REPORTS\Paris\Report2009.doc`.

It is possible to use part of the source file path to determine the destination of the file. Using the file path example of `C:\MONITOR\REPORTS\Paris\Report2009.doc`, if the file were to be transferred to a folder depending upon the location of the source file, that is, `Paris` in this example, then the following could be done:

- Set the source **File name** to be `${FilePath}`.
- Set the destination **File name** to be the destination to where the folders for each location are situated, and then append the destination part of the file path and the file name. For example, this could be `C:\MFT\out\${FilePath{token=-2}{separator=\}}\${FileName}`, which would equate to `C:\MFT\out\Paris\Report2009.doc` with the source file name.

Related concepts

"Customizing MFT resource monitor tasks with variable substitution" on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related tasks

What to do if variable substitution causes multiple files to go to a single file name

Monitoring a queue and using variable substitution

You can monitor a queue and transfer messages from the monitored queue to a file by using the **`fteCreateMonitor`** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

About this task

In this example, the source agent is called AGENT_VENUS, which connects to QM_VENUS. The queue that AGENT_VENUS monitors is called START_QUEUE and is located on QM_VENUS. The agent polls the queue every 30 minutes.

When a complete group of messages is written to the queue the monitor task sends the group of messages to a file at one of a number of destination agents, all of which connect to the queue manager QM_MARS. The name of the file that the group of messages is transferred to is defined by the IBM MQ message property `usr.fileName` on the first message in the group. The name of the agent that the group of messages is sent to is defined by the IBM MQ message property `usr.toAgent` on the first message in the group. If the `usr.toAgent` header is not set, the default value to be used for the destination agent is AGENT_MAGENTA.

When you specify `useGroups="true"`, if you do not also specify `groupId="${GROUPID}"`, the transfer just takes the first message on the queue. For example, if you are using variable substitution to generate the fileName, it is therefore possible that the contents of a `.txt` will not be correct. This is because the fileName is generated by the monitor, but the transfer actually gets a message that is not the one that should generate the file called fileName.

Procedure

1. Create the task XML that defines the task that the monitor performs when it is triggered.

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_VENUS" QMgr="QM_VENUS" />
     <destinationAgent agent="${toAgent}" QMgr="QM_MARS" />
     <transferSet>
       <item mode="binary" checksumMethod="none">
         <source>
           <queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/${fileName}.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

   The variables that are replaced with the values of IBM MQ message headers are highlighted in **bold**. This task XML is saved to the file `/home/USER1/task.xml`

2. Create a resource monitor to monitor the queue START_QUEUE.

   Submit the following command:

```
fteCreateMonitor -ma AGENT_VENUS -mm QM_VENUS -mq START_QUEUE
                 -mn myMonitor -mt /home/USER1/task.xml
                 -tr completeGroups -pi 30 -pu minutes -dv toAgent=AGENT_MAGENTA
```

3. A user or program writes a group of messages to the queue START_QUEUE.

   The first message in this group has the following IBM MQ message properties set:

```
usr.fileName=larmer
usr.toAgent=AGENT_VIOLET
```

4. The monitor is triggered when the complete group is written. The agent substitutes the IBM MQ message properties into the task XML.

   This results in the task XML being transformed to:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_VENUS" QMgr="QM_VENUS" />
     <destinationAgent agent="AGENT_VIOLET" QMgr="QM_MARS" />
     <transferSet>
       <item mode="binary" checksumMethod="none">
         <source>
           <queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/larmer.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

Results

The transfer that is defined by the task XML is performed. The complete group of messages that are read from the START_QUEUE by AGENT_VENUS is written to a file called `/reports/larmer.rpt` on the system where AGENT_VIOLET is running.

What to do next

Transferring each message to a separate file

If you want to monitor a queue and have every message transferred to a separate file, you can use a similar technique to the one described previously in this topic.

1. Create the monitor as described previously, specifying the **`-tr completeGroups`** parameter on the **`fteCreateMonitor`** command.
2. In the task XML specify the following:

```
<queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
```

However, when you put the messages onto the source queue, do not put them in an IBM MQ group. Add IBM MQ message properties to each message. For example, specify the `usr.filename` property with a unique file name value for each message. This effectively causes the Managed File Transfer Agent to treat each message on the source queue as a separate group.

Related concepts

“Transferring data from messages to files” on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

“Customizing MFT resource monitor tasks with variable substitution” on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

What to do if destination files created by a transfer started by a queue resource monitor contain the wrong data

Related tasks

“Configuring MFT monitor tasks to start commands and scripts” on page 235
Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

“Example: Configuring an MFT resource” on page 241
You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **`-mq`** parameter with the **`fteCreateMonitor`** command.

Related reference

`fteCreateMonitor`: create an MFT resource monitor

MQ message properties read by MFT from messages on source queues

Configuring monitor retry behavior for message-to-file transfers

If a message-to-file transfer that is triggered by a resource monitor fails and leaves the message group that triggered the monitor on the queue, that transfer is resubmitted at subsequent poll intervals. The

number of times that the transfer is resubmitted is limited by the **monitorGroupRetryLimit** property of the monitoring agent.

About this task

Each time a new message-to-file transfer is triggered a new transfer ID is generated for the transfer task.

If the agent is restarted the monitor triggers a transfer again even if the number of times the transfer has been triggered has exceeded the value of **monitorGroupRetryLimit** in the `agent.properties` file. The value of the **monitorGroupRetryLimit** property is the maximum number of times that a monitor triggers a message-to-file transfer again if the message group still exists on the queue. The default value of this property is 10. The value of this property can be set to any positive integer value or -1. If the value -1 is specified for this property, the monitor triggers the transfer again an unlimited number of times, until the trigger condition is not satisfied.

If a transfer attempt causes the number of times that the transfer has been triggered to exceed the value of **monitorGroupRetryLimit**, the agent writes an error to its event log.

A single message is treated as if it was a single group, and the transfer is triggered again at each poll interval while the message remains on the queue and while the number of times the transfer has been triggered is less than the value of **monitorGroupRetryLimit**.

To set the **monitorGroupRetryLimit** property on the monitoring agent, perform the following steps:

Procedure

1. Stop the monitoring agent, using the **fteStopAgent** command.
2. Edit the `agent.properties` file for the monitoring agent to include the following line:

   ```
   monitorGroupRetryLimit=number_of_retries
   ```

   The `agent.properties` file is located in the directory *MQ_DATA_PATH*`/mqft/config/`*coordination_qmgr_name*`/agents/`*monitoring_agent_name*.
3. Start the monitoring agent, using the **fteStartAgent** command.

Related tasks

“Example: Configuring an MFT resource” on page 241
You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **-mq** parameter with the **fteCreateMonitor** command.

Using a trigger file

You can use the contents of a trigger file in a resource monitor to define a set of files to transfer in a single transfer request. Each time a matching trigger file is detected, its contents are parsed for source file paths and optionally for destination file paths. These file paths are then used to define file items in the task transfer XML file that you specify, which is submitted as single transfer request to the agent. The definition of the resource monitor determines whether trigger content is enabled.

You can enable file content triggering when you create a monitor by specifying the **-tc** (trigger content) parameter. This **-tc** parameter applies only to the file trigger options `match` and `noSizeChange`. For more information about creating a monitor, see **fteCreateMonitor**: create an MFT resource monitor.

When using a trigger content file, the default format of each line is either:

- A single source file path, or
- A source file path and a destination file path, separated by a comma

where white space characters are handled as part of the file paths.

It is possible to change the default line format by specifying the **-tcr** and **-tcc** parameters on the **fteCreateMonitor** command. For more information, see “Advanced options” on page 251.

After a trigger file is parsed, a list of file paths is generated and applied to the transfer task XML that you specified. As with all monitors, the format of the transfer task XML is a complete transfer task XML generated by the **fteCreateTransfer** command with a single item or file defined. The single item must use the substitution variables ${contentSource}, and optionally ${contentDestination}, as replacements for the source and destination file paths. The monitor expands the transfer task XML to include a file item for each line (file path) in the trigger file.

You cannot use file content triggering with the **-bs** parameter because the **-tc** parameter implies one transfer request for each trigger file.

Example

The following example defines a monitor to trigger on a file that ends in `trig` and reads the file paths in that file.

```
fteCreateTransfer -gt task.xml -sa SrcAgent -da DestAgent -dd /file/destdir ${contentSource}
fteCreateMonitor -mn TrigMonitor -md /home/trigdir -mt task.xml -ma SrcAgent -tr "match,*.trig"
-tc
```

The **fteCreateTransfer** command creates a file that is called `task.xml` for a single file with a source file path of ${contentSource}. For example:

```
<item checksumMethod="MD5" mode="binary">
    <source disposition="leave" recursive="false">
          <file>${contentSource}</file>
    </source>
</item>
```

The **fteCreateMonitor** command scans for files that end in `trig` in the `/home/trigdir` directory and uses the contents to create a single transfer request that is based on the `task.xml` for all paths in that trigger file. The format of the trigger file must be one file path (source only) on each line with no comma separator. For example:

```
/home/file/first.txt
/home/file/second.txt
/home/different/third.txt
⋮
```

All files are delivered to the `/file/destdir` directory with its file name and not its file path, that is, `/home/file/first.txt` is delivered to `/file/destdir/first.txt`.

Alternatively, if you change the **-dd /file/destdir** parameter in the **fteCreateTransfer** command to `-df ${contentDestination}` and the format of the content of a trigger file to *source file path*,*destination file path*, you can define different destination paths for the same destination agent. For example:

```
/home/file/first.txt,/home/other/sixth.txt
```

The destination location then becomes `/home/other/sixth.txt`.

The substitution variables can be tokenized. For example, you can separate the file name part from the provided path using ${contentDestination{token=-1}}. Therefore, if the **fteCreateTransfer** destination is defined as `-df /file/destdir/${contentDestination{token=-1}}`, the new destination for `/home/file/first.txt` is `/file/destdir/sixth.txt`.

Advanced options

You can change the default line format for the content of the trigger file by using the **-tcr** *regex* parameter. Supply a regular expression that matches the required line format and supplies either one or two capture groups.

For more information about regular expressions, see Regular expressions used by MFT.

The first capture group is the source and the second, optional, capture group is the destination. For example:

- The source and destination path are separated by a hyphen:

```
((?:[^-])+)-((?:[^-])+)
```

  In this example, the separator is defined in three locations and all three instances of the hyphen, -, can be changed to any character. Ensure that you escape any special characters.

- The source and destination paths are separated by a comma with trailing spaces. Comments that are indicated by a number sign (#) are ignored.

```
((?:[^,])+),((?:[^,])+) *(?:#.*)+
```

  File paths cannot contain the number sign (#). Typically an entry is as follows: `/home/source/from.txt,/home/destination/to.txt # some comment`.

If you use the **-tcr** parameter, ensure that the regular expression is well designed and tested so that the expression can detect errors and correctly parse the trigger files.

You can reverse the order of the capture by using the **-tcc destSrc** parameter. If you specify this parameter, the first capture group is the destination file path and the second group is the source file path.

How errors are handled

Empty trigger file
If the trigger file is empty, the outcome is no file transfer. That is, the monitor creates a transfer request but no file items are specified.

Trigger file with errors
If any entry in a trigger file fails to parse against the expected format, no transfer request is generated. A monitor error log is published and the error is also logged in the event log. The trigger file is marked as processed and the monitor does not attempt to process the file again until the file has been updated.

Mismatching transfer task XML
The transfer task XML must match the trigger file, that is if the transfer task XML has both ${contentSource} and ${contentDestination}, all trigger files for that monitor must have source and destination file paths and similarly for the reverse. In the first case the monitor reports a substitution failure of the ${contentDestination} if the trigger file supplies the source file path only.

Examples

The following example is a basic content trigger where the contents of a trigger file has a source file path only:

```
fteCreateTransfer -gt task.xml -sa SrcAgent -da DestAgent -dd /file/destdir ${contentSource}
fteCreateMonitor -mn TrigMonitor -md /home/trigdir -mt task.xml -ma SrcAgent -tr "match,*.trig"
-tc
```

The **-tcr** parameter defines two capture groups of a sequence of any characters that are separated by a space character. The **-tcc destSrc** parameter and option indicate that the capture groups are to be processed as destination then source.

```
fteCreateTransfer -gt task.xml -sa SrcAgent -da DestAgent -df ${contentDestination} $
{contentSource}
fteCreateMonitor -mn TrigMonitor -md /home/trigdir -mt task.xml -ma SrcAgent -tr "match,*.trig"
-tc
    -tcr "((?:[^ ])+) ((?:[^ ])+)" -tcc destSrc
```

Related concepts
“Customizing MFT resource monitor tasks with variable substitution” on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied,

and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

Related tasks

“Monitoring a queue and using variable substitution” on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

Related reference

fteCreateMonitor: create an MFT resource monitor

fteCreateTransfer: start a new file transfer

Logging MFT resource monitors

You can obtain diagnostic information about resource monitors by using logging.

About this task

You can use logging for resource monitors by using either the **fteSetAgentLoglevel** command or the `agent.properties` file to control resource monitor logging.

Note that existing trace points are still used for capturing information.

The resource monitor logs are written to a file named `resmonevent`*N*`.log`, where *N* stands for a number; for example, `resmonevent0.log`. The event log files record several actions that take place when a monitor polls a resource, for example, a directory or a queue.

Attention: All resource monitors of an agent write to the same log file.

For some example output of a `resmonevent`*N*`.log` file, see What to do if your MFT directory resource monitor is not triggering files.

The following table lists the type of events the resource monitor writes to the log file. The third column describes the log level needed to capture each event where the lowest level is INFO and highest is VERBOSE.

Note that setting a higher log level, also writes lower level events. For example, setting log level to MODERATE also writes INFO level events, but not VERBOSE level events.

Number	Event	Log level	Description
1	Monitor Created	INFO	A resource monitor has been created.
2	Monitor Deleted	INFO	A resource monitor has been deleted.
3	Monitor Stopped	INFO	A resource monitor has been stopped.
4	Monitor Started	INFO	A resource monitor has been started.
5	Start Poll	INFO	A resource monitor started a new poll cycle.
6	End Poll	INFO	A resource monitor poll cycle has ended.
7	Pattern Match	VERBOSE	A file on the trigger monitor directory, or a message in a queue that matches the specified pattern, has been found.
8	Pattern Mismatch	VERBOSE	An unmatched file on the trigger monitor directory, or a message in a queue that does not match the specified pattern, has been found.

Number	Event	Log level	Description
9	Transfer Request	INFO	A transfer request initiated or submitted by the resource monitor.
10	Items Matched	INFO	Total number of trigger files found in the directory polled by a resource monitor.
11	Transfer Items	INFO	Total number of items in the transfer request.
12	FDC Generated	MODERATE	A resource monitor has generated an exception.
13	Monitor Start Failed	MODERATE	A resource monitor failed to start.
14	History Cleared	INFO	Monitor history information has been cleared.
15	Clear Monitor History Failed	INFO	Attempt to clear monitor history information has failed.
16	Transfer ID	INFO	ID of the transfer request has been submitted by monitor.
17	Batching	INFO	Total number of transfer requests for matched items: *N*, where *N* is a number.
18	Connected	VERBOSE	A resource monitor has connected to the agent queue manager.
19	Disconnected	VERBOSE	A resource monitor has disconnected from the agent queue manager.
20	Error during disconnect	VERBOSE	A resource monitor encountered an issue when disconnecting from the agent queue manager.
21	Pattern No Match	MODERATE	No files in the trigger monitor directory or messages in a queue that match the specified pattern have been found.
22	Transfer Metadata	MODERATE	The metadata associated with the triggered resource including file or queue message properties.
23	Exception	MODERATE	An internal error occurred during the monitor operation.
24	Maximum Items	MODERATE	The resource monitor has reached a maximum number of matched resources.
25	Monitor Create Failed	MODERATE	A resource monitor failed to be created.

Procedure

- To use the **fteSetAgentLoglevel** command to turn resource monitor logging on and off, see fteSetAgentLogLevel for a description of the **logMonitor** parameter, and examples of how you use the different options.
- To use the `agent.properties` file to control resource monitor logging, see The MFT `agent.properties` file for a description of the additional properties that allow you to carry out the following logging activities:
  - Turn logging on or off
  - Limit the size of each log file

  - Limit the number of logs that resource monitors can generate
- Set the default logging level for all resource monitors by adding the **resourceMonitorLog** property to the `agent.properties` file. You can use one of the following values:
  - Info
  - Moderate
  - Verbose
  - Off

  For more information, see fteSetAgentLogLevel (Turn on or turn off logging to file of certain MFT agent operations).
- Restart the agent to apply the change.

Example

The following sample message sets `verbose` level logging for agent HA2, on queue manager MFTDEMO:

```
<?xml version="1.0"?>
<log:log version="6.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xmlns:log="https://www.ibm.com/log">
    <log:originator>
      <log:request>
        <log:hostName>192.168.7.1</log:hostName>
        <log:userID>johndoe</log:userID>
      </log:request>
    </log:originator>
    <log:endpoint agent="HA2" QMgr="MFTDEMO"/>
    <log:logMonitor>MON1="verbose"</log:logMonitor>
</log:log>
```

Related reference

fteSetAgentLogLevel command

The MFT `agent.properties` file

Starting an MFT resource monitor

From IBM MQ 9.3.0, you can start resource monitors without needing to stop or restart an agent by using the **fteStartMonitor** command.

Before you begin

If user authority management has been enabled by setting the **authorityChecking** attribute to `true` in the `agent.properties` file, you must have either `Monitor` or `Monitor operations` authority to start a resource monitor. For more information about user authority management, see Restricting user authorities on MFT agent actions.

About this task

You can run the **fteStartMonitor** command from any system where the Managed File Transfer commands component is installed, which means that you can start a resource monitor from anywhere, and are not restricted to the system where the agent that owns the resource monitor is running. For information about the required and optional parameters for this command, see fteStartMonitor (start an MFT resource monitor).

Procedure

- To find out the state of an agent before or after running the **fteStartMonitor** command, use the **fteListMonitors** command with the **-v** parameter as shown in the following example:

```
fteListMonitors -ma monitoring_agent_name -v
```

- To start a resource monitor in an agent running on the same machine, enter the **fteStartMonitor** command as follows:

  ```
  fteStartMonitor -mn monitor_name -ma agent_name
  ```

- To start a resource monitor in an agent running on a different machine, enter the **fteStartMonitor** command as follows:

  ```
  fteStartMonitor -mn monitor_name -ma agent_name -mm AgentQueueManager
  ```

  If the command queue manager is also the agent queue manager for the monitoring agent, then the **-mm** parameter is optional, otherwise you must specify the agent queue manager with the **-mm** parameter.

Results

If the agent is running, then the resource monitor is started if it is currently stopped. The command outputs the following messages and logs an event in the agent's `output0.log`.

```
BFGCL0816I: A request to start resource monitor 'monitor_name' of agent 'agent_name' has been
issued.
BFGCL0251I: The request has successfully completed.
```

For information about the messages that the command outputs if it cannot start the resource monitor, see fteStartMonitor (start an MFT resource monitor).

Related concepts

“MFT resource monitoring concepts” on page 231
An overview of the key concepts of the Managed File Transfer resource monitoring feature.

Related tasks

“Stopping an MFT resource monitor” on page 256
From IBM MQ 9.3.0, you can stop resource monitors without needing to stop or restart an agent by using the **fteStopMonitor** command.

Related reference

fteStartMonitor (start an MFT resource monitor)

Stopping an MFT resource monitor

From IBM MQ 9.3.0, you can stop resource monitors without needing to stop or restart an agent by using the **fteStopMonitor** command.

Before you begin

If user authority management has been enabled by setting the **authorityChecking** attribute to `true` in the `agent.properties` file, you must have either `Monitor` or `Monitor operations` authority to stop a resource monitor. For more information about user authority management, see Restricting user authorities on MFT agent actions.

About this task

You can run the **fteStopMonitor** command from any system where the Managed File Transfer commands component is installed, which means that you can stop a resource monitor from anywhere, and are not restricted to the system where the agent that owns the resource monitor is running. For information about the required and optional parameters for this command, see fteStopMonitor (stop an MFT resource monitor).

When a resource monitor is stopped, it writes a message to the agent's resource monitor event log, `resmoneventnumber.log`. If the resource monitor is stopped with the **fteStopMonitor** command, the message includes the name of the user who issued the stop request:

```
Resource Monitor Stopped by user '<mquser_id>'
```

A resource monitor is automatically started if its agent restarts, even if the resource monitor was previously stopped by using the **fteStopMonitor** command.

Agents process stop monitor requests serially rather than in parallel, so, for example, if an agent receives a request to stop monitor M1 and then another request to stop monitor M2 in quick succession, it stops M1 first before trying to stop M2.

Procedure

- To find out the state of an agent before or after running the **fteStopMonitor** command, use the **fteListMonitors** command with the **-v** parameter as shown in the following example:

  ```
  fteListMonitors -ma monitoring_agent_name -v
  ```

- To stop a resource monitor in an agent running on the same machine, enter the **fteStopMonitor** command as follows:

  ```
  fteStopMonitor -mn monitor_name -ma agent_name
  ```

- To stop a resource monitor in an agent running on a different machine, enter the **fteStopMonitor** command as follows:

  ```
  fteStopMonitor -mn monitor_name -ma agent_name -mm AgentQueueManager
  ```

  If the command queue manager is also the agent queue manager for the monitoring agent, then the **-mm** parameter is optional, otherwise you must specify the agent queue manager with the **-mm** parameter.

Results

If the agent is running, then the resource monitor is stopped if it is currently started. The command outputs the following messages and logs an event in the agent's `output0.log`.

```
BFGCL0813I: A request to stop resource monitor 'MNTR' of agent 'SOURCE' has been issued.
BFGCL0251I: The request has successfully completed.
```

For information about the messages that the command outputs if it cannot stop the resource monitor, see fteStopMonitor (stop an MFT resource monitor)

Related concepts

“MFT resource monitoring concepts” on page 231
An overview of the key concepts of the Managed File Transfer resource monitoring feature.

Related tasks

“Starting an MFT resource monitor” on page 255
From IBM MQ 9.3.0, you can start resource monitors without needing to stop or restart an agent by using the **fteStartMonitor** command.

Related reference

fteStopMonitor (stop an MFT resource monitor)

Backing up and restoring MFT resource monitors

You can back up the resource monitors that you want to have available for future use by exporting their definitions to an XML file that you can then import to create a new resource monitor from the backup.

About this task

You might need to back up the resource monitors that you have previously defined so that you can re-use their definitions in the future, for example to re-create the resource monitors in a different infrastructure or if a resource monitor needs to be re-created because of queue manager issues.

You can back up a single resource manager definition by using either the **fteCreateMonitor** command or the **fteListMonitors** command with the **-ox** parameter. In both cases, the resource manager definition is backed up by exporting it to an XML file. You can then use the **-ix** parameter of the **fteCreateMonitor** command to create a new resource manager by importing the definition from the XML file.

With the **-ox** parameter, you can back up only one resource monitor definition at a time.

The **-od** parameter is added to the **fteListMonitors** command. By specifying this parameter, you can back up more than one resource monitor at a time by exporting their definitions in bulk to a specified directory. Each resource monitor definition is saved to a separate XML file with a name in the format *agent name*.*monitor name*.xml.

The **-od** parameter is particularly useful if you have a large number of resource monitors that you want to back up because you need to run the **fteListMonitors -od** command once only, instead of having to run the **fteListMonitors -ox** command separately for each resource definition, or use a separate script to run the **fteListMonitors -ox** command for each resource monitor.

Procedure

- To back up the definition of one resource monitor by exporting it to an XML file, use either of the following commands:
  - The **fteCreateMonitor** command with the **-ox** parameter.
  - The **fteListMonitors** command with the **-ox** parameter.

  When you are using the **-ox** parameter, you must also specify the **-ma** and **-mn** parameters, as shown in the following example:

  ```
  fteListMonitors -ma AGENT1 -mn MONITOR1 -ox filename1.xml
  ```

- To back up multiple resource monitor definitions by exporting them to XML files in a specified directory, use the **fteListMonitors** command with the **-od** parameter as shown in the following example:

  ```
  fteListMonitors -od /usr/mft/resmonbackup
  ```

  You must specify a valid target directory when you are backing up resource monitors in bulk. Not specifying a target path results in an error message as shown in the following example:

  ```
  BFGCL0762E: Output directory not specified. Rerun the command specifying a valid path.
  ```

  The **-od** parameter must not be combined with the **-ox** parameter, otherwise the following error message is displayed:

  ```
  BFGCL0761E: It is not valid to specify both the '-od' and '-ox' parameters together.
  ```

  You can define a particular set of resource monitors to include in the backup. For instance, by using the **-ma** parameter to specify the name of an agent, you can back up all the resource monitors for that agent, as shown in the following example:

  ```
  fteListMonitors -ma AGENT1 -od /usr/mft/resmonbackup
  ```

  You can also use wildcard matching by including an asterisk character (*) character when defining a pattern to use for matching agent names or monitor names, or both. The following example backs up all the resource monitors that have names matching a specified pattern and that are in an agent with a name matching a specified pattern:

  ```
  fteListMonitors -ma AGENT* -mn MON* -od /usr/mft/resmonbackup
  ```

  While the command is running, it displays the following progress report messages:

  ```
  A total of number matching resource monitor definitions found.
  index of number resource monitor definitions saved to file system.
  ```

  If you are using the verbose option, the running total is still displayed, but instead of displaying

index of *number* resource monitor definitions saved to file system

the command displays the name of the monitor definition being saved, for example:

```
BFGCL0762I: Definition of monitor 'FILEMON' of agent 'XFERAGENT' saved as
FILEMON.XFERAGENT.XML to file system.
```

- To back up one resource monitor for a particular agent by exporting it to an XML file in a specified directory, use the **fteListMonitors** command with the **-od** parameter:

  ```
  fteListMonitors -ma AGENT1 -mn MONITOR1 -od /usr/mft/resmonbackup
  ```

  Using the **-od** parameter to back up a single resource monitor is similar to using the **-ox** parameter, except that the output file name is in the format *agent name*.*monitor name*.xml.
- To restore resource monitor definitions from a backup, use the **fteCreateMonitor** command with the **-ix** parameter as shown in the following example:

  ```
  fteCreateMonitor -ix file name
  ```

  For more examples of how to use the **-od** parameter, see fteListMonitors: list MFT resource monitors.

Related reference
fteCreateMonitor: create an MFT resource monitor
fteListMonitors: list MFT resource monitors

Clearing resource monitor history

You can clear the history of a resource monitor so that another file transfer request can be submitted for a file that was not transferred earlier due to a failure. To clear the resource monitor history, you can use either the **fteClearMonitorHistory** command or the IBM MQ Explorer.

Before you begin

If user authority management has been enabled by setting the **authorityChecking** attribute to true in the agent.properties file, the user who clears the monitor history must have the appropriate authority as shown in the following table.

*Table 15. User authority required to run the **fteClearMonitorHistory** command*

User clearing the monitor history	MFT access authority	Required authority
The same user as the one that created the resource monitor.	Monitor	BROWSE on SYSTEM.FTE.AUTHMON1.<*monitor_agent_name*> This is the same authority as the authority required to create or delete the resource monitor.
Any user other than the user that created resource monitor.	Monitor operations	SET on SYSTEM.FTE.AUTHPS1.<*agent_name*> This is the same authority as the authority required to delete the resource monitor.

For more information about user authority management, see Restricting user authorities on MFT agent actions.

If a user without the required authority attempts to clear the resource monitor history, the **fteClearMonitorHistory** command outputs an error message and logs the failure in the agent's output0.log file. For more information, see fteClearMonitorHistory: clear resource monitor history.

About this task

If a file transfer has been initiated and a file cannot be transferred for any reason, the resource monitor does not select this file for transfer again in its next poll because the monitor history indicates that the file was seen in an earlier poll and it has not been modified since then (see “MFT resource monitoring concepts” on page 231).

Before IBM MQ 9.1.3, if a file fails to transfer, the file transfer can only be initiated again either if the file is deleted and then placed in the directory again, or if the file is updated so that the last modified date attribute is changed, or if the resource monitor itself is recreated.

However, you can clear the resource monitor history either by using the **fteClearMonitorHistory** command or by using the IBM MQ Explorer. Clearing the history allows another transfer request for a file that has failed to transfer to be submitted without the need to either delete file and then place in the directory again, or update the file to change its last modified date attribute, which is useful, for example, in situations where there is need for the file to be transferred but modifying the file is not possible. Being able to clear the history of a resource monitor also means that there is no need to recreate the resource monitor in order to submit another transfer request for a file that has failed to transfer.

z/OS The sample SCSQFCMD member shipped with Managed File Transfer on z/OS includes a JCL script to clear a monitor's history.

Procedure

- To use the **fteClearMonitorHistory** command to clear the resource monitor history, enter the command in the following format:

  ```
  fteClearMonitorHistory -p <configuration> -ma <agent name> -mn <monitor name> -w 1000
  ```

  Only the **-ma** and **-mn** parameters are required. All other parameters are optional. For more information about how to use the **fteClearMonitorHistory** command, including examples, see fteClearMonitorHistory: clear resource monitor history.

  If the history is cleared successfully, the command outputs the following message:

  ```
  BFGCL0780I: A request to clear history of resource monitor 'monitor name' of agent 'agent
  name' has been issued.
  BFGCL0251I: The request has successfully completed.
  ```

  and logs the success in the agent's `output0.log` file.

  If the attempt to clear the resource monitor history fails, the **fteClearMonitorHistory** outputs an error message and logs the failure in the agent's `output0.log` file.

- To use the resource monitor view in the IBM MQ Explorer MFT Plug-in to clear the resource monitor history, right-click the resource monitor and select **Clear History** from the drop-down menu.

  If the history is cleared successfully the following message is displayed:

  ```
  BFGUI00171: Resource monitor history cleared successfully.
  ```

  If the attempt to clear the history fails, an error message is displayed. For example:

  ```
  BFGUI0016E Failed to clear history of specified resource monitor - reason 2059
  ```

Working with file transfer templates

You can use file transfer templates to store common file transfer settings for repeated or complex transfers. Either create a transfer template from the command line by using the **fteCreateTemplate** command or use the IBM MQ Explorer to create a transfer template by using the **Create New Template for Managed File Transfer** wizard, or save a template while you are creating a file transfer by selecting the **Save transfer settings as a template** check box. The **Transfer Templates** window displays all of the transfer templates that you have created in your Managed File Transfer network.

About this task

To create a transfer template from the command line, use the fteCreateTemplate command. Then when you want to submit a transfer template that you created on the command line, click **Submit** in IBM MQ Explorer.

To view transfer templates in the IBM MQ Explorer, use the following steps:

Procedure

1. Expand **Managed File Transfer** in the Navigator view. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are listed in the Navigator view. Expand the name of the coordination queue manager that you have used for the scheduled transfer. If you want to change which coordination queue manager you are connected to, right-click the name of the coordination queue manager you want to use in Navigator view and click **Connect**.
3. Click **Transfer Templates**. The **Transfer Templates** window is displayed in the Content view.
4. The **Transfer Templates** window lists the following details about your file transfers:
   a) **Name** The name of your file transfer template.
   b) **Source** The name of the agent used to transfer the file from the source system.
   c) **Source File** The name of the file to be transferred on its host system.
      Expand the transfer template information to view this field.
   d) **Destination** The name of the agent used to receive the file at the destination system.
   e) **Destination File** The name of the file after it is transferred to the destination system.
      Expand t the transfer template information to view this field.
   f) **Scheduled Start (selected time zone)** The time and date that the file transfer is scheduled to start in the time zone used by the administrator. To change the time zone displayed, click **Window** > **Preferences** > **IBM MQ Explorer** > **Managed File Transfer** and select an alternative time zone from the **Time zone:** list. Click **OK**.
   g) **Trigger Events** The type of event that triggers the file transfer to start. The type can be one of the following values: `exists`, `does not exist`, or `exceeds`.

Results

To refresh what is displayed in the **Transfer Templates** window, click the Refresh button on the Content view toolbar.

To submit a transfer template and start the transfer defined in the template, right-click the template name and click **Submit**.

To change a transfer template, right-click the template name and click **Edit**. All files included in the original template are listed as part of a transfer group, even if they were not included as part of a group in the original template. If you want to remove a file from the template you must select the file specification from the group and click **Remove selected**. If you want to add new file specifications to the template use the fields in the template panel and click the **Add to group** button. When you have made your edits, you are prompted to give the edited template a new name.

To create a file transfer from a transfer template, right-click the template name and click **Edit as New Transfer**.

To create a duplicate copy of a transfer template, right-click the template name and click **Duplicate**. The duplicate transfer template is automatically saved with the same name as the original template, appended with "(copy)".

To delete a transfer template, right-click the template name and click **Delete**.

Related tasks

"Creating a file transfer template using IBM MQ Explorer" on page 262
You can create a file transfer template from IBM MQ Explorer or from the command line. You can then use that template to create new file transfers using the template details or submit the template to start the file transfer.

Related reference

fteCreateTemplate: create new file transfer template

fteListTemplates

fteDeleteTemplates

Creating a file transfer template using IBM MQ Explorer

You can create a file transfer template from IBM MQ Explorer or from the command line. You can then use that template to create new file transfers using the template details or submit the template to start the file transfer.

About this task

To create a file transfer template from the command line, use the fteCreateTemplate command.

To create a file transfer template using the **Create New Template for Managed File Transfer** wizard in IBM MQ Explorer, use the following steps:

Procedure

1. In the Navigator view, click **Managed File Transfer**. **Managed File Transfer Central** is displayed in the Content view.
2. All of your coordination queue managers are displayed in the Navigator view. Expand the name of the coordination queue manager that you have used for the scheduled transfer. If you want to change which coordination queue manager you are connected to, right-click the name of the coordination queue manager you want to use in Navigator view and click **Connect**.
3. Start the **Create New Template for Managed File Transfer** wizard by right-clicking **Transfer Templates** and then clicking **New Template**.
4. Follow the instructions on the wizard panels. There is context-sensitive help provided for each panel. To access the context-sensitive help on Windows, press F1. On Linux, press `Ctrl+F1` or `Shift+F1`.

   If you have created a template that contains all the required transfer details, ensure that you select the **Save transfer settings as a template** check box on the **Transfer Summary** page if this check box is not already selected. Also enter a name for the template in the Name field. If you create a template that does not yet contain all of the required transfer details, the **Save transfer settings as a template** check box is automatically checked for you.

Related tasks

"Working with file transfer templates" on page 260
You can use file transfer templates to store common file transfer settings for repeated or complex transfers. Either create a transfer template from the command line by using the **fteCreateTemplate** command or use the IBM MQ Explorer to create a transfer template by using the **Create New Template for Managed File Transfer** wizard, or save a template while you are creating a file transfer by selecting the **Save transfer settings as a template** check box. The **Transfer Templates** window displays all of the transfer templates that you have created in your Managed File Transfer network.

Related reference

fteCreateTemplate: create new file transfer template

fteListTemplates

fteDeleteTemplates

Backing up a file transfer template definition

File transfer templates contain an XML document that defines the source and destination file specifications for the transfer. You can use this XML file as input to the **fteCreateTemplate** command to recreate a file transfer template.

About this task

To back up the XML document containing the source and destination file specifications for a transfer template, use either the fteCreateTransfer command command or IBM MQ Explorer. To create a transfer template XML-formatted backup file, use the following steps:

Procedure

- Method one: use the **-gt** parameter on an fteCreateTransfer command to generate a transfer template XML message to a new file.
- Method two: create the template using IBM MQ Explorer.

  When you get to the *Transfer template summary* page:

  a) Copy the *Request message XML preview*.

  b) Save this transfer template XML message to a new file.
- Method three: use IBM MQ Explorer to backup up existing templates.

  a) Go to **Managed File Transfer** > **Queue Manger Name** > **Transfer Templates**.

  b) In the Transfer panel, highlight the template that needs to be backed up, right click and select **Edit** from the pop up menu.

  c) Click **Next** until you get to the *Transfer template summary* page.

  d) Copy the *Request message XML preview*.

  e) Save this transfer template XML message to a new file.

Results

You can use the transfer template XML message file, created by one of the above methods, as input to the fteCreateTemplate command. Refer to the **fteCreateTemplate** command for details of how you use this command.

Related reference

fteCreateTemplate command

fteListtTmplates command

Transferring data from files to messages

You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

For information about message-to-file transfers, see “Transferring data from messages to files” on page 278.

The destination agent for a file-to-message transfer cannot be a protocol bridge agent or a Connect:Direct bridge agent.

You can transfer file data to IBM MQ message data. The IBM MQ messages can be read and used by applications. The following types of file-to-message transfer are supported:

- From a single file to a single message. The message does not have an IBM MQ group ID set.
- From a single file to multiple messages, by splitting the file into messages of a given length. The messages all have the same IBM MQ group ID.

- From a single file to multiple messages, by splitting a text file at a Java regular expression delimiter. The messages all have the same IBM MQ group ID.
- From a single file to multiple messages, by splitting a binary file at a hexadecimal delimiter. The messages all have the same IBM MQ group ID.

  If you want to split a binary file using a sequence of bytes as the delimiter, use the **-sqdb** parameter of the **fteCreateTransfer** command. For more information, see -sqdb parameter.

By default the messages created by a file-to-message transfer are persistent. The messages can be set to be non-persistent or to have the persistence value defined by the destination queue.

If you specify that a file is split into multiple messages, all messages created from the file have the same IBM MQ group ID. If you do not specify that a file is split into multiple messages, only one message is created from the file and this message does not have the IBM MQ group ID set.

If you are transferring files to large messages, or many small messages, you might need to change some IBM MQ or Managed File Transfer properties. For information about, see Guidance for setting MQ attributes and MFT properties associated with message size.

Note: If the destination queue is either a clustered queue, or an alias to a clustered queue, you will get an error message when transferring a file into a queue if the agent property enableClusterQueueInputOutput has not been set to true. For more information see What to do if the destination queue is a clustered queue, or an alias to a clustered queue

Related tasks

"Configuring an agent to perform file-to-message transfers" on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

"Example: Transferring a single file to a single message" on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

"Example: Splitting a single file into multiple messages by length" on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

"Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages" on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

"Example: Splitting a text file into multiple messages using a regular expression delimiter" on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

"Example: Setting IBM MQ message properties on a file-to-message transfer" on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Example: Setting user-defined properties on a file-to-message transfer" on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages

to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

"Failure of a file-to-message transfer" on page 277
If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

MQ message properties set by MFT on messages written to destination queues
Guidance for setting MQ attributes and MFT properties associated with message size

Configuring an agent to perform file-to-message transfers

By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

About this task

If you attempt to perform a file-to-message transfer to a destination agent that does not have the `enableQueueInputOutput` property set to true, the transfer fails. The transfer log message that is published to the coordination queue manager contains the following message:

```
BFGIO0197E: An attempt to write to a queue was rejected by the destination agent. The
agent must have enableQueueInputOutput=true set in the agent.properties file to
support transferring to a queue.
```

To enable the agent to write to and read from queues perform the following steps:

Procedure

1. Stop the destination agent using the **fteStopAgent** command.
2. Edit the `agent.properties` file to include the line `enableQueueInputOutput=true`.

   The `agent.properties` file is located in the directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*destination_agent_name*.
3. Optional: Edit the `agent.properties` file to include the line `enableClusterQueueInputOutput=true`. The `agent.properties` file is located in the directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*destination_agent_name*.
4. Start the destination agent using the **fteStartAgent** command.

Related concepts

"Transferring data from files to messages" on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

"Example: Transferring a single file to a single message" on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

"Example: Splitting a single file into multiple messages by length" on page 268

You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

"Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages" on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

"Example: Splitting a text file into multiple messages using a regular expression delimiter" on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

"Example: Setting IBM MQ message properties on a file-to-message transfer" on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Example: Setting user-defined properties on a file-to-message transfer" on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

Related reference
fteStopAgent
fteStartAgent
The MFT `agent.properties` file
"Failure of a file-to-message transfer" on page 277
If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

Example: Transferring a single file to a single message

You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

About this task

The source file is called `/tmp/single_record.txt` and is located on the same system as the source agent, AGENT_NEPTUNE. The source agent, AGENT_NEPTUNE, uses the queue manager QM_NEPTUNE. The destination agent is AGENT_VENUS and this agent connects to the queue manager QM_VENUS. The destination queue, RECEIVING_QUEUE, is located on the queue manager QM_MERCURY. QM_MERCURY is in the same IBM MQ network as, and can be accessed by, the queue manager QM_VENUS.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_VENUS -dm QM_VENUS
                  -dq RECEIVING_QUEUE@QM_MERCURY /tmp/single_record.txt
```

If the destination queue is on a different queue manager to the queue manager used by the destination agent you must specify the value of the **-dq** parameter in the following format *queue_name*@*queue_manager_name*. If you do not specify @*queue_manager_name* in the value, the

destination agent assumes that the destination queue is located on the destination agent queue manager. The exception is when the enableClusterQueueInputOutput agent property has been set to true. In this case the destination agent will use standard IBM MQ resolution procedures to determine where the queue is located.

The source agent, AGENT_NEPTUNE, reads the data from the file `/tmp/single_record.txt` and transfers this data to the destination agent, AGENT_VENUS. The destination agent, AGENT_VENUS, sends the data to a persistent message on the queue RECEIVING_QUEUE@QM_MERCURY. The message does not have an IBM MQ group ID set.

Related concepts

"Transferring data from files to messages" on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

"Configuring an agent to perform file-to-message transfers" on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

"Example: Splitting a single file into multiple messages by length" on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

"Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages" on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

"Example: Splitting a text file into multiple messages using a regular expression delimiter" on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

"Example: Setting IBM MQ message properties on a file-to-message transfer" on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Example: Setting user-defined properties on a file-to-message transfer" on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

"Failure of a file-to-message transfer" on page 277

If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

Example: Splitting a single file into multiple messages by length

You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

About this task

The source file is called `/tmp/source.file` and is 36 KB in size. The source file is located on the same system as the source agent AGENT_NEPTUNE. The source agent, AGENT_NEPTUNE, connects to the queue manager QM_NEPTUNE. The destination agent is AGENT_MERCURY, which connects to the queue manager QM_MERCURY. The destination queue, RECEIVING_QUEUE, is also located on the queue manager QM_MERCURY. The transfer splits the source file into sections that are 1 KB in size and writes each of these sections to a message on RECEIVING_QUEUE.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_MERCURY -dm QM_MERCURY
                  -dq RECEIVING_QUEUE -qs 1K /tmp/source.file
```

The source agent, AGENT_NEPTUNE, reads the data from the file `/tmp/source.file` and transfers this data to the destination agent, AGENT_MERCURY. The destination agent, AGENT_MERCURY, writes the data to thirty-six 1 KB persistent messages on the queue RECEIVING_QUEUE@QM_MERCURY. These messages all have the same IBM MQ group ID and the last message in the group has the IBM MQ LAST_MSG_IN_GROUP flag set.

Related concepts

“Transferring data from files to messages” on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

“Configuring an agent to perform file-to-message transfers” on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

“Example: Transferring a single file to a single message” on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

“Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages” on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

“Example: Splitting a text file into multiple messages using a regular expression delimiter” on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

“Example: Setting IBM MQ message properties on a file-to-message transfer” on page 273

You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Example: Setting user-defined properties on a file-to-message transfer" on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

"Failure of a file-to-message transfer" on page 277
If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

Example: Splitting a text file into multiple messages using a regular expression delimiter

Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

About this task

The file is split into variable-length sections, each of which is written to an individual message. The text file is split at each point where the text in the file matches a given regular expression. The source file is called `/tmp/names.text` and has the following contents:

```
Jenny Jones,John Smith,Jane Brown
```

The regular expression that specifies where to split the file is the comma character (`,`).

The source file is located on the same system as the source agent *AGENT_NEPTUNE*, which connects to the queue manager QM_NEPTUNE. The destination queue, *RECEIVING_QUEUE*, is located on the queue manager *QM_MERCURY*. *QM_MERCURY* is also the queue manager used by the destination agent *AGENT_MERCURY*. The transfer splits the source file into sections and writes each of these sections to a message on *RECEIVING_QUEUE*.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_MERCURY -dm QM_MERCURY
                  -dq RECEIVING_QUEUE -t text -dqdp postfix -dqdt "," /tmp/names.text
```

The source agent, AGENT_NEPTUNE, reads the data from the file `/tmp/names.text` and transfers this data to the destination agent, AGENT_MERCURY. The destination agent, AGENT_MERCURY, writes the data to three persistent messages on the queue *RECEIVING_QUEUE*. These messages all have the same IBM MQ group ID and the last message in the group has the IBM MQ LAST_MSG_IN_GROUP flag set.

The data in the messages is as follows.

- First message:

  ```
  Jenny Jones
  ```

- Second message:

```
John Smith
```

- Third message:

```
Jane Brown
```

Related concepts

“Transferring data from files to messages” on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

“Configuring an agent to perform file-to-message transfers” on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

“Example: Transferring a single file to a single message” on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

“Example: Splitting a single file into multiple messages by length” on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

“Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages” on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

“Example: Setting IBM MQ message properties on a file-to-message transfer” on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

“Example: Setting user-defined properties on a file-to-message transfer” on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

“Starting a new file transfer” on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

“Failure of a file-to-message transfer” on page 277
If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

Regular expressions used by MFT

Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages

Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

About this task

Transfer a single text file to multiple messages on a queue. The file is split into variable-length sections, each of which is written to an individual message. The text file is split at each point where the text in the file matches a given regular expression. The source file is called `/tmp/customers.text` and has the following contents:

```
Customer name: John Smith
Customer contact details: john@example.net
Customer number: 314

Customer name: Jane Brown
Customer contact details: jane@example.com
Customer number: 42

Customer name: James Jones
Customer contact details: jjones@example.net
Customer number: 26
```

The regular expression that specifies where to split the file is `Customer\snumber:\s\d+`, which matches the text "Customer number: " followed by any number of digits. Regular expressions specified at the command line must be enclosed in double quotation marks to prevent the command shell evaluating the regular expression. The regular expression is evaluated as a Java regular expression. For more information, see Regular expressions used by MFT.

By default the number of characters that a regular expression can match is set to five. The regular expression used in this example matches strings that are longer than five characters. To enable matches that are longer than five characters edit the agent properties file to include the property **maxDelimiterMatchLength**.

By default, the text that matches the regular expression is not included in the messages. To include the text that matches the regular expression in the messages, as in this example, use the **-qi** parameter. The source file is located on the same system as the source agent AGENT_NEPTUNE, which connects to the queue manager QM_NEPTUNE. The destination queue, RECEIVING_QUEUE, is located on the queue manager QM_MERCURY. QM_MERCURY is also the queue manager used by the destination agent AGENT_MERCURY. The transfer splits the source file into sections and writes each of these sections to a message on RECEIVING_QUEUE.

Procedure

1. Stop the destination agent using the following command:

   ```
   fteStopAgent AGENT_MERCURY
   ```

2. Add the following line to the agent properties file for AGENT_MERCURY:

   ```
   maxDelimiterMatchLength=25
   ```

   Note: Increasing the value of **maxDelimiterMatchLength** can decrease performance.

3. Start the destination agent using the following command:

   ```
   fteStartAgent AGENT_MERCURY
   ```

4. Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_MERCURY -dm QM_MERCURY
 -dq RECEIVING_QUEUE
     text -dqdt "Customer\snumber:\s\d+" -qi -dqdp postfix /tmp/customers.text
```

The source agent, AGENT_NEPTUNE, reads the data from the file `/tmp/customers.text` and transfers this data to the destination agent, AGENT_MERCURY. The destination agent, AGENT_MERCURY, writes the data to three persistent messages on the queue RECEIVING_QUEUE. These messages all have the same IBM MQ group ID and the last message in the group has the IBM MQ LAST_MSG_IN_GROUP flag set.

The data in the messages is as follows.

- First message:

```
Customer name: John Smith
Customer contact details: john@example.net
Customer number: 314
```

- Second message:

```
Customer name: Jane Brown
Customer contact details: jane@example.com
Customer number: 42
```

- Third message:

```
Customer name: James Jones
Customer contact details: jjones@example.net
Customer number: 26
```

Related concepts

“Transferring data from files to messages” on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

“Configuring an agent to perform file-to-message transfers” on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

“Example: Transferring a single file to a single message” on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

“Example: Splitting a single file into multiple messages by length” on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

“Example: Splitting a text file into multiple messages using a regular expression delimiter” on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

“Example: Setting IBM MQ message properties on a file-to-message transfer” on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

“Example: Setting user-defined properties on a file-to-message transfer” on page 274

User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

The MFT `agent.properties` file

Regular expressions used by MFT

Example: Setting IBM MQ message properties on a file-to-message transfer

You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

About this task

Include the parameter `-qmp true` in the **fteCreateTransfer** command. In this example, the MQMD user ID of the user submitting the command is `larmer`.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_JUPITER -da AGENT_SATURN -dq MY_QUEUE@MyQM -qmp true
      -t text /tmp/source_file.txt
```

The IBM MQ message properties of the first message written by the destination agent, AGENT_SATURN, to the queue, MY_QUEUE, on queue manager, MyQM, are set to these values:

```
usr.WMQFTETransferId=414cbaedefa234889d999a8ed09782395ea213ebbc9377cd
usr.WMQFTETransferMode=text
usr.WMQFTESourceAgent=AGENT_JUPITER
usr.WMQFTEDestinationAgent=AGENT_SATURN
usr.WMQFTEFileName=source_file.txt
usr.WMQFTEFileSize=1024
usr.WMQFTEFileLastModified=1273740879040
usr.WMQFTEFileIndex=0
usr.WMQFTEMqmdUser=larmer
```

Related concepts

"Transferring data from files to messages" on page 263
You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

"Configuring an agent to perform file-to-message transfers" on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

"Example: Transferring a single file to a single message" on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

“Example: Splitting a single file into multiple messages by length” on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

“Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages” on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

“Example: Splitting a text file into multiple messages using a regular expression delimiter” on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

“Example: Setting user-defined properties on a file-to-message transfer” on page 274
User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

“Starting a new file transfer” on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

“Failure of a file-to-message transfer” on page 277
If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

MQ message properties set by MFT on messages written to destination queues

Example: Setting user-defined properties on a file-to-message transfer

User-defined metadata is set as an IBM MQ message property on the first message written to the destination queue by the transfer. IBM MQ message properties enable an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

About this task

Include the parameters `-qmp true` and `-md account=123456` in the **fteCreateTransfer** command, to set the `usr.account` property to 123456 in the RFH2 header.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_JUPITER -da AGENT_SATURN -dq MY_QUEUE@MyQM
                  -qmp true -md account=123456 /tmp/source_file.txt
```

In addition to the standard set of IBM MQ message properties, the user-defined property is set in the message header of the first message written by the destination agent, `AGENT_SATURN`, to the queue, `MY_QUEUE`, on queue manager, `MyQM`. The header is set to the following value:

```
usr.account=123456
```

The prefix `usr` is added to the beginning of the name of the user-defined metadata.

Related concepts

“Transferring data from files to messages” on page 263

You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

"Configuring an agent to perform file-to-message transfers" on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

"Example: Transferring a single file to a single message" on page 266
You can specify a queue as the destination of a file transfer by using the **-dq** parameter with the **fteCreateTransfer** command. The source file must be smaller than the maximum message length set on the destination queue. The destination queue does not have to be on the same queue manager as the queue manager that the destination agent connects to, but these two queue managers must be able to communicate.

"Example: Splitting a single file into multiple messages by length" on page 268
You can split a file into multiple IBM MQ messages by using the **-qs** parameter of the **fteCreateTransfer** command. The file is split into fixed-length sections, each of which is written to an individual message.

"Example: Splitting a text file with a regular expression delimiter and including the delimiter in the messages" on page 271
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression and include the regular expression match in the resulting messages. To do this you use the **-dqdt** and **-qi** parameters of the **fteCreateTransfer** command.

"Example: Splitting a text file into multiple messages using a regular expression delimiter" on page 269
Transfer a single text file to multiple messages by splitting the file at each match of a given Java regular expression. To do this you use the **-dqdt** parameter of the **fteCreateTransfer** command.

"Example: Setting IBM MQ message properties on a file-to-message transfer" on page 273
You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

MQ message properties set by MFT on messages written to destination queues

Example: adding a user-defined message property for a file-to-message transfer

If you are using Managed File Transfer for message-to-file managed transfers, you can include a user-defined message property for the resulting message.

About this task

You can use any of the following methods to define a custom message property:

- Specify the **-md** parameter on the transfer request. For more information, see "Example: Setting user-defined properties on a file-to-message transfer" on page 274.
- Use an Ant task; you can use either fte:filecopy or fte:filemove. The following example is a fte:filecopy task:

```
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="complete">
<!-- Initialise the properties used in this script.-->

<target name="init" description="initialise task properties">
               <property name="src.file" value="/home/user/file1.bin"/>
               <property name="dst.queue" value="TEST.QUEUE@qm2"/>
               <fte:uuid property="job.name" length="8"
```

```
prefix="copyjob#"/>
</target>
<target name="step1" depends="init" description="transfer file">

<fte:filecopy cmdqm="qm0@localhost@1414@SYSTEM.DEF.SVRCONN"
              src="agent1@qm1" dst="agent2@qm2"
             rcproperty="copy.result">

<fte:metadata>
<fte:entry name="fileName" value="${FileName}"/>
</fte:metadata>

<fte:filespec srcfilespec="${src.file}" dstqueue="${dst.queue}"
dstmsgprops="true"/>

</fte:filecopy>

</target>
</project>
```

- Use a resource monitor and variable substitution. The following example shows some transfer task XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<monitor:monitor
xmlns:monitor="https://www.ibm.com/xmlns/wmqfte/7.0.1/MonitorDefinition"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance" version="5.00"
xsi:schemaLocation="https://www.ibm.com/xmlns/wmqfte/7.0.1/MonitorDefinit ion ./Monitor.xsd">
 <name>METADATA</name>
 <pollInterval units="minutes">5</pollInterval>
 <batch maxSize="5"/>
 <agent>AGENT1</agent>
 <resources>
   <directory recursionLevel="0">e:\temp</directory>
 </resources>
 <triggerMatch>
   <conditions>
     <allOf>
       <condition>
         <fileMatch>
           <pattern>*.txt</pattern>
         </fileMatch>
       </condition>
     </allOf>
   </conditions>
 </triggerMatch>
 <tasks>
   <task>
     <name/>
     <transfer>
       <request version="5.00"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
         <managedTransfer>
           <originator>
             <hostName>mqjason.raleigh.ibm.com.</hostName>
             <userID>administrator</userID>
           </originator>
           <sourceAgent QMgr="AGENTQM" agent="AGENT1"/>
           <destinationAgent QMgr="AGENTQM" agent="AGENT2"/>
           <transferSet priority="0">
             <metaDataSet>
               <metaData key="FileName">${FileName}</metaData>
             </metaDataSet>
             <item checksumMethod="MD5" mode="text">
               <source disposition="delete" recursive="false">
                 <file>${FilePath}</file>
               </source>
               <destination type="queue">
                 <queue persistent="true"
setMqProps="true">TEST.QUEUE@AGENTQM</queue>
               </destination>
             </item>
           </transferSet>
           <job>
             <name>Metadata_example</name>
           </job>
         </managedTransfer>
       </request>
     </transfer>
```

```
    </task>
  </tasks>
  <originator>
    <hostName>mqjason.raleigh.ibm.com.</hostName>
    <userID>administrator</userID>
  </originator>
</monitor:monitor>
```

Related tasks

“Example: Setting IBM MQ message properties on a file-to-message transfer” on page 273

You can use the **-qmp** parameter on the **fteCreateTransfer** command to specify whether IBM MQ message properties are set on the first message written to the destination queue by the transfer. IBM MQ message properties allow an application to select messages to process, or to retrieve information about a message without accessing IBM MQ Message Descriptor (MQMD) or MQRFH2 headers.

Related reference

fte:filecopy Ant task

fte:filemove Ant task

Failure of a file-to-message transfer

If a file-to- message transfer fails after the agent has started writing file data to the destination queue, the agent writes a message to the queue to indicate to an application consuming the messages that a failure has occurred.

The message written to the destination queue if a failure occurs:

- Is blank
- Has the same IBM MQ group ID as the previous message written to the destination queue by the agent
- Has the IBM MQ LAST_MSG_IN_GROUP flag set
- Contains additional IBM MQ message properties, if message properties are enabled. For more information, see the topic MQ message properties set by MFT on messages written to destination queues.

Example

A transfer is requested by running the following command:

```
fteCreateTransfer -sa AGENT_JUPITER -da AGENT_SATURN -dq RECEIVING_QUEUE
                  -qmp true -qs 1K /tmp/source1.txt
```

The file `source1.txt` is 48 KB. The transfer splits this file into 1 KB messages and writes these messages to the destination queue RECEIVING_QUEUE.

While the transfer is in progress, after the agent has written 16 messages to RECEIVING_QUEUE, a failure occurs at the source agent.

The agent writes a blank message to RECEIVING_QUEUE. In addition to the standard set of message properties, the blank message has the following message properties set:

```
usr.WMQFTEResultCode = 40
usr.WMQFTESupplement = BFGTR0036I: The transfer failed to complete successfully.
```

From IBM MQ 9.3.0, when a transfer from a file fails, because of a delimiter size check error, only one empty message is sent. Additionally, the user properties are added to this message if transfer failure was due to the delimiter exceeding the set size on the destination agent.

Related concepts

“Transferring data from files to messages” on page 263

You can use the file-to-message feature of Managed File Transfer to transfer data from a file to a single message, or multiple messages, on an IBM MQ queue.

Related tasks

"Configuring an agent to perform file-to-message transfers" on page 265
By default agents cannot perform file-to-message or message-to-file transfers. To enable this function you must set the agent property enableQueueInputOutput to true. To enable writing to IBM MQ clustered queues, you must also set the agent property enableClusterQueueInputOutput to true.

"Starting a new file transfer" on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

Related reference

The MFT `agent.properties` file
MQ message properties set by MFT on messages written to destination queues

Transferring data from messages to files

The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

For information about file-to-message transfers, see "Transferring data from files to messages" on page 263.

Attention: The source agent for a message-to-file transfer cannot be a protocol bridge agent or a Connect:Direct bridge agent.

You can transfer IBM MQ message data to a file. The following types of message-to-file transfer are supported:

- From a single message to a single file
- From multiple messages to a single file
- From multiple messages with the same IBM MQ group ID to a single file.
- From multiple messages to a single file, including a text or binary delimiter between the data from each message written to the file.

If you are transferring files from large messages, or many small messages, you might need to change some IBM MQ or Managed File Transfer properties. For more information about, see Guidance for setting MQ attributes and MFT properties associated with message size.

In a message to file transfer, the source agent browses the messages from the source queue, unlike the destructive GET in earlier versions of IBM MQ. The messages are removed from the source queue after all messages (in a group if message grouping is used) have been browsed and data written to the destination file. This allows messages to remain in the source queue if a transfer fails or is canceled. Due to this change, authority to BROWSE must also be provided along with GET authority to run message to file transfers.

Managed File Transfer compares the transfer identifier and the value of the groupId attribute within the transfer request XML payload. If these two identifiers are equivalent, the source agent uses the identifier as a message identifier match option (as opposed to a group identifier match option) for the first MQGET attempt that is made on the input queue for the message-to-file transfer.

Related tasks

"Example: Configuring an MFT resource" on page 241

You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **-mq** parameter with the **fteCreateMonitor** command.

Related reference

MQ message properties read by MFT from messages on source queues

Guidance for setting MQ attributes and MFT properties associated with message size

Configuring an agent to perform message-to-file transfers

By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

About this task

If you attempt to perform a message to file transfer from a source agent that does not have the `enableQueueInputOutput` property set to true, the transfer fails. The transfer log message that is published to the coordination queue manager contains the following message:

```
BFGIO0197E: An attempt to read from a queue was rejected by the source agent.
The agent must have enableQueueInputOutput=true set in the agent.properties file
to support transferring from a queue.
```

To enable the agent to write to and read from queues perform the following steps:

Procedure

1. Stop the source agent using the **fteStopAgent** command.
2. Edit the `agent.properties` file to include the line `enableQueueInputOutput=true`.

   The `agent.properties` file is located in the directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*source_agent_name*.
3. Start the source agent using the **fteStartAgent** command.

Related concepts

“Transferring data from messages to files” on page 278

The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

“Example: Transferring from a queue to a single file” on page 280

You can specify an IBM MQ queue as the source of a file transfer by using the **-sq** parameter with the **fteCreateTransfer** command.

“Example: Transferring a group of messages from a queue to a single file” on page 281

You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **-sq** and **-sqgi** parameters with the **fteCreateTransfer** command.

“Example: Inserting a text delimiter before the data from each message” on page 282

When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **-sq**, **-sqdt** and **-sqdp** parameters with the **fteCreateTransfer** command.

“Example: Inserting a binary delimiter after the data from each message” on page 283

When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

“Monitoring a queue and using variable substitution” on page 247

You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to

be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

“Example: Failing a message-to-file transfer using IBM MQ message properties” on page 286
You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

Related reference

The MFT `agent.properties` file

Example: Transferring from a queue to a single file

You can specify an IBM MQ queue as the source of a file transfer by using the **-sq** parameter with the **fteCreateTransfer** command.

About this task

The source data is contained in three messages on the queue START_QUEUE. This queue must be on the source agent's queue manager, *QM_NEPTUNE*.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE
                  -da AGENT_VENUS -df /out/three_to_one.txt
                  -sq START_QUEUE
```

The data in the messages on the queue *START_QUEUE* is written to the file `/out/three_to_one.txt` on the system where AGENT_VENUS is running.

Related concepts

“Transferring data from messages to files” on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

“Configuring an agent to perform message-to-file transfers” on page 279
By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

“Example: Transferring a group of messages from a queue to a single file” on page 281
You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **-sq** and **-sqgi** parameters with the **fteCreateTransfer** command.

“Example: Inserting a text delimiter before the data from each message” on page 282
When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **-sq**, **-sqdt** and **-sqdp** parameters with the **fteCreateTransfer** command.

“Example: Inserting a binary delimiter after the data from each message” on page 283
When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

“Monitoring a queue and using variable substitution” on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to

be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

“Example: Failing a message-to-file transfer using IBM MQ message properties” on page 286
You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

Related reference

MQ message properties read by MFT from messages on source queues

`fteCreateTransfer`: start a new file transfer

Example: Transferring a group of messages from a queue to a single file

You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **`-sq`** and **`-sqgi`** parameters with the **`fteCreateTransfer`** command.

About this task

In this example, there are ten messages on the queue START_QUEUE. This queue must be on the source agent's queue manager, *QM_NEPTUNE*. The first three messages belong to a group with the IBM MQ group ID `41424b3ef3a2202020202020202020202020202020201111`; this group is not a complete group. The next five messages belong to a group with the IBM MQ group ID `41424b3ef3a2202020202020202020202020202020202222`; this group is complete. The remaining two messages belong to a group with the IBM MQ group ID `41424b3ef3a2202020202020202020202020202020203333`; this group is complete.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_VENUS
                  -df /out/group.txt -sqgi -sq START_QUEUE
```

The data in the messages belonging to the first complete group on the queue *START_QUEUE*, the group with IBM MQ group ID `41424b3ef3a2202020202020202020202020202020202222`, is written to the file `/out/group.txt` on the system where AGENT_VENUS is running.

Related concepts

“Transferring data from messages to files” on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

“Configuring an agent to perform message-to-file transfers” on page 279
By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

“Example: Transferring from a queue to a single file” on page 280
You can specify an IBM MQ queue as the source of a file transfer by using the **`-sq`** parameter with the **`fteCreateTransfer`** command.

“Example: Inserting a text delimiter before the data from each message” on page 282
When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **`-sq`**, **`-sqdt`** and **`-sqdp`** parameters with the **`fteCreateTransfer`** command.

“Example: Inserting a binary delimiter after the data from each message” on page 283

When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

"Monitoring a queue and using variable substitution" on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

"Example: Failing a message-to-file transfer using IBM MQ message properties" on page 286
You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

Related reference

fteCreateTransfer: start a new file transfer

Example: Inserting a text delimiter before the data from each message

When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **-sq**, **-sqdt** and **-sqdp** parameters with the **fteCreateTransfer** command.

About this task

In this example, there are four messages on the queue START_QUEUE. This queue is on the source agent's queue manager, *QM_NEPTUNE*. The text delimiter to be inserted before the data from each message can be expressed as a Java literal string, for example: `\n\u002D\u002D\u002D\n`.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_VENUS -df /out/output.txt
                  -t text -sqdt "\n\u002D\u002D\u002D\n" -sqdp prefix -sq START_QUEUE
```

The text delimiter is added to the beginning of the data from each of the four messages on START_QUEUE by the source agent, AGENT_NEPTUNE. This data is written to the destination file, `/out/output.txt`.

Related concepts

"Transferring data from messages to files" on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

"Configuring an agent to perform message-to-file transfers" on page 279
By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

"Example: Transferring from a queue to a single file" on page 280
You can specify an IBM MQ queue as the source of a file transfer by using the **-sq** parameter with the **fteCreateTransfer** command.

"Example: Transferring a group of messages from a queue to a single file" on page 281
You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **-sq** and **-sqgi** parameters with the **fteCreateTransfer** command.

"Example: Inserting a binary delimiter after the data from each message" on page 283

When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

"Monitoring a queue and using variable substitution" on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

"Example: Failing a message-to-file transfer using IBM MQ message properties" on page 286
You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

Related reference

fteCreateTransfer: start a new file transfer

Example: Inserting a binary delimiter after the data from each message

When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

About this task

In this example, there are three messages on the queue START_QUEUE. This queue is on the source agent's queue manager, *QM_NEPTUNE*. The binary delimiter to be inserted after the data from each message must be expressed as a comma-separated list of hexadecimal bytes, for example: `x34,xE7,xAE`.

Procedure

Type the following command:

```
fteCreateTransfer -sa AGENT_NEPTUNE -sm QM_NEPTUNE -da AGENT_VENUS -df /out/binary.file
                  -sqdp postfix -sqdb x34,xE7,xAE -sq START_QUEUE
```

The binary delimiter is appended to the data from each of the three messages on START_QUEUE by the source agent, AGENT_NEPTUNE. This data is written to the destination file, `/out/binary.file`.

Related concepts

"Transferring data from messages to files" on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

"Configuring an agent to perform message-to-file transfers" on page 279
By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

"Example: Transferring from a queue to a single file" on page 280
You can specify an IBM MQ queue as the source of a file transfer by using the **-sq** parameter with the **fteCreateTransfer** command.

"Example: Transferring a group of messages from a queue to a single file" on page 281

You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **-sq** and **-sqgi** parameters with the **fteCreateTransfer** command.

"Example: Inserting a text delimiter before the data from each message" on page 282
When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **-sq**, **-sqdt** and **-sqdp** parameters with the **fteCreateTransfer** command.

"Monitoring a queue and using variable substitution" on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

"Example: Failing a message-to-file transfer using IBM MQ message properties" on page 286
You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

Related reference

fteCreateTransfer: start a new file transfer

Monitoring a queue and using variable substitution

You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

About this task

In this example, the source agent is called AGENT_VENUS, which connects to QM_VENUS. The queue that AGENT_VENUS monitors is called START_QUEUE and is located on QM_VENUS. The agent polls the queue every 30 minutes.

When a complete group of messages is written to the queue the monitor task sends the group of messages to a file at one of a number of destination agents, all of which connect to the queue manager QM_MARS. The name of the file that the group of messages is transferred to is defined by the IBM MQ message property `usr.fileName` on the first message in the group. The name of the agent that the group of messages is sent to is defined by the IBM MQ message property `usr.toAgent` on the first message in the group. If the `usr.toAgent` header is not set, the default value to be used for the destination agent is AGENT_MAGENTA.

When you specify `useGroups="true"`, if you do not also specify `groupId="${GROUPID}"`, the transfer just takes the first message on the queue. For example, if you are using variable substitution to generate the fileName, it is therefore possible that the contents of a `.txt` will not be correct. This is because the fileName is generated by the monitor, but the transfer actually gets a message that is not the one that should generate the file called fileName.

Procedure

1. Create the task XML that defines the task that the monitor performs when it is triggered.

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_VENUS" QMgr="QM_VENUS" />
     <destinationAgent agent="${toAgent}" QMgr="QM_MARS" />
     <transferSet>
```

```
       <item mode="binary" checksumMethod="none">
         <source>
           <queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/${fileName}.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

The variables that are replaced with the values of IBM MQ message headers are highlighted in **bold**. This task XML is saved to the file `/home/USER1/task.xml`

2. Create a resource monitor to monitor the queue START_QUEUE.

   Submit the following command:

```
fteCreateMonitor -ma AGENT_VENUS -mm QM_VENUS -mq START_QUEUE
                 -mn myMonitor -mt /home/USER1/task.xml
                 -tr completeGroups -pi 30 -pu minutes -dv toAgent=AGENT_MAGENTA
```

3. A user or program writes a group of messages to the queue START_QUEUE.

   The first message in this group has the following IBM MQ message properties set:

```
usr.fileName=larmer
usr.toAgent=AGENT_VIOLET
```

4. The monitor is triggered when the complete group is written. The agent substitutes the IBM MQ message properties into the task XML.

   This results in the task XML being transformed to:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request version="4.00"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
   <managedTransfer>
     <originator>
       <hostName>reportserver.com</hostName>
       <userID>USER1</userID>
     </originator>
     <sourceAgent agent="AGENT_VENUS" QMgr="QM_VENUS" />
     <destinationAgent agent="AGENT_VIOLET" QMgr="QM_MARS" />
     <transferSet>
       <item mode="binary" checksumMethod="none">
         <source>
           <queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
         </source>
         <destination type="file" exist="overwrite">
           <file>/reports/larmer.rpt</file>
         </destination>
       </item>
     </transferSet>
   </managedTransfer>
</request>
```

Results

The transfer that is defined by the task XML is performed. The complete group of messages that are read from the START_QUEUE by AGENT_VENUS is written to a file called `/reports/larmer.rpt` on the system where AGENT_VIOLET is running.

What to do next

Transferring each message to a separate file

If you want to monitor a queue and have every message transferred to a separate file, you can use a similar technique to the one described previously in this topic.

1. Create the monitor as described previously, specifying the **`-tr completeGroups`** parameter on the **`fteCreateMonitor`** command.

2. In the task XML specify the following:

```
<queue useGroups="true" groupId="${GROUPID}">START_QUEUE</queue>
```

However, when you put the messages onto the source queue, do not put them in an IBM MQ group. Add IBM MQ message properties to each message. For example, specify the `usr.filename` property with a unique file name value for each message. This effectively causes the Managed File Transfer Agent to treat each message on the source queue as a separate group.

Related concepts

"Transferring data from messages to files" on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

"Customizing MFT resource monitor tasks with variable substitution" on page 242
When the trigger conditions of an active resource monitor are satisfied, the defined task is called. In addition to calling the transfer or command task with the same destination agent or the same destination file name every time, you can also modify the task definition at run time. You do this by inserting variable names into the task definition XML. When the monitor determines that the trigger conditions are satisfied, and that the task definition contains variable names, it substitutes the variable names with the variable values, and then calls the task.

What to do if destination files created by a transfer started by a queue resource monitor contain the wrong data

Related tasks

"Configuring MFT monitor tasks to start commands and scripts" on page 235
Resource monitors are not limited to performing file transfers as their associated task. You can also configure the monitor to call other commands from the monitoring agent, including executable programs, Ant scripts or JCL jobs. To call commands, edit the monitor task definition XML to include one or more command elements with corresponding command call parameters, such as arguments and properties.

"Example: Configuring an MFT resource" on page 241
You can specify an IBM MQ queue as the resource to be monitored by a resource monitor by using the **-mq** parameter with the **fteCreateMonitor** command.

Related reference

fteCreateMonitor: create an MFT resource monitor

MQ message properties read by MFT from messages on source queues

Example: Failing a message-to-file transfer using IBM MQ message properties

You can cause a message to file transfer to fail by setting the `usr.UserReturnCode` IBM MQ message property to a non-zero value. You can also specify supplementary information about the reason for the failure by setting the `usr.UserSupplement` IBM MQ message property.

About this task

In this example, a transfer is in progress between the queue INPUT_QUEUE and the file `/home/user/output.file`.

A user is creating messages and placing them on the queue INPUT_QUEUE. The source agent is consuming messages from the queue INPUT_QUEUE and is sending the transfer data to the destination agent. The destination agent is writing this data to the file `/home/user/output.file`.

The user writing messages to the queue INPUT_QUEUE wants to stop the transfer that is in progress and delete any data that has already been written to the destination file.

Procedure

1. The user writes a message to the queue INPUT_QUEUE that has the following IBM MQ message properties set:

```
usr.UserReturnCode=1
usr.UserSupplement="Cancelling transfer - sent wrong data."
```

2. The source agent reads the IBM MQ message properties and stops processing messages from the queue. The destination agent deletes any file data that has been written to the destination directory.
3. The source agent sends a transfer log message to the coordination queue manager reporting the transfer failure.

   The message contains the following information:

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction version="1.00"
               ID="414d5120514d312020202020202020207e970d4920008702" agentRole="sourceAgent"
             xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="TransferLog.xsd"
             xmlns="">
  <action time="2008-11-02T21:28:09.593Z">progress</action>
  <sourceAgent agent="FTEAGENT" QMgr="QM1">
    <systemInfo architecture="x86" name="Windows 7"
           version="6.1 build 7601 Service Pack 1" />
  </sourceAgent>
  <destinationAgent agent="FTEAGENT" QMgr="QM1">
    <systemInfo architecture="x86" name="Windows 7"
           version="6.1 build 7601 Service Pack 1" />
  </destinationAgent>
  <originator>
    <hostName>reportserver.com</hostName>
    <userID>USER1</userID>
    <mqmdUserID>USER1   </mqmdUserID>
  </originator>
  <transferSet index="0" size="1"
               startTime="2008-11-02T21:28:09.281Z"
               total="1">
    <item mode="binary">
      <source>
        <queue>INPUT_QUEUE@QM1</queue>
      </source>
      <destination exist="error">
        <file>/home/user/output.file</file>
      </destination>
      <status resultCode="1">
        <supplement>Cancelling transfer - sent wrong data.</supplement>
      </status>
    </item>
  </transferSet>
</transaction>
```

Related concepts

“Transferring data from messages to files” on page 278
The message-to-file feature of Managed File Transfer enables you to transfer data from one or more messages on an IBM MQ queue to a file, a data set (on z/OS), or a user file space. If you have an application that creates or processes IBM MQ messages, you can use the message-to-file capability of Managed File Transfer to transfer these messages to a file on any system in your Managed File Transfer network.

Related tasks

“Configuring an agent to perform message-to-file transfers” on page 279
By default agents cannot perform message to file, or file to message, transfers. To enable this function you must set the agent property `enableQueueInputOutput` to true.

“Example: Transferring from a queue to a single file” on page 280
You can specify an IBM MQ queue as the source of a file transfer by using the **-sq** parameter with the **fteCreateTransfer** command.

“Example: Transferring a group of messages from a queue to a single file” on page 281

You can specify a single complete group on an IBM MQ queue as the source of a file transfer by using the **-sq** and **-sqgi** parameters with the **fteCreateTransfer** command.

"Example: Inserting a text delimiter before the data from each message" on page 282
When you are transferring in text mode from a source queue to a file, you can specify that a text delimiter is inserted before the data from individual messages by using the **-sq**, **-sqdt** and **-sqdp** parameters with the **fteCreateTransfer** command.

"Example: Inserting a binary delimiter after the data from each message" on page 283
When transferring in binary mode from a source queue to a file, you can specify that a binary delimiter is inserted after the data from individual messages by using the **-sq**, **-sqdb**, and **-sqdp** parameters with the **fteCreateTransfer** command.

"Monitoring a queue and using variable substitution" on page 247
You can monitor a queue and transfer messages from the monitored queue to a file by using the **fteCreateMonitor** command. The value of any IBM MQ message property in the first message to be read from the monitored queue can be substituted in the task XML definition and used to define the transfer behavior.

Related reference

MQ message properties read by MFT from messages on source queues

The protocol bridge

The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

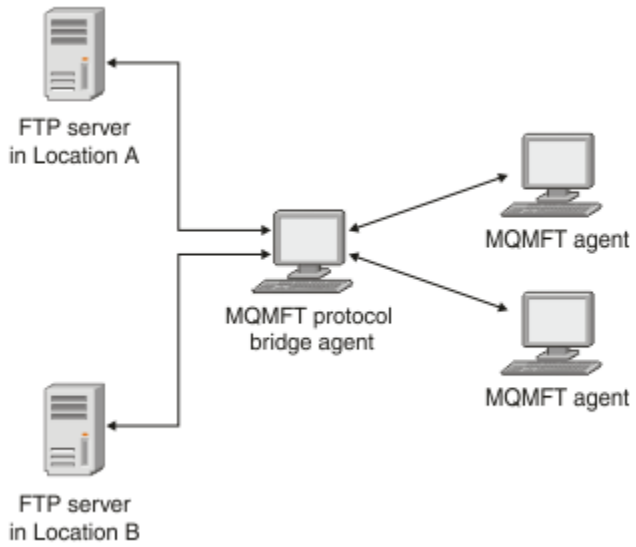
The protocol bridge is available as part of the Service component of Managed File Transfer. You can have multiple dedicated agents on a single system running MFT that connect to different file servers.

You can use a protocol bridge agent to transfer files to multiple endpoints simultaneously. MFT provides a file called `ProtocolBridgeProperties.xml` that you can edit to define the different protocol file servers that you want to transfer files to. The **fteCreateBridgeAgent** command adds the details of the default protocol file server to `ProtocolBridgeProperties.xml` for you. This file is described in Protocol bridge properties file format.

You can use the protocol bridge agent to perform the following actions:

- Upload files from the MFT network to a remote server using FTP, FTPS, or SFTP.
- Download files from a remote server, using FTP, FTPS, or SFTP, to the MFT network

Note: The protocol bridge agent can support only FTP, FTPS, or SFTP servers that allow files to be accessed by their absolute file path. If a relative file path is specified in a transfer request, the protocol bridge agent will attempt to convert the relative path into an absolute file path based on the home directory used to login to the protocol server. Those protocol servers that allow access to files based only on the current directory are not supported by the protocol bridge agent.



The diagram shows two FTP servers, at different locations. The FTP servers are being used to exchange files with the Managed File Transfer agents. The protocol bridge agent is between the FTP servers and the rest of the MFT network, and is configured to communicate with both FTP servers.

Ensure that you have another agent in your MFT network in addition to the protocol bridge agent. The protocol bridge agent is a bridge to the FTP, FTPS, or SFTP server only and does not write transferred files to the local disk. If you want to transfer files to or from the FTP, FTPS, or SFTP server you must use the protocol bridge agent as the destination or source for the file transfer (representing the FTP, FTPS, or SFTP server) and another standard agent as the corresponding source or destination.

When you transfer files using the protocol bridge, the bridge must have permission to read the source or destination directory containing the files you want to transfer. For example, if you want to transfer files from the directory `/home/fte/bridge` that has execute permissions (d--x--x--x) only, any transfers you attempt from this directory fail with the following error message:

```
BFGBR0032E: Attempt to read filename from the protocol file server
  has failed with server error 550. Failed to open file.
```

Configuring a protocol bridge agent

A protocol bridge agent is like a standard MFT agent. Create a protocol bridge agent by using the **fteCreateBridgeAgent** command. You can configure a protocol bridge agent using the `ProtocolBridgeProperties.xml` file, which is described in Protocol bridge properties file format. If you are using an earlier version, configure the agent using the specific protocol bridge properties described in Advanced agent properties: Protocol bridge and Advanced agent properties: Protocol bridge agent logging. For all versions, you can also configure a credential mapping as described in “Mapping credentials for a file server” on page 296. After you have configured a protocol bridge agent for a particular protocol file server, you can then use that agent for that purpose only.

Protocol bridge recovery

If the protocol bridge agent is unable to connect to the file server because the file server is unavailable, all file transfer requests are queued until the file server becomes available. If the protocol bridge agent is unable to connect to the file server because the agent is using the wrong credentials, the transfer fails and the transfer log message reflects this error. If the protocol bridge agent is ended for any reason, all requested file transfers are retained and continue when the protocol bridge is restarted.

During file transfer, files are typically written as temporary files at the destination and are then renamed when the transfer is complete. However, if the transfer destination is a protocol file server that is configured as limited write (users can upload files to the protocol file server but cannot change those uploaded files in any way; effectively users can write once only), transferred files are written to the

destination directly. This means that if a problem occurs during the transfer, the partially written files remain on the destination protocol file server and Managed File Transfer cannot delete or edit these files. In this situation, the transfer fails.

Related tasks

"Example: How to configure a protocol bridge agent to use private key credentials with a UNIX SFTP server" on page 301
This example demonstrates how you can generate and configure the `ProtocolBridgeCredentials.xml` file. This example is a typical example and the details might vary according to your platform, but the principles remain the same.

"Defining properties for protocol file servers using the ProtocolBridgeProperties.xml file" on page 290
Define the properties of one or more protocol file servers that you want to transfer files to and from using the `ProtocolBridgeProperties.xml` file, which is provided by Managed File Transfer in the agent configuration directory.

Related reference

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

"Mapping credentials for a file server" on page 296
Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent or by writing your own user exit. Managed File Transfer provides a sample user exit that performs user credential mapping.

ProtocolBridgeCredentialExit.java interface

Sample protocol bridge credential user exit

FTPS server support by the protocol bridge

Defining properties for protocol file servers using the `ProtocolBridgeProperties.xml` file

Define the properties of one or more protocol file servers that you want to transfer files to and from using the `ProtocolBridgeProperties.xml` file, which is provided by Managed File Transfer in the agent configuration directory.

About this task

The **fteCreateBridgeAgent** command creates the `ProtocolBridgeProperties.xml` file in the agent configuration directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*bridge_agent_name*. The command also creates an entry in the file for the default protocol file server, if a default was specified when the command was run.

The message BFGCL0392I gives the location of the `ProtocolBridgeProperties.xml` file.

```
<?xml version="1.0" encoding="IBM-1047"?>
<!--
This ProtocolBridgeProperties.xml file determines the protocol servers that will be accessed by
the
MQMFT protocol bridge agent.

Each protocol server is defined using either a <tns:ftpServer>, <tns:ftpsServer>, or
<tns:sftpServer>
element - depending on the protocol used to communicate with the server.  When the protocol
bridge agent participates in a managed file transfer it will determine which server to used
based on
the prefix (if any) present on the file path.  For example a file path of 'server1:/home/user/
file.txt' would
be interpreted as a request to transfer /home/user/file.txt using 'server1'.  The server name
is compared
to the 'name' attribute of each <tns:ftpServer>, <tns:ftpsServer> or <tns:sftpServer> element
in this
XML document and the first match is used to determine which protocol server the protocol bridge
agent will connect to.  If no match is found then the managed file transfer operation will fail.

If a file path is not prefixed with a server name, for example '/home/user/file.txt' then this
XML
document can specify a default server to use for the managed file transfer.  To specify a
default server use the <tns:defaultServer> element as the first element inside the
```

```
<tns:serverProperties>
element.  The default server will be used whenever the protocol bridge agent participates in
a managed file transfer for file names which do not specify a prefix.

An optional <tns:limits> element can be specified within each server definition. This element
contains
attributes that govern the amount of resources used by each defined server.

An optional <tns:credentialsFile> element can be specified within each serverProperties
definition. This
element contains a path to a file containing credentials to be used when connecting to defined
servers.

An example ProtocolBridgeProperties.xml file is as follows:

<?xml version="1.0" encoding="UTF-8"?>
<tns:serverProperties xmlns:tns="http://wmqfte.ibm.com/ProtocolBridgeProperties"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://wmqfte.ibm.com/ProtocolBridgeProperties
ProtocolBridgeProperties.xsd">

    <tns:credentialsFile path="$HOME/ProtocolBridgeCredentials.xml" />

    <tns:defaultServer name="myFTPserver" />

    <tns:ftpServer name="myFTPserver" host="windows.hursley.ibm.com" port="1234"
platform="windows"
        timeZone="Europe/London" locale="en_GB" fileEncoding="UTF-8"
        listFormat="unix" limitedWrite="false">

        <tns:limits maxListFileNames="100" maxListDirectoryLevels="999999999"
            maxReconnectRetry="2"  reconnectWaitPeriod="10"
            maxSessions="60" socketTimeout="30" />

    </tns:ftpServer>

    <tns:ftpsServer name="myFTPSserver" host="unix.hursley.ibm.com" platform="unix"
        timeZone="Europe/London" locale="en_GB" fileEncoding="UTF8"
        listFormat="unix" limitedWrite="false" ftpsType="explicit"
        trustStore="C:\FTE\keystores\myFTPSserver\FTPSKeyStore.jks"
trustStorePassword="password">

        <tns:limits maxReconnectRetry="10" connectionTimeout="10"/>

    </tns:ftpsServer>

    <tns:sftpServer name="mySFTPserver" host="windows.hursley.ibm.com" platform="windows"
        timeZone="Europe/London" locale="en_GB" fileEncoding="UTF-8"
        limitedWrite="false">

        <tns:limits connectionTimeout="60"/>

    </tns:sftpServer>

</tns:serverProperties>

This example shows the outermost <tns:serverProperties> element which must exist for the
document to
be valid, an optional <tns:defaultServer> element, as well as definitions for an FTP, FTPS and
SFTP server.

The attributes of the <tns:ftpServer>, <tns:ftpsServer> and <tns:sftpServer> elements determine
the
characteristics of the connection established to the server.  These attributes correspond to
the command
line parameters for the 'fteCreateBridgeAgent' command.

The following attributes are valid for all of the <tns:ftpServer>, <tns:ftpsServer> and
<tns:sftpServer>
elements: name, host, port, platform, fileEncoding, limitedWrite and controlEncoding.

The following attributes are valid for the <tns:ftpServer> and <tns:ftpsServer> elements:
timezone, locale,
listFormat, listFileRecentDateFormat, listFileOldDateFormat, and monthShortNames.

The following attributes are valid for the <tns:ftpServer> element only: passiveMode

The following attributes are valid for the <tns:ftpsServer> element only: ftpsType, trustStore,
trustStorePassword,
trustStoreType, keyStore, keyStorePassword, keyStoreType, ccc, protFirst, auth, and
connectTimeout.
```

```
The following attributes are valid for the <tns:limits> element within all of the
<tns:ftpServer>, <tns:ftpsServer>
and <tns:sftpServer> elements: maxListFileNames, maxListDirectoryLevels, maxReconnectRetry,
reconnectWaitPeriod,
maxSessions and socketTimeout

-->
<tns:serverProperties xmlns:tns="http://wmqfte.ibm.com/ProtocolBridgeProperties"
        xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://wmqfte.ibm.com/ProtocolBridgeProperties
ProtocolBridgeProperties.xsd">

    <!-- By default the location of the credentials file is in the home directory of the user
that started the -->
    <!-- protocol bridge agent. If you wish to specify a different location use the
credentialsFile element to -->
    <!-- do this. For
example:                                                                                -->
    <!-- <tns:credentialsFile path="/test/
ProtocolBridgeCredentials.xml"/>                                    -->

    <tns:defaultServer name="WINMVSCA.HURSLEY.IBM.COM" />
    <tns:ftpServer name="WINMVSCA.HURSLEY.IBM.COM" host="WINMVSCA.HURSLEY.IBM.COM"
platform="unix"
                timeZone="Europe/London" locale="en-GB" fileEncoding="US-ASCII"
                listFormat="unix" limitedWrite="false"  />

    <!-- Define servers here -->
</tns:serverProperties>
```

The command can produce the following message:BFGCL0532I:

```
For this agent to function an additional credentials file must be created manually.
By default this file is called ProtocolBridgeCredentials.xml and is located in the home
directory of the user who starts the agent. For example,if this user started the agent
the location would be: $HOME/ProtocolBridgeCredentials.xml
```

If you use a credentials file:

1. See the following text for further information on how to create one.
2. The credentials file must be in a directory with restricted permissions. For example, there must be no read access for other users.
3. Specify the location of the directory for the credentials file in the $HOME environment variable for the user ID of the started agent, or edit the `ProtocolBridgeProperties.xml` file and specify the location in:

   ```
   <tns:credentialsFile path="/test/ProtocolBridgeCredentials.xml"/>
   ```

If you want to add further non-default protocol servers, edit this file to define their properties. This example adds an additional FTP server.

Note: The protocol bridge agent does not support file locking. This is because Managed File Transfer does not support the file locking mechanism on a file server.

Procedure

1. Define a protocol file server by inserting the following lines into the file as a child element of `<tns:serverProperties>`:

```
<tns:ftpServer name="myserver" host="myhost.hursley.ibm.com" port="1234"
   platform="windows"
                          timeZone="Europe/London" locale="en-GB" fileEncoding="UTF-8"
                         listFormat="unix" limitedWrite="false" >
<tns:limits maxListFileNames="10" maxListDirectoryLevels="500"/>
```

2. Then change the value of the attributes:
   - `name` is the name of your protocol file server
   - `host` is the host name or IP address of the protocol file server
   - `port` is the port number of the protocol file server

- `platform` is the platform that the protocol file server runs on
- `timeZone` is the time zone that the protocol file server runs in
- `locale` is the language used on the protocol file server
- `fileEncoding` is the character encoding of the protocol file server
- `listFormat` is the file listing format returned from the protocol file server
- `limitedWrite` determines whether to follow the default mode when writing to a file server, which is to create a temporary file and then rename that file when the transfer has completed. For a file server that is configured as write only, the file is created directly with its final name. The value of this property can be true or false. The `limitedWrite` attribute and the `doNotUseTempOutputFile` agent property are used together in the case of protocol bridge agents. If you want to use temporary files, then you must not set the value of `doNotUseTempOutputFile`, and you must set the value of `limitedWrite` to false. Any other combination of settings means that temporary files will not be used.
- `maxListFileNames` is the maximum number of names collected when scanning a directory on the protocol file server for file names.
- `maxListDirectoryLevels` is the maximum number of directory levels to recurse when scanning a directory on the protocol file server for file names.

For more details about these attributes, including whether they are required or optional and their default values, see Protocol bridge properties file format.

Related reference

Protocol bridge properties file format

Regular expressions used by MFT

Looking up protocol file server properties: ProtocolBridgePropertiesExit2

If you have a large number of protocol file servers, you can implement the `com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2` interface to look up protocol file server properties that are referenced in transfers. You can implement this interface in preference to maintaining a `ProtocolBridgeProperties.xml` file.

About this task

Managed File Transfer provides a sample user exit that looks up protocol file server properties. For more information, see "Using the sample user exit to look up protocol file server properties" on page 294.

Any user exit that looks up protocol bridge properties must implement the interface `com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2`. For more information, see ProtocolBridgePropertiesExit.java interface.

You can chain multiple protocol server properties exits together in a similar manner to other user exits. The exits are called in the order that they are specified in using the `protocolBridgePropertiesExitClasses` property in the agent properties file. The initialize methods all return separately and if one or more returns a value of false, the agent does not start. The error is reported in the agent event log.

Only one overall result is returned for the getProtocolServerProperties methods of all of the exits. If the method returns a properties object as the result code, this value is the returned result and the getProtocolServerProperties methods of the subsequent exits are not called. If the method returns a value of null as the result code, the getProtocolServerProperties method of the next exit is called. If there is no subsequent exit, the null result is returned. An overall result code of null is considered as a lookup failure by the protocol bridge agent.

You are recommended to use the ProtocolBridgePropertiesExit2.java interface, but for information about the ProtocolBridgePropertiesExit.java interface, see "Looking up protocol file server properties: ProtocolBridgePropertiesExit" on page 295.

To run your exit, complete the following steps:

Procedure

1. Compile the protocol server properties user exit.
2. Create a Java archive (JAR) file containing the compiled exit and its package structure.
3. Put the JAR file containing the exit class in the `exits` directory of the protocol bridge agent . This directory is found in the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent_name* directory.
4. Edit the property file of the protocol bridge agent to include the property `protocolBridgePropertiesExitClasses`. For the value of this property, specify a comma-separated list of classes that implement a protocol bridge server properties user exit. The exit classes are called in the order that they are specified in this list. For more information, see The MFT `agent.properties` file.
5. You can optionally specify the `protocolBridgePropertiesConfiguration` property. The value you specify for this property is passed in as a String to the initialize() method of the exit classes specified by `protocolBridgePropertiesExitClasses`. For more information, see The MFT `agent.properties` file.

Using the sample user exit to look up protocol file server properties

Managed File Transfer provides a sample user exit that looks up protocol file server properties.

About this task

A sample user exit that looks up protocol bridge properties is provided in the *MQ_INSTALLATION_PATH*/mqft/samples/protocolBridge directory and in the topic Sample protocol bridge properties user exit.

The `SamplePropertiesExit2.java` exit reads a properties file that contains properties for protocol servers. The format of each entry in the properties file is as follows:

```
serverName=type://host:port
```

The location of the properties file is taken from the protocol bridge agent property `protocolBridgePropertiesConfiguration`.

To run the sample user exit, complete the following steps:

Procedure

1. Compile the `SamplePropertiesExit2.java` file.
2. Create a JAR file containing the compiled exit and its package structure.
3. Put the JAR file in the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent*/exits directory.
4. Edit the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent_name*/agent.properties file to contain the line:

   ```
   protocolBridgePropertiesExitClasses=SamplePropertiesExit2
   ```

5. Create a protocol bridge properties file, for example `protocol_bridge_properties.properties`, in the directory *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent*. Edit this file to include entries in the format:

   ```
   serverName=type://host:port
   ```

6. Edit the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent*/agent.properties file to contain the line:

   ```
   protocolBridgePropertiesConfiguration=MQ_DATA_PATH/mqft/config/coordination_queue_manager/
   agents/bridge_agent/protocol_bridge_properties.properties
   ```

   You must use the absolute path to the `protocol_bridge_properties.properties` file.

7. Start the protocol bridge agent by using the **fteStartAgent** command.

Related concepts

"The protocol bridge" on page 288

The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

Related reference

ProtocolBridgePropertiesExit.java interface

Sample protocol bridge properties user exit

The MFT `agent.properties` file

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

Looking up protocol file server properties: ProtocolBridgePropertiesExit

If you have a large number of protocol file servers, you can implement the `com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit` interface to look up protocol file server properties that are referenced in transfers.

About this task

You can implement the `com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit` interface in preference to maintaining a `ProtocolBridgeProperties.xml` file.
Use the ProtocolBridgePropertiesExit2.java interface. The **getCredentialLocation** method in ProtocolBridgePropertiesExit2.java uses the default location of the `ProtocolBridgeCredentials.xml` file, which is your home directory.

Any user exit that looks up protocol bridge properties must implement the interface `com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit`:

```
package com.ibm.wmqfte.exitroutine.api;

import java.util.Map;
import java.util.Properties;

/**
 * An interface that is implemented by classes that are to be invoked as part of
 * user exit routine processing. This interface defines methods that will be
 * invoked by a protocol bridge agent to look up properties for protocol servers
 * that are referenced in transfers.
 * <p>
 * There will be one instance of each implementation class for each protocol
 * bridge agent. The methods can be called from different threads so the methods
 * must be synchronised.
 */
public interface ProtocolBridgePropertiesExit {

    /**
     * Invoked once when a protocol bridge agent is started. It is intended to
     * initialize any resources that are required by the exit.
     *
     * @param bridgeProperties
     *            The values of properties defined for the protocol bridge.
     *            These values can only be read, they cannot be updated by the
     *            implementation.
     * @return {@code true} if the initialization is successful and {@code
     *         false} if unsuccessful. If {@code false} is returned from an exit
     *         the protocol bridge agent will not start.
     */
    public boolean initialize(final Map<String, String> bridgeProperties);

    /**
     * Obtains a set of properties for the specified protocol server name.
     * <p>
     * The returned {@link Properties} must contain entries with key names
     * corresponding to the constants defined in
     * {@link ProtocolServerPropertyConstants} and in particular must include an
```

```
     * entry for all appropriate constants described as required.
     *
     * @param protocolServerName
     *            The name of the protocol server whose properties are to be
     *            returned. If a null or a blank value is specified, properties
     *            for the default protocol server are to be returned.
     * @return The {@link Properties} for the specified protocol server, or null
     *         if the server cannot be found.
     */
    public Properties getProtocolServerProperties(
            final String protocolServerName);

    /**
     * Invoked once when a protocol bridge agent is shut down. It is intended to
     * release any resources that were allocated by the exit.
     *
     * @param bridgeProperties
     *            The values of properties defined for the protocol bridge.
     *            These values can only be read, they cannot be updated by the
     *            implementation.
     */
    public void shutdown(final Map<String, String> bridgeProperties);

}
```

You can chain multiple protocol server properties exits together in a similar manner to other user exits. The exits are called in the order that they are specified in using the `protocolBridgePropertiesExitClasses` property in the agent properties file. The initialize methods all return separately and if one or more returns a value of false, the agent does not start. The error is reported in the agent event log.

Only one overall result is returned for the getProtocolServerProperties methods of all of the exits. If the method returns a properties object as the result code, this value is the returned result and the getProtocolServerProperties methods of the subsequent exits are not called. If the method returns a value of null as the result code, the getProtocolServerProperties method of the next exit is called. If there is no subsequent exit, the null result is returned. An overall result code of null is considered as a lookup failure by the protocol bridge agent.

Procedure

To run your exit, complete the following steps:

1. Compile the protocol server properties user exit.
2. Create a Java archive (JAR) file containing the compiled exit and its package structure.
3. Put the JAR file containing the exit class in the `exits` directory of the protocol bridge agent.

   This directory is found in the *MQ_DATA_PATH*/`mqft/config/`*coordination_queue_manager*`/agents/`*bridge_agent_name* directory.
4. Edit the property file of the protocol bridge agent to include the property `protocolBridgePropertiesExitClasses`.

   For the value of this property, specify a comma-separated list of classes that implement a protocol bridge server properties user exit. The exit classes are called in the order that they are specified in this list. For more information, see The MFT `agent.properties` file.
5. You can optionally specify the `protocolBridgePropertiesConfiguration` property.

   The value you specify for this property is passed in as a String to the initialize() method of the exit classes specified by `protocolBridgePropertiesExitClasses`. For more information, see The MFT `agent.properties` file.

Mapping credentials for a file server

Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent or by writing your own user exit. Managed File Transfer provides a sample user exit that performs user credential mapping.

Related concepts

“The protocol bridge” on page 288

The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

Related tasks

"Mapping credentials for a file server by using the ProtocolBridgeCredentials.xml file" on page 297
Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent. Managed File Transfer provides an XML file that you can edit to include your credential information.

"Mapping credentials for a file server by using exit classes" on page 299
If you do not want to use the default credential mapping function of the protocol bridge agent, you can map user credentials in Managed File Transfer to user credentials on the file server by writing your own user exit. If you configure credential mapping user exits, they take the place of the default credential mapping function.

"Example: How to configure a protocol bridge agent to use private key credentials with a UNIX SFTP server" on page 301
This example demonstrates how you can generate and configure the `ProtocolBridgeCredentials.xml` file. This example is a typical example and the details might vary according to your platform, but the principles remain the same.

Related reference

ProtocolBridgeCredentialExit.java interface

Sample protocol bridge credential user exit

The MFT `agent.properties` file

Mapping credentials for a file server by using the `ProtocolBridgeCredentials.xml` file

Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent. Managed File Transfer provides an XML file that you can edit to include your credential information.

About this task

The `ProtocolBridgeCredentials.xml` file must be manually created by the user. By default, the location of this file is the home directory of the user who started the protocol bridge agent, but this can be stored anywhere on the file system accessible by the agent. To specify a different location, add the <credentialsFile> element to the `ProtocolBridgeProperties.xml` file. For example,

```
<tns:credentialsFile path="/example/path/to/ProtocolBridgeCredentials.xml"/>
```

Before you can use a protocol bridge agent, set up credential mapping by editing this file to include host, user, and credential information. For more information and samples, see Protocol bridge credentials file format.

Procedure

1.
   - Edit the line <tns:server name="*server name*"> to change the value of the name attribute to the server name in the `ProtocolBridgeProperties.xml` file.

     You can use the pattern attribute to specify that you used a server name that contains wildcards or regular expressions. For example,

     ```
     <tns:server name="serverA*" pattern="wildcard">
     ```

2. Insert user ID and credential information into the file as child elements of <tns:server>.

   You can insert one or many of the following elements into the file:

- If the protocol file server is an FTP, FTPS, or SFTP server, you can use passwords to authenticate the user requesting the transfer. Insert the following lines into the file:

```
<tns:user name="FTE User ID"
  serverUserId="Server User ID"
  serverPassword="Server Password">
</tns:user>
```

  Then change the value of the attributes.

  - name is a Java regular expression to match the MQMD user ID associated with the MFT transfer request
  - serverUserId is the value that is passed to the protocol file server as the login user ID. If the serverUserId attribute is not specified, the MQMD user ID associated with the MFT transfer request is used instead
  - serverPassword is the password that is associated with the serverUserId.

  The name attribute can contain a Java regular expression. The credential mapper attempts to match the MQMD user ID of the MFT transfer request to this regular expression. The protocol bridge agent attempts to match the MQMD user ID to the regular expression in the name attribute of the <tns:user> elements in the order that the elements exist in the file. When a match is found the protocol bridge agent does not look for more matches. If a match is found, the corresponding serverUserId and serverPassword values are passed to the protocol file server as the login user ID and password. The MQMD user ID matches are case-sensitive.

- If the protocol file server is an SFTP server, you can use public and private keys to authenticate the user requesting the transfer. Insert the following lines into the file and change the value of the attributes. The <tns:user> element can contain one or many <tns:privateKey> elements.

```
<tns:user name="FTE User ID"
    serverUserId="Server User ID"
    hostKey="Host Key">
    <tns:privateKey associationName="association"
      keyPassword="Private key password">
    Private key file text
    </tns:privateKey>
</tns:user>
```

  - name is a Java regular expression to match the MQMD user ID associated with the MFT transfer request
  - serverUserId is the value that is passed to the protocol file server as the login user ID. If the serverUserId attribute is not specified, the MQMD user ID associated with the MFT transfer request is used instead
  - hostKey is the expected key that is returned from the server when logging on
  - key is the private key of the serverUserId
  - keyPassword is the password for the key to generate public keys
  - associationName is a value that is used to identify for trace and logging purposes

  The name attribute can contain a Java regular expression. The credential mapper attempts to match the MQMD user ID of the MFT transfer request to this regular expression. The protocol bridge agent attempts to match the MQMD user ID to the regular expression in the name attribute of the <tns:user> elements in the order that the elements exist in the file. When a match is found the protocol bridge agent does not look for more matches. If a match is found, the corresponding serverUserId and key values are used to authenticate the MFT user with the protocol file server. The MQMD user ID matches are case-sensitive.

  For more information about using private keys with a protocol bridge agent, see “Example: How to configure a protocol bridge agent to use private key credentials with a UNIX SFTP server” on page 301.

Note: z/OS

When the transfer request is written to the command queue, the MQMD user ID might be converted to uppercase if the source agent command queue is on a z/OS or IBM i system. As a result the MQMD user ID for the same originating user might arrive at the credentials exit in the original case or converted to uppercase depending on the source agent that is specified in the transfer request. The default credential mapping exit performs case-sensitive matches against the supplied MQMD user ID, which you might need to allow for in the mapping file.

Related reference

Protocol bridge credentials file format

Protocol bridge properties file format

Regular expressions used by MFT

Mapping credentials for a file server by using exit classes

If you do not want to use the default credential mapping function of the protocol bridge agent, you can map user credentials in Managed File Transfer to user credentials on the file server by writing your own user exit. If you configure credential mapping user exits, they take the place of the default credential mapping function.

About this task

Managed File Transfer provides a sample user exit that performs user credential mapping. For more information, see "Using the sample protocol bridge credential user exit" on page 300.

A user exit for mapping protocol bridge credentials must implement one of the following interfaces:

- `com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit`, which allows a protocol bridge agent to transfer files to and from one default protocol file server
- `com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit2`, which allows you to transfer files to and from multiple endpoints.

The `com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit2` interface contains the same function as `com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit` and also includes extended function. For more information, see ProtocolBridgeCredentialExit.java interface and ProtocolBridgeCredentialExit2.java interface.

The credential exits can be chained together in a similar manner to other user exits. The exits are called in the order that they are specified in using the `protocolBridgeCredentialConfiguration` property in the agent properties file. The initialize methods all return separately and if one or more returns a value of false, the agent does not start. The error is reported in the agent event log.

Only one overall result is returned for the mapMQUserId methods of all of the exits as follows:

- If the method returns a value of USER_SUCCESSFULLY_MAPPED or USER_DENIED_ACCESS as the result code, this value is the returned result and the mapMQUserId methods of the subsequent exits are not called.
- If the method returns a value of NO_MAPPING_FOUND as the result code, the mqMQUserId method of the next exit is called.
- If there is no subsequent exit, the NO_MAPPING_FOUND result is returned.
- An overall result code of USER_DENIED_ACCESS or NO_MAPPING_FOUND is considered as a transfer failure by the bridge agent.

To run your exit, complete the following steps:

Procedure

1. Compile the protocol bridge credential user exit.
2. Create a Java archive (JAR) file that contains the compiled exit and its package structure.

3. Place the JAR file that contains the exit class in the `exits` directory of the bridge agent. The directory is in the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent_name* directory.
4. Edit the property file of the protocol bridge agent to include the property `protocolBridgeCredentialExitClasses`. For the value of this property, specify a comma-separated list of classes that implement a protocol bridge credential exit routine. The exit classes are called in the order that they are specified in this list. For more information, see The MFT `agent.properties` file.
5. Edit the property file of the protocol bridge agent to include:

```
exitClassPath=IBM MQ
installation_directory\mqft\config\configuration_queue_manager\agents\protocol_bridge_agent_n
ame\exits\SampleCredentialExit.jar
```

   The `agent.properties` file for an agent is in your *MQ_DATA_PATH*/mqft/config/*coordination_qmgr_name*/agents/*bridge_agent_name* directory.

   If you change the `agent.properties` file, you must restart the agent to pick up the changes.
6. You can optionally specify the `protocolBridgeCredentialConfiguration` property. The value that you specify for this property is passed in as a String object to the initialize() method of the exit classes specified by `protocolBridgeCredentialExitClasses`. For more information, see The MFT `agent.properties` file.
7. Start the protocol bridge agent with the **fteStartAgent** command.

Using the sample protocol bridge credential user exit
Managed File Transfer provides a sample user exit that performs user credential mapping.

About this task

A sample protocol bridge credential exit is provided in the *MQ_INSTALLATION_PATH*/mqft/samples/protocolBridge directory and in the topic Sample protocol bridge credential user exit. This sample is based on the `com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit` interface.

The `SampleCredentialExit.java` exit reads a properties file that maps the MQMD user IDs associated with transfer requests to server user IDs and server passwords. The location of the properties file is taken from the protocol bridge agent property `protocolBridgeCredentialConfiguration`.

To run the sample user exit, complete the following steps:

Procedure

1. Compile the `SampleCredentialExit.java` file.
2. Create a JAR file that contains the compiled exit and its package structure.
3. Place the JAR file in the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent_name*/exits directory.
4. Edit the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*bridge_agent_name*/agent.properties file to contain the line:

```
protocolBridgeCredentialExitClasses=SampleCredentialExit
```

5. Edit the property file of the protocol bridge agent to include:

```
exitClassPath=IBM MQ
installation_directory\mqft\config\configuration_queue_manager\agents\protocol_bridge_agent_n
ame\exits\SampleCredentialExit.jar
```

   The `agent.properties` file for an agent is in your *MQ_DATA_PATH*/mqft/config/*coordination_qmgr_name*/agents/*agent_name* directory.

   If you change the `agent.properties` file, you must restart the agent to pick up the changes.

6. Create a credential properties file (`credentials.properties`) in the directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*bridge_agent* and edit it to include entries in the format:

```
mqUserId=serverUserId,serverPassword
```

7. Edit the *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*bridge_agent_name*`/agent.properties` file to contain the line:

```
protocolBridgeCredentialConfiguration=MQ_DATA_PATH/mqft/
config/coordination_queue_manager/agents/bridge_agent_name/credentials.properties
```

   You must use the absolute path to the `credentials.properties` file.

8. Start the protocol bridge agent by using the **fteStartAgent** command.

Related concepts

"The protocol bridge" on page 288

The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

Related reference

ProtocolBridgeCredentialExit.java interface

ProtocolBridgeCredentialExit2.java interface

Sample protocol bridge credential user exit

The MFT `agent.properties` file

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

Example: How to configure a protocol bridge agent to use private key credentials with a UNIX SFTP server

This example demonstrates how you can generate and configure the `ProtocolBridgeCredentials.xml` file. This example is a typical example and the details might vary according to your platform, but the principles remain the same.

Procedure

1. Generate a public and private key to be used to authenticate with the SFTP server.

   For example, on a Linux host system, you can use the tool **ssh-keygen**, supplied as part of the 'openssh' package, to create the public/private key pair.

   By default, with no arguments, the **ssh-keygen** command prompts you for a location and passphrase for the two key files, which defaults to the names:

```
id_rsa      <-- Private key
id_rsa.pub  <-- Public key
```

   Attention: If you are using the **ssh-keygen** command from a recent version of OpenSSH, such as that supplied with RHEL 8, the key format that is used is not compatible with the protocol bridge agent, and transfer attempts to the SFTP server fail with the message:

```
BFGBR0216E: Authentication to protocol server 'sftp.host.address' failed because of
invalid private key.
```

   To create a compatible RSA private key with these newer versions of OpenSSH, specify the key format with the `-m PEM` option to the **ssh-keygen** command:

```
ssh-keygen -m PEM
```

The contents of the `id_rsa` private key then have the first and last lines of:

```
-----BEGIN RSA PRIVATE KEY-----
... ... ...
-----END RSA PRIVATE KEY-----
```

V 10.0.0 To generate a private key in OpenSSH format, run the **ssh-keygen** command without specifying the `-m PEM` option:

```
ssh-keygen
```

The contents of the private key will then have the first and last lines as follows:

```
-----BEGIN OPENSSH PRIVATE KEY----
… … …
-----END OPENSSH PRIVATE KEY-----
```

Both of these formats are compatible with the protocol bridge agent. Ensure that you choose the appropriate format based on your requirements.

2. Copy the entire contents of the `id_rsa.pub` file into the `~/.ssh/authorized_keys` file of the SFTP user on the SFTP server.

   Ensure that the file permissions on this file and the `~/.ssh` directory are set appropriately for the SFTP server to allow key authentication. These permissions are typically:

```
~/.ssh                    Mode 700
~/.ssh/authorized_keys    Mode 600
```

3. Managed File Transfer requires a host ssh fingerprint that is generated using the MD5 algorithm. Run one of the following commands to obtain the host ssh fingerprint of the SFTP server.

   - For Red Hat® Enterprise Linux version 6.x and earlier, and Linux Ubuntu 14.04, run the following command:

     ```
     ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
     ```

   - Starting with Red Hat Enterprise Linux 7.x, Linux Ubuntu 16.04 and SUSE Linux 12.4, the **ssh-keygen** command generates, by default, the ssh fingerprint using the SHA256 algorithm. To generate the ssh fingerprint using the MD5 algorithm, run the following command:

     ```
     ssh-keygen -l -E MD5 -f /etc/ssh/ssh_host_rsa_key.pub
     ```

     The output of the command will be similar to the following example:

     ```
     2048 MD5:64:39:f5:49:41:10:55:d2:0b:81:42:5c:87:62:9d:27 no comment (RSA)
     ```

     Extract the hexadecimal portion only of the output to use as the hostKey in the `ProtocolBridgeCredentials.xml` file (see Step “4” on page 302). Therefore, in this example, you would extract `64:39:f5:49:41:10:55:d2:0b:81:42:5c:87:62:9d:27`.

4. On the protocol bridge agent system, edit the `ProtocolBridgeCredentials.xml` file as shown in the following example. Substitute the values shown in italics with your own values:

```
<tns:credentials xmlns:tns="http://wmqfte.ibm.com/ProtocolBridgeCredentials"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wmqfte.ibm.com/ProtocolBridgeCredentials
 ProtocolBridgeCredentials.xsd ">

<tns:agent name="Agent_name">

<tns:server name="SFTP_name">

<tns:user name="mq_User_ID" serverUserId="SFTP_user_ID"
  hostKey="ssh_host_finger">
<tns:privateKey associationName="name" keyPassword="pass_phrase">
 Complete contents of the id_rsa file including the entries
 -----BEGIN RSA PRIVATE KEY-----
```

```
 -----END RSA PRIVATE KEY-----
</tns:privateKey>
</tns:user>

</tns:server>
</tns:agent>
</tns:credentials>
```

where:

- *Agent_name* is the name of the protocol bridge agent.
- *SFTP_host_name* is the name of the SFTP server as shown in the `ProtocolBridgeProperties.xml` file.
- *mq_User_ID* is the MQMD user ID associated with the transfer request.
- *SFTP_user_ID* is the SFTP user ID as used in Step "2" on page 302. It is the value that is passed to the SFTP serve as the login user ID.
- *ssh_host_finger* is the fingerprint that is collected in Step "3" on page 302.
- *name* is a name that you can specify to be used for trace and logging purposes.
- *pass_phrase* is the pass phrase that you provided in the **ssh-keygen** in Step "1" on page 301.
- *Complete contents of the id_rsa file* is the complete contents of the generated `id_rsa` file from Step "1" on page 301. To avoid connection errors, ensure that your key includes the appropriate header and footer based on its type:

  For RSA keys, include:

  ```
  -----BEGIN RSA PRIVATE KEY-----

  -----END RSA PRIVATE KEY-----
  ```

  V 10.0.0 For OpenSSH keys, include:

  ```
  -----BEGIN OPENSSH PRIVATE KEY-----
  -----END OPENSSH PRIVATE KEY-----
  ```

  You can add more keys by duplicating the `<tns:privatekey>` element.

5. If the agent is not already started, start the protocol bridge agent. Alternatively, the protocol bridge agent periodically polls the `ProtocolBridgeCredentials.xml` file and picks up the changes.

Related reference

Protocol bridge credentials file format

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

The MFT `agent.properties` file

"Mapping credentials for a file server" on page 296

Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent or by writing your own user exit. Managed File Transfer provides a sample user exit that performs user credential mapping.

Configuring a protocol bridge for an FTPS server

Configure an FTPS server in a similar way as you configure an FTP server: create a bridge agent for the server, define the server properties, and map user credentials.

About this task

To configure an FTPS server, complete the following steps:

Procedure

1. Create a protocol bridge agent for the FTPS server by using the **fteCreateBridgeAgent** command. The parameters that are applicable to FTP are also applicable to FTPS but there are also three required parameters specific to FTPS:
   a) The **-bt** parameter. Specify FTPS as the value of this parameter.
   b) The **-bts** parameter for the truststore file. The command assumes that only server authentication is required and you must specify the location of the truststore file.

   The explicit form of the FTPS protocol is configured by the **fteCreateBridgeAgent** command by default but you can configure the implicit form by changing the protocol bridge properties file. The protocol bridge always connects to FTPS servers in passive mode.

   For more information about the **fteCreateBridgeAgent** command, see fteCreateBridgeAgent (create and configure an MFT protocol bridge agent).

   If you need instructions about how to create truststore files see the information about the keytool at the Oracle keytool documentation.
2. Define the FTPS server properties within an <ftpsServer> element in the protocol bridge properties file: `ProtocolBridgeProperties.xml`. For more information, see "Defining properties for protocol file servers using the ProtocolBridgeProperties.xml file" on page 290. You can also enable client authentication by editing the protocol bridge properties file. For details of all the configuration options, see Protocol bridge properties file format.
3. Map user credentials in Managed File Transfer to user credentials on the FTPS server either by using the default credential mapping function of the protocol bridge agent or by writing your own user exit. For more information, see "Mapping credentials for a file server" on page 296.
4. By default, the truststore file is configured as having the JKS format; if you want to change the format, edit the protocol bridge properties file.

Example

An example entry for an FTPS server in the protocol bridge properties file is shown as follows:

```
<tns:serverProperties xmlns:tns="http://wmqfte.ibm.com/ProtocolBridgeProperties"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://wmqfte.ibm.com/ProtocolBridgeProperties
       ProtocolBridgeProperties.xsd">
    <tns:defaultServer name="ftpsserver.mycompany.com" />

    <tns:ftpsServer name="ftpsserver.mycompany.com" host="ftpsserver.mycompany.com" port="990"
platform="windows"
       timeZone="Europe/London" locale="en_US" fileEncoding="UTF8"
       listFormat="unix" limitedWrite="false"
       trustStore="c:\mydirec\truststore.jks" />

    <!-- Define servers here -->
</tns:serverProperties>
```

What to do next

For information about the parts of the FTPS protocol that are supported and, which are not supported, see FTPS server support by the protocol bridge.

Related concepts

"The protocol bridge" on page 288
The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

Related tasks

"Mapping credentials for a file server by using the ProtocolBridgeCredentials.xml file" on page 297
Map user credentials in Managed File Transfer to user credentials on the file server by using the default credential mapping function of the protocol bridge agent. Managed File Transfer provides an XML file that you can edit to include your credential information.

"Defining properties for protocol file servers using the ProtocolBridgeProperties.xml file" on page 290
Define the properties of one or more protocol file servers that you want to transfer files to and from using the `ProtocolBridgeProperties.xml` file, which is provided by Managed File Transfer in the agent configuration directory.

Related reference

fteCreateBridgeAgent (create and configure an MFT protocol bridge agent)

Protocol bridge credentials file format

Protocol bridge properties file format

FTPS server support by the protocol bridge

Scenarios and examples for limiting the number of file transfers to individual file servers

How the revised protocol bridge agent works with the **maxActiveDestinationTransfers** and **failTransferWhenCapacityReached** attributes, together with some examples.

Scenarios showing the working of the protocol bridge agent based on the `maxActiveDestinationTransfers` value

Scenario 1

The `ProtocolBridgeProperties.xml` file for a protocol bridge agent contains two file server definitions:

- You have not set the global **maxActiveDestinationTransfers** attribute.
- You have not set the **maxActiveDestinationTransfers** attribute on both `fileServerA` and `FileServerB`.
- You have set the protocol bridge agent **maxDestinationTransfers** attribute to the default value.

If you have set the protocol bridge agent **maxDestinationTransfers** attribute to the default value of 25, then:

- The destination agent starts processing two managed transfers to `fileServerA`.
- Both the transfers complete.

  At this point of time, the client realizes that `fileServerA` has failed, and sets the following values for `fileServerA` in the `ProtocolBridgeProperties.xml` file:

  maxActiveDestinationTransfers = 0
  failTransferWhenCapacityReached =true
- Another transfer arrives for `fileServerA` and a few for `fileServerB`:

  Based on the properties set in the previous step, the managed transfer to `fileServerA` is rejected and marked as failed, whereas the transfers for `fileServerB` are handled in the standard existing flow.
- After some time, the client finds out that the `fileServerA` is running again, so the client removes or comments out the previously added value in the `ProtocolBridgeProperties.xml`. A new managed transfer arrives for `fileServerA` and is handled in the standard existing flow.

Scenario 2

- You have set the **maxActiveDestinationTransfers** attribute for a file server and not set the **failTransferWhenCapacityReached** attribute.
- The protocol bridge agent is acting as the destination agent for this number of managed transfers to the file server.
- The value of the **maxActiveDestinationTransfers** attribute is reduced by 1.

The protocol bridge agent dynamically updates its configuration and sets **maxActiveDestinationTransfers** to the new value while it is still active. The in-progress managed transfers are not affected by this update and are allowed to complete.

Scenario 3

The ProtocolBridgeProperties.xml file for a protocol bridge agent contains two file server definitions:

- You have not set the global **maxActiveDestinationTransfers** attribute.
- You have not set the **failTransferWhenCapacityReached** attribute.
- You have set the **maxActiveDestinationTransfers** to 1 on `fileServerA`.
- You have not set the **maxActiveDestinationTransfers** attribute on `fileServerB`.

If the protocol bridge agent has the **maxDestinationTransfers** attribute set to 5:

- The maximum number of active destination transfers from the protocol bridge agent to `fileServerA` is 1 (although the destination agent has 5 destination transfer slots, only 1 can be used for managed transfers to `fileServerA`).

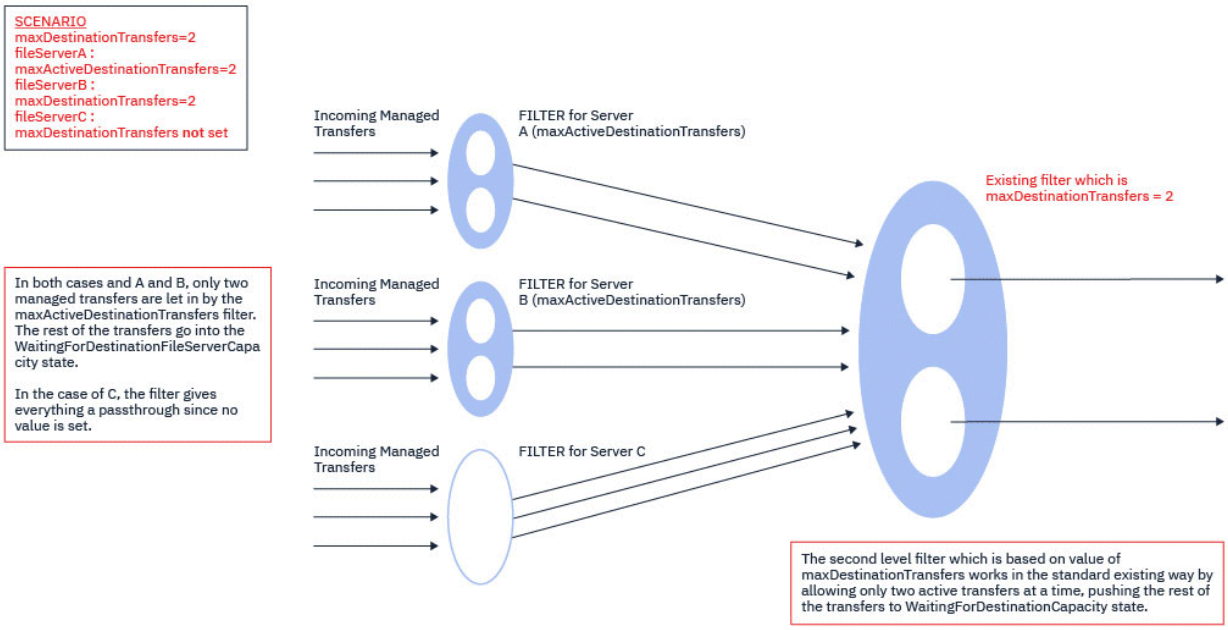
  This is useful when `fileServerA` fails. Once `fileServerA` is running again, the value of **maxActiveDestinationTransfers** can be increased to 5 to permit full capacity of the allowed destination transfers.
- The maximum number of active destination transfers from the protocol bridge agent to `fileServerB` is 5.

  As **maxActiveDestinationTransfers** is not set for this file server, the protocol bridge agent can use all 5 of its destination transfers slots for managed transfers to it.

Scenario 4

In the following diagram:

- You have set the **maxDestinationTransfers** attribute to 2 in the `agent.properties` file.
- You have set the **maxActiveDestinationTransfers** to 2 on `fileServerA`.
- You have set the **maxActiveDestinationTransfers** attribute to 2 on `fileServerB`.
- You have not set the **maxActiveDestinationTransfers** attribute on `fileServerC`.



As the diagram shows, the **maxActiveDestinationTransfers** and **maxDestinationTransfers** attributes are independent of each other.

The values for **maxActiveDestinationTransfers** for each of the servers is checked. Based on this value, the transfers are either allowed to continue further, or pushed to the **WaitingForDestinationFileServerCapacity** state.

The transfers that are allowed then pass through the existing standard flow of check against the **maxDestinationTransfers**.

Scenario 5

Attention: You should be careful when setting the values of the **maxActiveDestinationTransfers** attributes as you must keep the value of the **maxDestinationTransfers** attribute in mind.

If you do not do this, a situation as described in the following text can occur:

- You have not set a value for the global **maxActiveDestinationTransfers** attribute.
- You have set a value of **maxDestinationTransfers**=2 in the `agent.properties` file.
- You have set a value of **maxActiveDestinationTransfers**=2 on `fileServerA`.
- You have not set a value for **maxActiveDestinationTransfers** on `fileServerB`.

Suppose that the following sequence of events occurs:

- The protocol bridge agent receives a request to transfer a file to `fileServerA`. The protocol bridge agent is not currently doing anything, so it accepts this managed transfer request.

  The transfer slots now look like this:

  - Destination Transfers: 1
  - Destination Transfers for `fileServerA`: 1
  - Destination transfers for `fileServerB`: 0
- Now, the protocol bridge agent receives another request to act as the destination agent for a managed transfer involving `fileServerA`. Once again, it accepts this request, and so the transfer slots look like this:

  - Destination Transfers: 2

- Destination Transfers for `fileServerA`: 2
- Destination transfers for `fileServerB`: 0

The two `Destination Transfer` slots in the agent are now occupied, and so the agent cannot participate in any more managed transfers until one of the transfers to `fileServerA` has finished.

- A short time later `fileServerA` fails, which causes the two managed transfers to go into recovery. The `Destination transfer` slots that these managed transfers are using remain in use during this time.
- Next, the protocol bridge agent receives a request to transfer a file to `fileServerB`. There is a space for this transfer in the `Destination Transfers for fileServerB` slots, however, all of the `Destination Transfer` slots for the agent are being used, and so the transfer is put onto the backlog so that it can be retried later.

As a result, the transfer to `fileServerB` is blocked until at least one of the transfers to `fileServerA` has completed and released its `Destination Transfer` slot.

To prevent this situation happening:

- Either set the value of **maxActiveDestinationTransfers** on the file servers to be less than the **maxDestinationTransfers** value, so that free slots remain.
- Or evenly distribute the value of the **maxActiveDestinationTransfers** attribute among all the endpoint servers.

Behavior of the protocol bridge agent based on the values of the `maxActiveDestinationTransfers` attribute

Note: In all the error cases listed in the following table, if the **maxActiveDestinationTransfers** attribute is set to a value that is not valid the protocol bridge agent assumes that this attribute is not set.

maxActiveDestinationTransfers	Sample value	Description
Not specified	Not specified	The transfers go as usual. There is no limit placed on the number of transfers for *ftp* endpoint.
Specified	0	No transfers allowed to this specific *ftp* endpoint.
Negative value	-1	Error logged in output0.log Value -1 is not valid for a non negative integer. The protocol bridge agent assumes that the attribute is not set.
Non integer value	abc	Error logged in output0.log Value abc is not valid for an integer. The protocol bridge agent assumes that the attribute is not set.
Empty	""	The value '' of the attribute maxActiveDestinationTransfers is not valid for a non negative integer.
Specified	5	Allows only five active transfers to be running at any point of time for this *ftp* endpoint. The excessive transfers are retried or rejected, based on the value of the failTransferWhenCapacityReached attribute.

Behavior of the protocol bridge agent for the combination of the `maxActiveDestinationTransfers` and `failTransferWhenCapacityReached` attributes

failTransferWhenCapacityReached value	maxActiveDestinationTransfers value	Result
False	3	Three active transfers are allowed to this endpoint server. Any more transfers are retried.
True	3	Three active transfers are allowed to this endpoint server. Any more transfers are rejected and marked as failed.
Not specified	3	Default value **false** for failTransferWhenCapacityReached is considered. The result is that three active transfers are allowed to this endpoint server. Any more transfers are retried.
Values other than Boolean value	Specified	Error logged in output.log. The value specified for failTransferWhenCapacityReached is not a Boolean value. The default value for failTransferWhenCapacityReached is considered.

Behavior of the protocol bridge agent for the combination of the `maxDestinationTransfers` and `failTransferWhenCapacityReached` attributes

failTransferWhenCapacityReached value	maxDestinationTransfers value	Result
True	10	When the number of concurrent active transfers reaches 10, the 11th managed transfer is failed by the protocol bridge agent.
False	10	Existing behavior. When the number of concurrent active transfers reaches 10, the 11th managed transfer is queued waiting for a slot to be freed.
Not specified	10	Existing behavior

Error messages

Existing message:

BFGS0082I
Is logged in the output0.log file of the source agent when the protocol bridge agent rejects the transfer, when the protocol bridge agent is already running the maximum number of transfers defined in the **maxDestinationTransfers** attribute.

New messages:

BFGSS0085I
Is logged in the output0.log file of the source agent when the protocol bridge agent rejects and retries a managed transfer,

BFGSS0086I
Is logged in the output0.log file of the source agent when the protocol bridge agent rejects and retries a managed transfer, and the destination item does not include the file server name

BFGSS0084E
Is logged in the Explorer and audit.xml file when the protocol bridge agent rejects, for exceeding the maximum number of concurrent transfers specified in the **maxActiveDestinationTransfers** attribute, and marks a managed transfer as failed.

BFGSS0087E
Is logged in the Explorer and audit.xml file when the protocol bridge agent rejects, for exceeding the maximum number of destination transfers specified in the **maxActiveDestinationTransfers** attribute, and marks a managed transfer as failed.

BFGSS0088W
Is logged in the output0.log, when the value of the **maxActiveDestinationTransfers** attribute exceeds the value of the **maxDestinationTransfers** attribute.

BFGSS0089I
Is logged in the output0.log file of the destination protocol bridge agent, when it is working a source agent that is not at IBM MQ 9.3.0, or later.

Related concepts

“The protocol bridge” on page 288
The protocol bridge enables your Managed File Transfer (MFT) network to access files stored on a file server outside your MFT network, either in your local domain or a remote location. This file server can use the FTP, FTPS, or SFTP network protocols. Each file server needs at least one dedicated agent. The dedicated agent is known as the protocol bridge agent. A bridge agent can interact with multiple file servers.

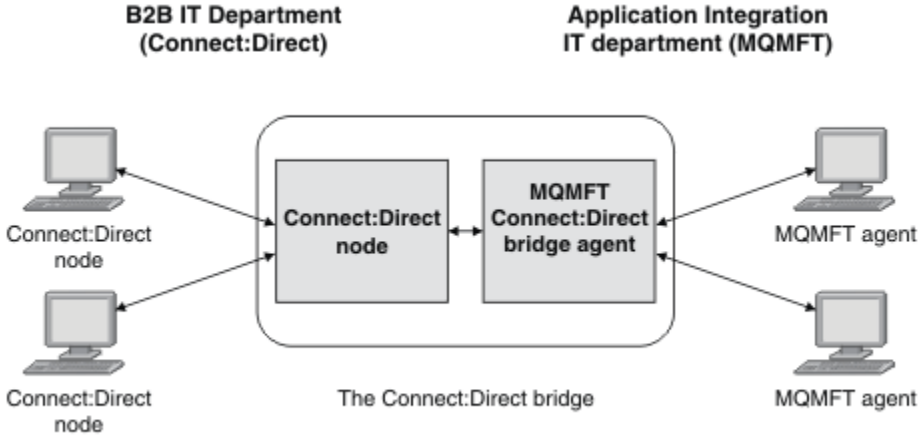
Related tasks

“Defining properties for protocol file servers using the ProtocolBridgeProperties.xml file” on page 290

Define the properties of one or more protocol file servers that you want to transfer files to and from using the `ProtocolBridgeProperties.xml` file, which is provided by Managed File Transfer in the agent configuration directory.

The Connect:Direct bridge

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.



The diagram shows an MFT Connect:Direct bridge between two departments, the B2B IT department and the Application Integration IT department. The B2B IT department uses Connect:Direct to transfer files to and from the company's business partners. The Application Integration IT department uses IBM MQ as its messaging infrastructure and so has recently chosen Managed File Transfer as its file transfer solution.

Using the MFT Connect:Direct bridge, the two departments can transfer files between the Connect:Direct network in the B2B IT department and the MFT network in the Application Integration IT department. The Connect:Direct bridge is a component of Managed File Transfer, which includes an MFT agent that communicates with a Connect:Direct node. The MFT agent is dedicated to transfers with the Connect:Direct node, and is known as the Connect:Direct bridge agent.

The Connect:Direct bridge is available as part of the Service and Agent components of Managed File Transfer, and can be used for the following tasks:

1. Use Managed File Transfer commands to initiate a transfer of a file, or multiple files, from an MFT agent to a Connect:Direct node.
2. Use Managed File Transfer commands to initiate a transfer of a file, or multiple files, from a Connect:Direct node to an MFT agent.
3. Use Managed File Transfer commands to initiate a file transfer that starts a user-defined Connect:Direct process.
4. Use Connect:Direct process to submit an MFT file transfer request.

A Connect:Direct bridge can transfer files to or from only Connect:Direct nodes. The Connect:Direct bridge can transfer files to or from its local file system only as part of a transfer submitted by a Connect:Direct process.

z/OS You can use the Connect:Direct bridge to transfer to or from a data set that is located on a Connect:Direct node on a z/OS system. There are some differences in behavior compared to data set transfers that only involve Managed File Transfer agents. For more information, see z/OS Transferring data sets to and from Connect:Direct nodes.

Supported platforms

The Connect:Direct bridge is made up of an MFT Connect:Direct bridge agent and a Connect:Direct node. The agent is supported on Windows and Linux for x86-64. The node is supported on the platforms that are supported for IBM Sterling Connect:Direct for Windows and IBM Sterling Connect:Direct for UNIX. For instructions on creating the Connect:Direct bridge agent and configuring a Connect:Direct node for the agent to communicate with, see Configuring the Connect:Direct bridge.

The Connect:Direct bridge can transfer files to and from Connect:Direct nodes that are running as part of a Connect:Direct for Windows or Connect:Direct for UNIX , or Connect:Direct for z/OS Service installation. For details of the versions of Connect:Direct that are supported, see the web page System Requirements for IBM MQ.

The agent and node that make up the Connect:Direct bridge must be on the same system, or have access to the same file system, for example through a shared NFS mount. This file system is used to temporarily store files during file transfers that involve the Connect:Direct bridge, in a directory defined by the **cdTmpDir** parameter. The Connect:Direct bridge agent and the Connect:Direct bridge node must be able to address this directory using the same path name. For example, if the agent and node are on separate Windows systems, the systems must use the same drive letter to mount the shared file system. The following configurations allow the agent and the node to use the same path name:

- The agent and node are on the same system, which is either running Windows or Linux for x86-64
- The agent is on Linux for x86-64, and the node is on AIX
- The agent is on one Windows system, and the node is on another Windows system

The following configurations do not allow the agent and the node to use the same path name:

- The agent is on Linux for x86-64, and the node is on Windows
- The agent is on Windows, and the node is on UNIX

Consider this restriction when planning your installation of the Connect:Direct bridge.

Related concepts

“Recovery and restart for transfers to and from Connect:Direct nodes” on page 319
Managed File Transfer might be unable to connect to your IBM Sterling Connect:Direct node during a transfer; for example, if the node becomes unavailable. Either Managed File Transfer attempts to recover the transfer, or the transfer fails and an error message is produced.

“Submitting a user-defined Connect:Direct process from a file transfer request” on page 320
You can submit a transfer request for a transfer that goes through the Connect:Direct bridge agent that calls a user-defined Connect:Direct process as part of the file transfer.

“Using Connect:Direct processes to submit Managed File Transfer transfer requests” on page 324
You can submit a transfer request to the Connect:Direct bridge agent from a Connect:Direct process. Managed File Transfer provides commands that can be called from a **RUN TASK** statement in a Connect:Direct process.

Related tasks

Configuring the Connect:Direct bridge

“Transferring a file to a Connect:Direct node” on page 313
You can transfer a file from a Managed File Transfer agent to a Connect:Direct node using the Connect:Direct bridge. Specify a Connect:Direct node as the destination of the transfer by specifying the Connect:Direct bridge agent as the destination agent and specifying the destination file in the form *connect_direct_node_name*:*file_path*.

“Transferring a file from a Connect:Direct node” on page 314
You can transfer a file from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the transfer by specifying the Connect:Direct bridge agent as the source agent and specifying the source specification in the form *connect_direct_node_name*:*file_path*.

“Transferring multiple files to a Connect:Direct node” on page 316

You can transfer multiple files from a Managed File Transfer Agent to a Connect:Direct node by using the Connect:Direct bridge. To use a Connect:Direct node as the destination of the multiple file transfer, specify the Connect:Direct bridge agent as the destination agent and specify the destination directory in the form *connect_direct_node_name*:*directory_path*.

"Transferring multiple files from a Connect:Direct node" on page 317
You can transfer multiple files from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the multiple file transfer by specifying the Connect:Direct bridge agent as the source agent and specifying one or more source specifications in the form *connect_direct_node_name*:*file_path*.

"Transferring multiple files to Connect:Direct by using wildcards" on page 318
To transfer multiple files from a Managed File Transfer agent to a Connect:Direct node, use the Connect:Direct bridge. You can use wildcard characters in the source specification that you provide to the **fteCreateTransfer** command. As with all Managed File Transfer transfers involving wildcards, only the last part of the file path can contain a wildcard character. For example, `/abc/def*` is a valid file path and `/abc*/def` is not valid.

Troubleshooting the Connect:Direct bridge

Related reference

fteCreateCDAgent: create a Connect:Direct bridge agent

Restrictions of the Connect:Direct bridge agent

Transferring a file to a Connect:Direct node

You can transfer a file from a Managed File Transfer agent to a Connect:Direct node using the Connect:Direct bridge. Specify a Connect:Direct node as the destination of the transfer by specifying the Connect:Direct bridge agent as the destination agent and specifying the destination file in the form *connect_direct_node_name*:*file_path*.

Before you begin

Before transferring a file, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. For more information, see Configuring the Connect:Direct bridge.

About this task

In this example, the Connect:Direct bridge agent is called CD_BRIDGE. The source agent is called FTE_AGENT and can be any version of WMQFTE. The destination Connect:Direct node is called CD_NODE1. The file to be transferred is located at the file path `/home/helen/file.log` on the system where FTE_AGENT is located. The file is transferred to the file path `/files/data.log` on the system where CD_NODE1 is running.

Procedure

1. Use the fteCreateTransfer command with the value for the **-df** (destination file) parameter in the form *connect_direct_node_name*:*file_path* and the value of the **-da** (destination agent) parameter specified as the name of the Connect:Direct bridge agent.

   Note: The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the file to be transferred to, not the Connect:Direct node that operates as part of the Connect:Direct bridge.

   ```
   fteCreateTransfer -sa FTE_AGENT -da CD_BRIDGE
     -df CD_NODE1:/files/data.log /home/helen/file.log
   ```

   For more information, see **fteCreateTransfer**: start a new file transfer.
2. The source agent FTE_AGENT transfers the file to the Connect:Direct bridge agent CD_BRIDGE. The file is temporarily stored on the system where the Connect:Direct bridge agent is running, in the location defined by the cdTmpDir agent property. The Connect:Direct bridge agent transfers the file to the Connect:Direct node CD_NODE1.

Related concepts

"The Connect:Direct bridge" on page 311

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related tasks

"Transferring a file from a Connect:Direct node" on page 314

You can transfer a file from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the transfer by specifying the Connect:Direct bridge agent as the source agent and specifying the source specification in the form *connect_direct_node_name*:*file_path*.

Related reference

The MFT `agent.properties` file

Transferring a file from a Connect:Direct node

You can transfer a file from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the transfer by specifying the Connect:Direct bridge agent as the source agent and specifying the source specification in the form *connect_direct_node_name*:*file_path*.

Before you begin

Before transferring a file, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. See Configuring the Connect:Direct bridge.

About this task

In this example, the Connect:Direct bridge agent is called CD_BRIDGE. The destination agent is called FTE_AGENT and can be any version of Managed File Transfer. The source Connect:Direct node is called CD_NODE1. The file to be transferred is located at the file path `/home/brian/in.file` on the system where CD_NODE1 is located. The file is transferred to the file path `/files/out.file` on the system where FTE_AGENT is running.

Procedure

Use the **fteCreateTransfer** command with the value for the source specification in the form *connect_direct_node_name*:*file_path* and the value of the **-sa** parameter specified as the name of the Connect:Direct bridge agent.

Note: The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the file to be transferred from, not the Connect:Direct node that operates as part of the Connect:Direct bridge. For example:

```
fteCreateTransfer -sa CD_BRIDGE -da FTE_AGENT
                  -df /files/out.file CD_NODE1:/home/brian/in.file
```

For more information, see **fteCreateTransfer**: start a new file transfer.

Results

The Connect:Direct bridge agent CD_BRIDGE requests the file from the Connect:Direct node CD_NODE1. The Connect:Direct node sends the file to the Connect:Direct bridge. While the file is being transferred from the Connect:Direct node, the Connect:Direct bridge stores the file temporarily in the location defined by the `cdTmpDir` agent property. When the file has finished transferring from the Connect:Direct node to the Connect:Direct bridge, the Connect:Direct bridge then sends the file to the destination agent FTE_AGENT and deletes the file from the temporary location.

Related concepts

"The Connect:Direct bridge" on page 311

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related reference

The MFT `agent.properties` file

z/OS Transferring a data set to a Connect:Direct node on z/OS

You can transfer a data set from a Managed File Transfer agent on z/OS to a Connect:Direct node on z/OS by using a Connect:Direct bridge that is located on a Windows or Linux system.

Before you begin

Before transferring a file, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. See Configuring the Connect:Direct bridge.

About this task

In this example, the parameter **-df** is used to specify the destination of the transfer. The parameter **-df** is valid for use when the source agent of the transfer is any version of Managed File Transfer. You can use the **-ds** parameter instead. The source agent is called FTE_ZOS1 and is an Managed File Transfer agent. The Connect:Direct bridge agent is called CD_BRIDGE and is located on a Linux system. The destination Connect:Direct node is called CD_ZOS2. Both the source agent and the destination Connect:Direct node are located on z/OS systems. The data set to be transferred is located at `//FTEUSER.SOURCE.LIB` on the system where FTE_ZOS1 is located. The data set is transferred to the data set `//CDUSER.DEST.LIB` on the system where CD_ZOS2 is located.

Procedure

1. Use the fteCreateTransfer command with the value for the **-df** parameter in the form: *connect_direct_node_name*:*data_set_name*;*attributes* and the value of the **-da** (destination agent) parameter specified as the name of the Connect:Direct bridge agent.

   The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the data set to be transferred to, not the Connect:Direct node that operates as part of the Connect:Direct bridge.

   The data set name specified by *data_set_name* must be absolute, not relative. Connect:Direct does not prefix the data set name with the name of the user.

   ```
   fteCreateTransfer -sa FTE_ZOS1 -sm QM_ZOS
                     -da CD_BRIDGE -dm QM_BRIDGE
                     -df CD_ZOS2://'CDUSER.DEST.LIB;BLKSIZE(8000);LRECL(80)'
                     //'FTEUSER.SOURCE.LIB'
   ```

   For more information, see **fteCreateTransfer**: start a new file transfer.
2. The source agent FTE_ZOS1 transfers the data in the data set to the Connect:Direct bridge agent CD_BRIDGE. The data is temporarily stored as a flat file on the system where the Connect:Direct bridge agent is running, in the location defined by the cdTmpDir agent property. The Connect:Direct bridge agent transfers the data to the Connect:Direct node CD_ZOS2. When the transfer is complete, the flat file is deleted from the system where the Connect:Direct bridge agent is running.

Related concepts

"The Connect:Direct bridge" on page 311

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related tasks

Transferring data sets to and from Connect:Direct nodes

Related reference

BPXWDYN properties you must not use with MFT

Transferring multiple files to a Connect:Direct node

You can transfer multiple files from a Managed File Transfer Agent to a Connect:Direct node by using the Connect:Direct bridge. To use a Connect:Direct node as the destination of the multiple file transfer, specify the Connect:Direct bridge agent as the destination agent and specify the destination directory in the form *connect_direct_node_name*:*directory_path*.

Before you begin

Before transferring files, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. See Configuring the Connect:Direct bridge.

About this task

In this example, the source agent is called FTE_AGENT. The Connect:Direct bridge agent is called CD_BRIDGE. The destination Connect:Direct node is called CD_NODE1. The files to be transferred are `/home/jack/data.log`, `/logs/log1.txt`, and `/results/latest` on the system where FTE_AGENT is located. The files are transferred to the directory `/in/files` on the system where CD_NODE1 is running.

Procedure

Use the fteCreateTransfer command with the value for the **-dd** (destination directory) parameter in the form *connect_direct_node_name*:*directory_path*. Specify the value of the **-da** (destination agent) parameter as the name of the Connect:Direct bridge agent.

Note: The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the files to be transferred to, not the Connect:Direct node that operates as part of the Connect:Direct bridge.

```
fteCreateTransfer -sa FTE_AGENT -da CD_BRIDGE
                  -dd CD_NODE1:/in/files /home/jack/data.log
                  /logs/log1.txt /results/latest
```

For more information, see **fteCreateTransfer**: start a new file transfer.

Results

The source agent FTE_AGENT transfers the first file to the Connect:Direct bridge agent CD_BRIDGE. The Connect:Direct bridge agent temporarily stores the file in the location defined by the cdTmpDir property. When the file has been completely transferred from the source agent to the Connect:Direct bridge, the Connect:Direct bridge agent sends the file to the Connect:Direct node that is defined by the cdNode agent property. This node sends the file to the destination Connect:Direct node CD_NODE1. The Connect:Direct bridge agent deletes the file from the temporary location when the transfer between the two Connect:Direct nodes is complete. This process is repeated for each specified source file.

Related concepts

"The Connect:Direct bridge" on page 311

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related tasks

"Transferring a file to a Connect:Direct node" on page 313

You can transfer a file from a Managed File Transfer agent to a Connect:Direct node using the Connect:Direct bridge. Specify a Connect:Direct node as the destination of the transfer by specifying the Connect:Direct bridge agent as the destination agent and specifying the destination file in the form *connect_direct_node_name*:*file_path*.

"Transferring multiple files to Connect:Direct by using wildcards" on page 318

To transfer multiple files from a Managed File Transfer agent to a Connect:Direct node, use the Connect:Direct bridge. You can use wildcard characters in the source specification that you provide to

the **fteCreateTransfer** command. As with all Managed File Transfer transfers involving wildcards, only the last part of the file path can contain a wildcard character. For example, `/abc/def*` is a valid file path and `/abc*/def` is not valid.

"Transferring a file from a Connect:Direct node" on page 314
You can transfer a file from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the transfer by specifying the Connect:Direct bridge agent as the source agent and specifying the source specification in the form *connect_direct_node_name*:*file_path*.

"Transferring multiple files from a Connect:Direct node" on page 317
You can transfer multiple files from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the multiple file transfer by specifying the Connect:Direct bridge agent as the source agent and specifying one or more source specifications in the form *connect_direct_node_name*:*file_path*.

Related reference

The MFT `agent.properties` file

z/OS Transferring multiple files from a Connect:Direct node

You can transfer multiple files from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the multiple file transfer by specifying the Connect:Direct bridge agent as the source agent and specifying one or more source specifications in the form *connect_direct_node_name*:*file_path*.

Before you begin

Before transferring a file, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. See Configuring the Connect:Direct bridge.

About this task

In this example, the Connect:Direct bridge agent is called CD_BRIDGE. The destination agent is called FTE_Z, and is running on a z/OS system. The source Connect:Direct node is called CD_NODE1. The files to be transferred are located at the file paths `/in/file1`, `/in/file2`, and `/in/file3` on the system where CD_NODE1 is located. The files are transferred to the partitioned data set `//OBJECT.LIB` on the system where FTE_Z is running.

Procedure

Use the fteCreateTransfer command with the values for the source specifications in the form *connect_direct_node_name*:*file_path* and the value of the **-sa** parameter specified as the name of the Connect:Direct bridge agent.

Note: The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the files to be transferred from, not the Connect:Direct node that operates as part of the Connect:Direct bridge.

```
fteCreateTransfer -sa CD_BRIDGE -da FTE_Z
                  -dp //'OBJECT.LIB' CD_NODE1:/in/file1
                  CD_NODE1:/in/file2 CD_NODE1:/in/file3
```

For more information, see **fteCreateTransfer**: start a new file transfer.

Results

The Connect:Direct bridge agent CD_BRIDGE requests the first file from the Connect:Direct node CD_NODE1. The Connect:Direct node sends the file to the Connect:Direct bridge. While the file is being transferred from the Connect:Direct node, the Connect:Direct bridge stores the file temporarily in the location defined by the `cdTmpDir` agent property. When the file has finished transferring from the Connect:Direct node to the Connect:Direct bridge, the Connect:Direct bridge sends the file to the

destination agent FTE_Z and then deletes the file from the temporary location. This process is repeated for each specified source file.

Related concepts

"The Connect:Direct bridge" on page 311

You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related reference

The MFT `agent.properties` file

Transferring multiple files to Connect:Direct by using wildcards

To transfer multiple files from a Managed File Transfer agent to a Connect:Direct node, use the Connect:Direct bridge. You can use wildcard characters in the source specification that you provide to the **fteCreateTransfer** command. As with all Managed File Transfer transfers involving wildcards, only the last part of the file path can contain a wildcard character. For example, `/abc/def*` is a valid file path and `/abc*/def` is not valid.

Before you begin

Before transferring a file, you must configure the Connect:Direct bridge, which is a component of Managed File Transfer. For more information, see Configuring the Connect:Direct bridge.

About this task

In this example, the source agent is called FTE_AGENT and the Connect:Direct bridge agent is called CD_BRIDGE. The destination Connect:Direct node is called CD_NODE1. The files to be transferred are located in the directory `/reports` on the system where FTE_AGENT is located. Only files with names that start with `report`, followed by two characters and the suffix `.log`, are transferred. For example, the file `/reports/report01.log` is transferred, but the file `/reports/report1.log` is not transferred. The files are transferred to the directory `/home/fred` on the system where CD_NODE1 is running.

Procedure

1. Use the fteCreateTransfer command with the value for the **-dd** (destination directory) parameter in the form *connect_direct_node_name*:*directory_path*. For the **-da** (destination agent) parameter, specify the Connect:Direct bridge agent.

   Note: The Connect:Direct node specified by *connect_direct_node_name* is the node that you want the files to be transferred to, not the Connect:Direct node that operates as part of the Connect:Direct bridge.

   ```
   fteCreateTransfer -sa FTE_AGENT -da CD_BRIDGE
                     -dd CD_NODE1:/home/fred "/reports/report??.log"
   ```

   For more information, see **fteCreateTransfer**: start a new file transfer.
2. The source agent FTE_AGENT transfers the first file that matches the pattern `/reports/report??.log` to the Connect:Direct bridge agent CD_BRIDGE. The Connect:Direct bridge agent temporarily stores the file in the location defined by the cdTmpDir property. When the file has been completely transferred from the source agent to the Connect:Direct bridge, the Connect:Direct bridge agent sends the file to the Connect:Direct node that is defined by the cdNode agent property. This node sends the file to the destination Connect:Direct node CD_NODE1. The Connect:Direct bridge agent deletes the file from the temporary location when the transfer between the two Connect:Direct nodes is complete.This process is repeated for each source file that matches the wildcard pattern `/reports/report??.log`.

   Note: The list of files that match the pattern `/reports/report??.log` varies depending on the operating system of the system where the source agent FTE_AGENT is located.

- If the source agent is located on a system with a Windows operating system, the pattern matching is not case sensitive. The pattern matches all files in the `/reports` directory with a file name of the form `report` followed by two characters and a suffix of `.log`, regardless of the case that the letters are in. For example, `Report99.Log` is a match.
- If the source agent is located on a system with a Linux or UNIX operating system, the pattern matching is case sensitive. The pattern matches only those files in the `/reports` directory with a file name of the form `report` followed by two characters and a suffix of `.log`. For example, `reportAB.log` is a match, but `reportAB.LOG` and `Report99.Log` are not matches.

Related concepts

"The Connect:Direct bridge" on page 311
You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

Related tasks

Using wildcard characters with MFT

"Transferring a file to a Connect:Direct node" on page 313
You can transfer a file from a Managed File Transfer agent to a Connect:Direct node using the Connect:Direct bridge. Specify a Connect:Direct node as the destination of the transfer by specifying the Connect:Direct bridge agent as the destination agent and specifying the destination file in the form *connect_direct_node_name*:*file_path*.

"Transferring multiple files to a Connect:Direct node" on page 316
You can transfer multiple files from a Managed File Transfer Agent to a Connect:Direct node by using the Connect:Direct bridge. To use a Connect:Direct node as the destination of the multiple file transfer, specify the Connect:Direct bridge agent as the destination agent and specify the destination directory in the form *connect_direct_node_name*:*directory_path*.

"Transferring multiple files from a Connect:Direct node" on page 317
You can transfer multiple files from a Connect:Direct node to a Managed File Transfer Agent by using the Connect:Direct bridge. You can specify a Connect:Direct node as the source of the multiple file transfer by specifying the Connect:Direct bridge agent as the source agent and specifying one or more source specifications in the form *connect_direct_node_name*:*file_path*.

Related reference

The MFT `agent.properties` file

Recovery and restart for transfers to and from Connect:Direct nodes

Managed File Transfer might be unable to connect to your IBM Sterling Connect:Direct node during a transfer; for example, if the node becomes unavailable. Either Managed File Transfer attempts to recover the transfer, or the transfer fails and an error message is produced.

If the Connect:Direct node becomes unavailable

If the Connect:Direct node becomes unavailable; for example, due to a network or power outage, Managed File Transfer recovers a file transfer in the following ways:

- If Managed File Transfer has not previously successfully connected to the Connect:Direct node as part of this transfer request, the transfer is tried again for a length of time determined by the values of the **cdMaxConnectionRetries** and **recoverableTransferRetryInterval properties**. These properties are specified in the `agent.properties` file for the Connect:Direct bridge agent. The transfer fails, and an error message is produced, after the number of failed attempts reaches the value of the **cdMaxConnectionRetries property**. By default, the transfer is attempted indefinitely, with 60 seconds between attempts.
- If Managed File Transfer has previously successfully connected to the Connect:Direct node as part of this transfer request, the transfer is tried again for a length of time determined by the values of the **cdMaxPartialWorkConnectionRetries** and **recoverableTransferRetryInterval** properties. The transfer fails, and an error message is produced, after the number of failed attempts reaches the

value of the **cdMaxPartialWorkConnectionRetries** property. By default, the transfer is attempted indefinitely, with 60 seconds between attempts.

- For certain types of Connect:Direct node failure, for example the node being forcibly stopped, Connect:Direct processes go into `Held Due to Error (HE)` status when the node recovers. After the node recovers, Managed File Transfer automatically resumes any Connect:Direct processes that are related to the file transfer and have a status of HE.
- If the transfer fails, any temporary files relating to the transfer are deleted from the system that hosts the Connect:Direct bridge. The location of these temporary files is defined by the **cdTmpDir** property.
- If the transfer is from Managed File Transfer to Connect:Direct, and a source disposition of delete is specified, then the source files are not deleted if the transfer fails.

If the Connect:Direct node user credentials are invalid

If Managed File Transfer fails to connect to the Connect:Direct node because the credentials of the user are rejected by the node, the transfer fails and an error message is produced. In this situation, check that you have provided the correct user credentials for the Connect:Direct node. For more information, see Mapping credentials for Connect:Direct.

If the Connect:Direct bridge agent becomes unavailable

If the Connect:Direct bridge agent becomes unavailable, any ongoing file transfers recover in the same way as standard Managed File Transfer transfers. For more information, see “MFT recovery and restart” on page 327.

Related concepts

“The Connect:Direct bridge” on page 311
You can transfer files to and from an existing IBM Sterling™ Connect:Direct network. Use the Connect:Direct bridge, which is a component of Managed File Transfer, to transfer files between MFT and IBM Sterling Connect:Direct.

“MFT recovery and restart” on page 327
If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

Related tasks

Configuring the Connect:Direct bridge

Related reference

The MFT `agent.properties` file

Submitting a user-defined Connect:Direct process from a file transfer request

You can submit a transfer request for a transfer that goes through the Connect:Direct bridge agent that calls a user-defined Connect:Direct process as part of the file transfer.

By default, when you submit a file transfer request for a transfer that goes through the Connect:Direct bridge, the Connect:Direct bridge agent generates the Connect:Direct process that is used to transfer the file to or from the remote Connect:Direct node.

However, you can configure the Connect:Direct bridge agent to instead call a user-defined Connect:Direct process by using the `ConnectDirectProcessDefinition.xml` file.

The `ConnectDirectProcessDefinition.xml` file

The **fteCreateCDAgent** command creates the file `ConnectDirectProcessDefinitions.xml` in the agent configuration directory *MQ_DATA_PATH*`/mqft/config/`*coordination_queue_manager*`/agents/`*cd_bridge_agent_name*. Before you can call user-defined Connect:Direct processes from the Connect:Direct bridge agent, you must set up process definitions by editing this file.

The file defines one or more process sets that includes the location of one or more Connect:Direct processes that are called as part of a transfer. Each process set includes a number of conditions. If the transfer satisfies all of the conditions of the process set, the process set is used to specify which Connect:Direct processes are called by the transfer. For more information, see "Specifying the Connect:Direct process to start by using the ConnectDirectProcessDefinition.xml file" on page 321.

Intrinsic symbolic variables

You can use the intrinsic symbolic variables that are defined by Managed File Transfer to substitute values into user-defined Connect:Direct processes. To follow the Connect:Direct naming convention, all intrinsic symbolic variables used by Managed File Transfer have the format %FTE followed by five uppercase alphanumeric characters.

When creating a process to transfer files from a Connect:Direct node to the Connect:Direct bridge system, you must use the intrinsic variable %FTETFILE as the value of TO FILE in the Connect:Direct process. When creating a process to transfer files to a Connect:Direct node from the Connect:Direct bridge system, you must use the intrinsic variable %FTEFFILE as the value of FROM FILE in the Connect:Direct process. These variables contain the temporary file paths that the Connect:Direct bridge agent uses for transfers into and out of the Managed File Transfer network.

For more information about intrinsic symbolic variables, see the Connect:Direct product documentation.

Sample Connect:Direct processes

Managed File Transfer provides sample Connect:Direct processes. These samples are located in the following directory: *MQ_INSTALLATION_PATH*/mqft/samples/ConnectDirectProcessTemplates.

Related tasks

"Specifying the Connect:Direct process to start by using the ConnectDirectProcessDefinition.xml file" on page 321
Specify which Connect:Direct process to start as part of a Managed File Transfer transfer. Managed File Transfer provides an XML file that you can edit to specify process definitions.

"Using intrinsic symbolic variables in Connect:Direct processes that are called by Managed File Transfer" on page 323
You can call a user-defined Connect:Direct process from a Managed File Transfer transfer and pass in information from the transfer to the Connect:Direct process by using intrinsic symbolic variables in the process definition.

Related reference

Connect:Direct process definition file format

Substitution variables for use with user-defined Connect:Direct processes

Specifying the Connect:Direct process to start by using the ConnectDirectProcessDefinition.xml file

Specify which Connect:Direct process to start as part of a Managed File Transfer transfer. Managed File Transfer provides an XML file that you can edit to specify process definitions.

About this task

The **fteCreateCDAgent** command creates the file ConnectDirectProcessDefinitions.xml in the agent configuration directory *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*cd_bridge_agent_name*. Before you can call user-defined Connect:Direct processes from the Connect:Direct bridge agent, you must set up process definitions by editing this file.

For each process that you want to specify to call as part of a transfer through the Connect:Direct bridge, perform the following steps:

Procedure

1. Define the Connect:Direct process that you want the Connect:Direct bridge agent to call as part of the transfer and save the process template in a file.
2. Open the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*cd_bridge_agent_name*/ConnectDirectProcessDefinitions.xml file in a text editor.
3. Create a <processSet> element.
4. Inside the <processSet> element, create a <condition> element.
5. Inside the <condition> element, create one or more elements that define a condition that the transfer request must match to call the Connect:Direct process you defined in Step 1. These elements can be either <match> elements or <defined> elements.
   - Use a <match> element to specify that the value of a variable must match a pattern. Create the <match> element with the following attributes:
     - variable - the name of the variable whose value is compared. The variable is an intrinsic symbol. For more information, see Substitution variables for use with user-defined Connect:Direct processes.
     - value - the pattern to compare to the value of the specified variable.
     - Optional: pattern - the type of pattern used by the value of the value attribute. This pattern type can be wildcard or regex. This attribute is optional and the default is wildcard.
   - Use a <defined> element to specify that a variable must have a value defined. Create the <defined> element with the following attribute:
     - variable - the name of the variable that must have a value defined. The variable is an intrinsic symbol. For more information, see Substitution variables for use with user-defined Connect:Direct processes.

   The conditions specified within the <condition> element are combined with a logical AND. All conditions must be met for the Connect:Direct bridge agent to call the process specified by this <processSet> element. If you do not specify a <condition> element, the process set matches all transfers.
6. Inside the <processSet> element, create a <process> element.
7. Inside the <process> element, create a <transfer> element.

   The transfer element specifies the Connect:Direct process that the Connect:Direct bridge agent calls as part of the transfer. Create the <transfer> element with the following attribute:

   - process- - the location of the Connect:Direct process that you defined in step 1. The location of this file is specified with an absolute path or relative to the *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*cd_bridge_agent_name* directory.

Results

When searching for a condition match, the Connect:Direct bridge agent searches from the start of the file to the end of the file. The first match that is found is the one that is used.

Related tasks

Configuring the Connect:Direct bridge

Related reference

Connect:Direct process definition file format

fteCreateCDAgent: create a Connect:Direct bridge agent

Using intrinsic symbolic variables in Connect:Direct processes that are called by Managed File Transfer

You can call a user-defined Connect:Direct process from a Managed File Transfer transfer and pass in information from the transfer to the Connect:Direct process by using intrinsic symbolic variables in the process definition.

About this task

This example uses intrinsic symbolic variables to pass information from a Managed File Transfer transfer in to a user-defined Connect:Direct process. For more information about intrinsic symbolic variables used by Managed File Transfer, see Substitution variables for use with user-defined Connect:Direct processes.

In this example, the file is transferred from a Managed File Transfer Agent to a Connect:Direct bridge node. The first part of the transfer is performed by Managed File Transfer. The second part of the transfer is performed by a user-defined Connect:Direct process.

Procedure

1. Create a Connect:Direct process that uses intrinsic symbolic variables.

```
%FTEPNAME PROCESS
    SNODE=%FTESNODE
    PNODEID=(%FTEPUSER,%FTEPPASS)
    SNODEID=(%FTESUSER,%FTESPASS)

COPY001 COPY
    FROM (
        FILE=%FTEFFILE
        DISP=%FTEFDISP
    )
    TO (
        FILE=%FTETFILE
        DISP=%FTETDISP
    )
PEND
```

2. Save this process to a text file at the following location: *MQ_DATA_PATH*/mqft/config/*coordination_queue_manager*/agents/*cd_bridge_agent*/Example.cdp
3. Edit the ConnectDirectProcessDefinition.xml file to include a rule that calls the Connect:Direct process that you created in Step 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:cdprocess xmlns:tns="http://wmqfte.ibm.com/ConnectDirectProcessDefinitions"
               xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://wmqfte.ibm.com/
                ConnectDirectProcessDefinitions ConnectDirectProcessDefinitions.xsd">


    <tns:processSet>
        <tns:condition>
            <tns:match variable="%FTESNODE" value="TOBERMORY" pattern="wildcard" />
        </tns:condition>
        <tns:process>
            <tns:transfer process="Example.cdp" />
        </tns:process>
    </tns:processSet>

</tns:cdprocess>
```

   In this example, if a transfer request is submitted to the Connect:Direct bridge agent that has TOBERMORY as its source or destination Connect:Direct node, the Example.cdp Connect:Direct process is called.

4. Submit a file transfer request that satisfies the conditions that you defined in the ConnectDirectProcessDefinition.xml file in Step 3.

   For example,

```
fteCreateTransfer -sa ORINOCO -da CD_BRIDGE
                  -sm QM_WIMBLEDON -dm QM_COMMON
```

```
                    -de overwrite -df TOBERMORY:/home/bulgaria/destination.txt
                    -sd leave c:\bungo\source.txt
```

In this example, the destination Connect:Direct node is TOBERMORY. This node is the secondary node in the transfer and the value of %FTESNODE is set to TOBERMORY. This command matches the condition that is set in the `ConnectDirectProcessDefinition.xml` file.

5. Managed File Transfer transfers the source file to a temporary location on the same system as the Connect:Direct bridge agent.
6. The Connect:Direct bridge agent sets the values of the intrinsic symbolic variables from the information in the transfer request and configuration information.

   The intrinsic symbolic variables are set to the following values:

   - %FTEPNAME=*process_name* - This value is an 8 character process name generated by the Connect:Direct bridge agent.
   - %FTESNODE=TOBERMORY - This value is set from the **-df** parameter of the **fteCreateTransfer** command.
   - %FTEPUSER,=*primary_node_user* - This information is taken from the `ConnectDirectCredentials.xml` file.
   - %FTEPPASS=*primary_node_user_password* - This information is taken from the `ConnectDirectCredentials.xml` file.
   - %FTESUSER,=*secondary_node_user* - This information is taken from the `ConnectDirectCredentials.xml` file.
   - %FTESPASS=*secondary_node_user_password* - This information is taken from the `ConnectDirectCredentials.xml` file.
   - %FTEFFILE =*temporary_location* - This value is the temporary location of the file on the same system as the Connect:Direct bridge agent.
   - %FTEFDISP=leave - This value is set from the **-sd** parameter of the **fteCreateTransfer** command.
   - %FTETFILE=`/home/bulgaria/destination.txt` - This value is set from the **-df** parameter of the **fteCreateTransfer** command.
   - %FTETDISP=overwrite - This value is set from the **-de** parameter of the **fteCreateTransfer** command.
7. The Connect:Direct process is started on the Connect:Direct bridge node. Connect:Direct transfers the file from the temporary location on the Connect:Direct bridge system to the destination `/home/bulgaria/destination.txt` on the system where the Connect:Direct node TOBERMORY is running.

Related concepts

“Submitting a user-defined Connect:Direct process from a file transfer request” on page 320
You can submit a transfer request for a transfer that goes through the Connect:Direct bridge agent that calls a user-defined Connect:Direct process as part of the file transfer.

Related reference

Substitution variables for use with user-defined Connect:Direct processes

Using Connect:Direct processes to submit Managed File Transfer transfer requests

You can submit a transfer request to the Connect:Direct bridge agent from a Connect:Direct process. Managed File Transfer provides commands that can be called from a **RUN TASK** statement in a Connect:Direct process.

Managed File Transfer provides the following commands for use with Connect:Direct processes:

ftetag
Specify this command in a step that precedes the **ftebxfer** or **ftecxfer** command to create the required audit information for the transfer. This command takes the source specification of the transfer as a parameter. For information about the format of the source specification, see **fteCreateTransfer**: start a new file transfer.

ftebxfer
Specify this command to create a file transfer request when the queue manager that the transfer request is submitted to is on the same system as the Connect:Direct node that submits the command. This command takes the same parameters as the **fteCreateTransfer** command. For information about these parameters, see **fteCreateTransfer**: start a new file transfer. This command also has an extra parameter:

-qmgrname
Required. The name of the queue manager to submit the command to.

ftecxfer
Specify this command to create a file transfer request when the queue manager that the transfer request is submitted to is located on a different system to the Connect:Direct node that submits the command. This command takes the same parameters as the **fteCreateTransfer** command. For information about the parameters, see **fteCreateTransfer**: start a new file transfer. This command also has three additional parameters:

-qmgrname
Required. The name of the queue manager to submit the command to.

-connname
Required. The host and port of the queue manager to submit the command to, specified in IBM MQ CONNAME format. For example, `host.example.com(1337)`.

-channelname
Optional. The name of the channel to use to connect to the queue manager to submit the command to. If this is not specified, a default of SYSTEM.DEF.SVRCONN is used.

Related tasks
"Creating and submitting a Connect:Direct process that calls Managed File Transfer by using the Connect:Direct Requester" on page 325
The Connect:Direct Requester is a graphical user interface that you can use to create and submit a Connect:Direct process that calls Managed File Transfer.

Related reference
Example: A Connect:Direct process file that calls MFT commands

Creating and submitting a Connect:Direct process that calls Managed File Transfer by using the Connect:Direct Requester

The Connect:Direct Requester is a graphical user interface that you can use to create and submit a Connect:Direct process that calls Managed File Transfer.

About this task

This task describes how to create a Connect:Direct process that calls the Managed File Transfer **ftecxfer** command or the **ftebxfer** command. Use the **ftecxfer** command when the queue manager that the transfer request is submitted to is located on a different system to the Connect:Direct node that submits the command. Use the **ftebxfer** command when the queue manager that the transfer request is submitted to is located on the same system as the Connect:Direct node that submits the command. The **ftecxfer** command makes a client connection to the agent queue manager of the source agent of the transfer. Before calling the **ftecxfer** command, you must call the **ftetag** command and pass it the source specification information. This allows the process to be logged and audited in the same way as transfers initiated from Managed File Transfer.

Procedure

1. Start the Connect:Direct Requester.
2. In the **Nodes** tab of the panel, select the Connect:Direct node that is used as the primary node of the process.
3. Select **File** > **New** > **Process**. The **Process Properties** window opens.
4. In the **Name:** field, type the name of the process.
5. Select the secondary node from the **Snode** > **Name:** list.
6. Select the operating system of the secondary node from the **Snode** > **Operating System:** list.
7. Optional: Complete any further information in this window that you require.
8. Click **OK**. The **Process Properties** window closes.
9. Create a statement that runs the Managed File Transfer **ftetag** command.
   a) Right-click in the **Process** window on the **End** statement.
   b) Select **Insert** > **Run Task**. The **Run Task Statement** window opens.
   c) In the **Label:** field, type `Tag`.
   d) In the **Optional Parameters or Commands** field, type `pgm(`*MQ_INSTALLATION_PATH*`/bin/ftetag) args(`*source_specification*`)`. For more information about the format of *source_specification*, see **fteCreateTransfer**: start a new file transfer.
   e) Click **OK**. The **Run Task Statement** window closes.
10. Create a statement that runs the Managed File Transfer **ftecxfer** or **ftebxfer** command.
    a) Right-click in the **Process** window on the **End** statement.
    b) Select **Insert** > **Run Task**. The **Run Task Statement** window opens.
    c) In the **Label:** field, type `Transfer`.
    d) In the **Optional Parameters or Commands** field, type `pgm(`*MQ_INSTALLATION_PATH*`/bin/ftecxfer) args(`*parameters*`)` or `pgm(`*MQ_INSTALLATION_PATH*`/bin/ftebxfer) args(`*parameters*`)` depending on which command you choose. The parameters used by the **ftecxfer** and **ftebxfer** commands are the same as the parameters used by the **fteCreateTransfer** command, plus some additional parameters specific to **ftecxfer** and **ftebxfer**. For more information, see **fteCreateTransfer**: start a new file transfer and "Using Connect:Direct processes to submit Managed File Transfer transfer requests" on page 324.
    e) Click **OK**. The **Run Task Statement** window closes.
11. Optional: Create any additional statements that you require.
12. Submit the process.
    a) Right-click in the **Process** window.
    b) Select **Submit**. The **Connect:Direct Attach** window opens.
    c) Enter the user name and password to use to run the process.
    d) Click **OK**.

Related concepts

"Using Connect:Direct processes to submit Managed File Transfer transfer requests" on page 324
You can submit a transfer request to the Connect:Direct bridge agent from a Connect:Direct process. Managed File Transfer provides commands that can be called from a **RUN TASK** statement in a Connect:Direct process.

Working with MFT from IBM Integration Bus

You can work with Managed File Transfer from IBM Integration Bus using the FTEOutput and FTEInput nodes.

- Use the FTEInput node to transfer a file across the network using Managed File Transfer and then process that file as part of an Integration Bus flow.

- Use the FTEOutput node to transfer a file that has been output by an Integration Bus flow to another location in the network.

The agents that transfer files to or from the broker agent can be at any level of Managed File Transfer.

For more information, refer to the IBM Integration Bus product documentation.

MFT recovery and restart

If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

- Typically, if there is a problem while a file is being transferred, Managed File Transfer recovers and restarts that file transfer after the problem is repaired.
- If a file that was in the process of being transferred is deleted or changed while the agent or queue manager are unavailable, the transfer fails and you get a message in the transfer log that provides details about the failure.
- If an agent process fails during a file transfer, the transfer continues when you restart the agent.
- If an agent loses the connection to its agent queue manager, the agent waits while trying to reconnect to the queue manager. When the agent successfully reconnects to its queue manager, the current transfer continues.
- If the agent is stopped for any reason, any resource monitors associated with an agent stop polling. When the agent recovers, the monitors are also restarted, and resource polling resumes.
- For a file transfer with a source disposition of `delete`, if a recovery occurs after all the data is sent from a source agent to a destination agent, the source file is unlocked before deletion. This unlocking means that the source file could possibly be modified before the file is deleted. Therefore, it is considered to be unsafe to delete the source file and the following warning is displayed:

  ```
  BFGTR0075W: The source file has not been deleted because it is possible that the source file
  was modified after the source file was transferred.
  ```

  In this case, verify that the content of the source file is unmodified and then manually delete the source file.

You can check the status of your transfers in the IBM MQ Explorer. If any transfers appear as `Stalled`, you might need to take corrective action because the stalled status denotes an issue either with the agent or between the two agents involved in the transfer.

Related tasks

"Setting a timeout for recovery of stalled transfers" on page 327
You can set a transfer recovery timeout for stalled file transfers that applies to all the transfers for a source agent. You can also set a transfer recovery timeout for an individual transfer. If you set a specific amount of time, in seconds, during which a source agent keeps trying to recover a stalled file transfer and the transfer is not successful when the agent reaches the timeout, the transfer fails.

Setting a timeout for recovery of stalled transfers

You can set a transfer recovery timeout for stalled file transfers that applies to all the transfers for a source agent. You can also set a transfer recovery timeout for an individual transfer. If you set a specific amount of time, in seconds, during which a source agent keeps trying to recover a stalled file transfer and the transfer is not successful when the agent reaches the timeout, the transfer fails.

About this task

You can set a transfer recovery timeout that applies to all the transfers for a source agent by adding a transfer recovery timeout parameter to the agent's `agent.properties` file. You can also set a transfer recovery timeout for an individual transfer from the command line, or with IBM MQ Explorer, or by using Apache Ant tasks. If there is a transfer recovery timeout value set in the `agent.properties` file, setting

the transfer recovery timeout for an individual transfer overrides the value in the `agent.properties` file.

There are three options for transfer recovery timeout:

- The agent continues to try to recover the stalled transfer until it completes successfully. This is the same as the default behavior of the agent if the transfer recovery timeout is not set.
- The agent marks the transfer as failed immediately upon entering recovery.
- The agent keeps retrying the stalled transfer for a specified amount of time before the transfer is marked as failed.

Setting the file transfer recovery timeout is optional. If you do not set it, transfers follow the default behavior, where the agent keeps trying to recover a stalled transfer until it is successful.

Related concepts

"MFT recovery and restart" on page 327
If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

Transfer recovery timeout concepts

You can set the amount of time, in seconds, during which a source agent keeps trying to recover a stalled file transfer. If the transfer is not successful when the agent reaches the timeout for the retry interval, the transfer fails.

Recovery timeout precedence

A transfer recovery timeout value for an individual transfer specified through the **fteCreateTransfer**, **fteCreateTemplate**, or **fteCreateMonitor** commands, or by using IBM MQ Explorer, or specified in the **fte:filespec** nested element, takes precedence over the value that is specified for the **transferRecoveryTimeout** parameter in the `agent.properties` file for the source agent.

For example, if the **fteCreateTransfer** command is started without the **-rt** parameter and value pair, the source agent AGENT1 checks the `agent.properties` file for a **transferRecoveryTimeout** value to determine the recovery timeout behavior:

```
fteCreateTransfer -sa AGENT1 -da AGENT2 -df C:\import\transferredfile.txt
C:\export\originalfile.txt
```

If the **transferRecoveryTimeout** parameter in the `agent.properties` file is either not set or is set to `-1`, the agent follows the default behavior and tries to recover the transfer until it is successful.

However, if the **fteCreateTransfer** command includes the **-rt** parameter, the value of this parameter takes precedence over the value in the `agent.properties` file and is used as the recovery timeout setting for the transfer:

```
fteCreateTransfer -sa AGENT1 -da AGENT2 -rt 21600 -df C:\import\transferredfile.txt
C:\export\originalfile.txt
```

Recovery timeout counter

The recovery timeout counter starts when the transfer enters recovering state. A transfer log message is published to the SYSTEM.FTE topic with the topic string `Log/agent_name/transfer_ID` to indicate that the transfer status is changed to recovering and the source agent clock time at which the status changed. If the transfer is resumed within the set retry interval and does not reach the recovery timeout (counter<=recovery timeout), then the counter is reset to 0, ready to start again if the transfer enters recovery.

If the counter reaches the maximum value set for the recovery timeout (counter==recovery timeout), the recovery of the transfer stops and the source agent reports the transfer as failed. This type of transfer failure, caused by the fact that the transfer reached the recovery timeout, is indicated by the message code, `RECOVERY TIMEOUT (69)`. Another transfer log message is published to the SYSTEM.FTE topic,

with a topic string of `Log/agent_name/transfer_ID`, to indicate that the transfer is failed and includes a message, the return code, and the source agent's event log. The source agent's event log is updated with a message when any of the following events occur during recovery:

- When the recovery timeout parameter is set to a value greater than `-1`, the transfer enters recovery. The agent's event log is updated to indicate the start of the recovery timer for the **`TransferId`** and the amount of time the source agent waits before it initiates the recovery timeout processing.
- When the recovering transfer is resumed, the source agent's event log is updated with a new message to indicate that the **`TransferId`** that was in recovery is resumed.
- When a recovering transfer has timed out, the source agent's event log is updated to indicate the **`TransferId`** that failed while recovering due to recovery timeout.

These log messages enable the users (subscribers and loggers) to identify the transfers that failed due to the transfer recovery timeout.

The counter for the recovery timeout is always at the source agent. However, if the destination agent fails to receive information from the source agent in a timely manner, it can send a request to the source agent to put the transfer in recovery. For a transfer where the recovery timeout option is set, the source agent starts the recovery timeout counter when it receives the request from the destination agent.

Manual handling is still required for transfers that do not use the recovery timeout option, the failed, and partially complete transfers.

For transfer sets, where a single transfer request is issued for multiple files, and some of the files completed successfully but one completed only partially, the transfer is still marked as failed as it did not complete as expected. The source agent might have timed out while transferring the partially completed file.

Ensure that the destination agent and file server are ready and in a state to accept file transfers.

You have to issue the transfer request again for the entire set, but to avoid problems because some of the files remain on the destination from the initial transfer attempt, you can issue the new request with the `overwrite if existing` option specified. This ensures that the incomplete set of files from the previous transfer attempt are cleaned up as a part of the new transfer, before the files are written to the destination again.

From IBM MQ 9.1.5, it is no longer necessary to manually remove part files left on a destination after an initial transfer attempt has failed. If a transfer recovery timeout is set for a transfer, the source agent moves the transfer into the RecoveryTimedOut state if transfer recovery times out. After the transfer has been resynchronized, the destination agent removes any part files that were created during the transfer and sends a completion message to the source agent.

Traces and messages

Tracing points are included for diagnostic purposes. The recovery timeout value, start of the retry interval, start of the resume period and counter reset, and whether the transfer timed out and failed, are logged. In case of a problem or unexpected behavior, you can collect the source agent output log and trace files, and provide them when requested by IBM Support, to help with troubleshooting.

Messages notify you when:

- A transfer enters recovery (BFGTR0081I)
- A transfer is terminated because it timed out from recovery (BFGSS0081E)
- Atransfer resumes after being in recovery (BFGTR0082I)

Related concepts

“MFT recovery and restart” on page 327

If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

Setting the transfer recovery timeout for all transfers for one source agent

You can set a transfer recovery timeout that applies to all the transfers for a source agent by adding the **transferRecoveryTimeout** parameter to the `agent.properties` file.

About this task

To set a treansfer recovery timeout that applies to all the transfers for a source agent, you add the parameter and value pair for the **transferRecoveryTimeout** to the agent.properties file.

There are three options for the **transferRecoveryTimeout** parameter:

-1
The agent continues to attempt to recover the stalled transfer until the transfer is successful. Using this option is the equivalent of the default behavior of the agent when the property is not set.

0
The agent stops the file transfer as soon as it enters recovery.

>0
The agent continues to attempt to recover the stalled transfer for the amount of time in seconds as set by the positive integer value specified.

Any changes that you make to the `agent.properties` file come into effect only after the agent is restarted.

If required, you can override the transfer recovery timeout value in the `agent.properties` file for an individual transfer. For more information, see “Setting the transfer recovery timeout for individual transfers” on page 330.

Procedure

- To specify that the agent continues to try to recover the stalled transfer until it completes successfully, set a transfer recovery timeout value of `-1` as shown in the following example:

  ```
  transferRecoveryTimeout=-1
  ```

- To specify that the agent marks the transfer as failed immediately upon entering recovery, set a transfer recovery timeout value of `0` as shown in the following example:

  ```
  transferRecoveryTimeout=0
  ```

- To specify that the agent keeps retrying a stalled transfer for a given amount of time before the transfer is marked as failed, set a transfer recovery timeout value for the amount of time, in seconds, that you want to agent to keep retrying.

  For example, setting a transfer recovery timeout value of `21600` means that the agent keeps trying to recover the transfer for six hours from when it enters recovery:

  ```
  transferRecoveryTimeout=21600
  ```

  The maximum value for this parameter is `999999999`.

Setting the transfer recovery timeout for individual transfers

You can set the transfer recovery timeout for an individual transfer from the command line, or with IBM MQ Explorer, or by using Apache Ant tasks. If there is a transfer recovery timeout value set in the

agent.properties file, setting the transfer recovery timeout for an individual transfer overrides the value set in the agent.properties file.

About this task

You can set the transfer recovery timeout parameter for an individual transfer when you are:

- Creating a transfer either by using the **fteCreateTransfer** command, or by using IBM MQ Explorer.
- Creating a transfer template either by using the **fteCreateTemplate** command, or by using IBM MQ Explorer.
- Creating a resource monitor either by using the **fteCreateMonitor** command, or by using IBM MQ Explorer.
- Copying or moving files by using the fte:filecopy or fte:filemove Ant tasks.

If you set a transfer recovery timeout value for an individual transfer, this value overrides the transfer recovery timeout value set in the agent.properties file (see"Setting the transfer recovery timeout for all transfers for one source agent" on page 330).

Procedure

- To use the **fteCreateTransfer** or **fteCreateTemplate** command to set the transfer recovery timeout, specify the appropriate option for the **-rt** parameter:

  -1
  The agent continues to attempt to recover the stalled transfer until the transfer is successful. Using this option is the equivalent of the default behavior of the agent when the property is not set.

  0
  The agent stops the file transfer as soon as it enters recovery.

  >0
  The agent continues to attempt to recover the stalled transfer for the specified amount of time in seconds.

  Examples for the **fteCreateTransfer** command

```
fteCreateTransfer -sa AGENT1 -da AGENT2 -rt -1 -df C:\import\transferredfile.txt
C:\export\originalfile.txt
```

```
fteCreateTransfer -sa AGENT1 -da AGENT2 -rt 0 -df C:\import\transferredfile.txt
C:\export\originalfile.txt
```

```
fteCreateTransfer -sa AGENT1 -da AGENT2 -rt 21600 -df C:\import\transferredfile.txt
C:\export\originalfile.txt
```

  Examples for the **fteCreateTemplate** command

```
fteCreateTemplate -tn "payroll accounts monthly report template" -rt -1 -sa PAYROLL -sm
QM_PAYROLL1 -da ACCOUNTS
-dm QM_ACCOUNTS -df C:\payroll_reports\*.xls C:\out\*.xls
```

```
fteCreateTemplate -tn "payroll accounts monthly report template" -rt 0 -sa PAYROLL -sm
QM_PAYROLL1 -da ACCOUNTS
-dm QM_ACCOUNTS -df C:\payroll_reports\*.xls C:\out\*.xls
```

```
fteCreateTemplate -tn "payroll accounts monthly report template" -rt 21600 -sa PAYROLL -sm
QM_PAYROLL1 -da ACCOUNTS
-dm QM_ACCOUNTS -df C:\payroll_reports\*.xls C:\out\*.xls
```

  There is no **-rt** parameter for the **fteCreateMonitor** command. If you set the **-rt** parameter with the **fteCreateTransfer** command and also set the **-gt** parameter, the recovery timeout parameter is included in the XML document with the transfer definition that is generated when you run the **fteCreateTransfer** command. The resource monitor then uses this the XML document when

you run the **fteCreateMonitor** command. In the following example, the transfer recovery timeout details would be included in the `task.xml` file:

```
fteCreateMonitor -ma AgentName -md C:\mqmft\monitors -mn Monitor_Name -mt task.xml -tr
"fileSize>=5MB,*.zip"
```

- To use IBM MQ Explorer New Transfer, New Monitor, or New Template wizard page to set the transfer recovery timeout, select the required option in the **Transfer Recovery Timeout** (seconds) field:

  As Source Agent
  If you select **As Source Agent**, the **transferRecoveryTimeout** parameter value from the `agent.properties` file is used if it is set, otherwise the default behavior for transfer recovery timeout is applied.

  Numeric list box
  If you enter a time in seconds in the numeric list box, the agent continues to attempt to recover the stalled transfer for the specified amount of time.

  None
  If you select **None**, no transfer recovery timeout is set and the agent continues to attempt to recover the stalled transfer until the transfer is successful.

- To set the recovery timeout by using Ant tasks. include the **transferRecoveryTimeout** option and value, with the **fte:filecopy** or **fte:filemove** elements for moving or copying files, for example:

  Example for **fte:filecopy**

```
<fte:filecopy cmdqm="qm0@localhost@1414@SYSTEM.DEF.SVRCONN"
              src="agent1@qm1" dst="agent2@qm2"
              rcproperty="copy.result" transferRecoveryTimeout="0">

    <fte:filespec srcfilespec="/home/fteuser1/file.bin" dstfile="/home/fteuser2/file.bin"/>

</fte:filecopy>
```

  Example for **fte:filemove**

```
<fte:filemove cmdqm="qm0@localhost@1414@SYSTEM.DEF.SVRCONN"
              src=agent1@qm1 dst="agent2@qm2"
              rcproperty="move.result" transferRecoveryTimeout="21600">

    <fte:filespec srcfilespec="/home/fteuser1/file.bin" dstfile="/home/fteuser2/file.bin"/>

</fte:filemove>
```

Related concepts

“MFT recovery and restart” on page 327
If your agent or queue manager are unavailable for any reason, for example because of a power or network failure, Managed File Transfer recovers as follows in these scenarios:

Related tasks

“Starting a new file transfer” on page 220
You can start a new file transfer from the IBM MQ Explorer or from the command line and you can choose to transfer either a single file or multiple files in a group.

“Creating a file transfer template using IBM MQ Explorer” on page 262
You can create a file transfer template from IBM MQ Explorer or from the command line. You can then use that template to create new file transfers using the template details or submit the template to start the file transfer.

“Monitoring MFT resources” on page 230
You can monitor Managed File Transfer resources; for example, a queue or a directory. When a condition on this resource is satisfied, the resource monitor starts a task, such as a file transfer. You can create a resource monitor by using the **fteCreateMonitor** command or the **Monitors** view in the Managed File Transfer plug-in for IBM MQ Explorer.

“Viewing the status of file transfers in the Transfer Log” on page 228

You can view the details of file transfers by using the **Transfer Log** in IBM MQ Explorer. These can be transfers started from either the command line or the IBM MQ Explorer. You can also customize what is displayed in the **Transfer Log**.

Related reference

The MFT `agent.properties` file

`fteCreateTransfer`: start a new file transfer

`fteCreateTemplate`: create new file transfer template

`fteCreateMonitor`: create an MFT resource monitor

fte:filecopy Ant task

fte:filemove Ant task

ALW Administering IBM MQ Telemetry

IBM MQ Telemetry is administered using IBM MQ Explorer or at a command line. Use the explorer to configure telemetry channels, control the IBM MQ Telemetry service, and monitor the MQTT clients that are connected to IBM MQ. Configure the security of IBM MQ Telemetry using JAAS, TLS and the IBM MQ object authority manager.

Administering using IBM MQ Explorer

Use the explorer to configure telemetry channels, control the IBM MQ Telemetry service, and monitor the MQTT clients that are connected to IBM MQ. Configure the security of IBM MQ Telemetry using JAAS, TLS and the IBM MQ object authority manager.

Administering using the command line

IBM MQ Telemetry can be completely administered at the command line using MQSC commands.

The IBM MQ Telemetry documentation also has sample scripts that demonstrate the basic usage of the MQ Telemetry Transport v3 Client application.

Read and understand the samples in MQ Telemetry Transport sample programs before using them.

Related concepts

IBM MQ Telemetry

Related reference

MQXR properties

Linux AIX Configuring a queue manager for telemetry on Linux and AIX

Follow these steps to configure IBM MQ Telemetry manually. If you only need a simple configuration that uses the guest user ID, you can instead run the IBM MQ Telemetry support wizard in IBM MQ Explorer.

Before you begin

If you only need a simple configuration, consider using the IBM MQ Telemetry support in IBM MQ Explorer. This support includes a wizard and a sample command procedure `sampleMQM`. These resources set up an initial configuration using the guest user ID. See Verifying the installation of IBM MQ Telemetry by using IBM MQ Explorer and MQ Telemetry Transport sample programs.

If you need a more complex configuration that uses a different authentication method, use the steps in this task. Begin with the following initial steps:

1. See Installation considerations for IBM MQ Telemetry for information on how to install IBM MQ, and the IBM MQ Telemetry feature.
2. Create and start a queue manager. The queue manager is referred to as *qMgr* in this task.
3. As part of this task you configure the telemetry (MQXR) service. The MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties`. You do not normally need to edit the

MQXR properties file directly, because almost all settings can be configured through MQSC admin commands or IBM MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes. See MQXR properties.

About this task

Follow the steps in this task to configure IBM MQ Telemetry manually, using different authorization schemes.

Procedure

1. Open a command window at the telemetry samples directory.

   The telemetry samples directory is `/opt/mqm/mqxr/samples`.
2. Create the telemetry transmission queue.

   If `SYSTEM.MQTT.TRANSMIT.QUEUE` does not exist, it is created automatically when the telemetry (MQXR) service is first started, and set to use the guest user ID. However this task configures IBM MQ Telemetry to use a different authorization scheme. For this task you create `SYSTEM.MQTT.TRANSMIT.QUEUE`, and configure access to it, before you start the telemetry (MQXR) service.

   Run the following command:

   ```
   echo "DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000)" | runmqsc qMgr
   ```
3. Set the default transmission queue.

   It is easier to send messages directly to MQTT clients if `SYSTEM.MQTT.TRANSMIT.QUEUE` is the default transmission queue. Otherwise, you have to add a remote queue definition for every client that receives IBM MQ messages; see “Sending a message to a client directly” on page 338. Note that altering the default transmission queue might interfere with your existing configuration.

   When the telemetry (MQXR) service is first started, it does not set `SYSTEM.MQTT.TRANSMIT.QUEUE` as the default transmission queue for the queue manager. To configure this setting, you alter the default transmission queue property. You do this either by using the IBM MQ Explorer, or by running the following command:

   ```
   echo "ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE')" | runmqsc qMgr
   ```
4. Follow a procedure in “Authorizing MQTT clients to access IBM MQ objects” on page 340 to create one or more user IDs. The user IDs have the authority to publish, subscribe, and send publications to MQTT clients.
5. Edit the `installMQXRService_unix.mqsc` file to configure the key file that is used to encrypt passphrase for MQTT TLS channels:
   a) Open the *WMQ program installation directory*`/mqxr/samples/installMQXRService_unix.mqsc` file.
   b) Locate the line that includes the **STARTARG**parameter and edit the **-sf** option to specify the location of your credentials key file.

      By default, the `installMQXRService_unix.mqsc` file uses a default key file named `[DEFAULT]`. The default key file is the same for all IBM MQ installations, so you must supply a key file that is unique to your installation when you encrypt passphrases.

      See also the example code in “Creating the SYSTEM.MQXR.SERVICE” on page 335.
6. Install the telemetry (MQXR) service by running the following command:

   ```
   cat /opt/<install_dir>/mqxr/samples/installMQXRService_unix.mqsc | runmqsc queue_manager
   ```
7. Start the service.

```
echo "START SERVICE(SYSTEM.MQXR.SERVICE)" | runmqsc qMgr
```

The telemetry (MQXR) service is started automatically when the queue manager is started. It is started manually in this task, because the queue manager is already running.

8. Using IBM MQ Explorer, configure telemetry channels to accept connections from MQTT clients.

   The telemetry channels must be configured such that their identities are one of the user IDs defined in step “4” on page 334.

   See also DEFINE CHANNEL (MQTT).

9. Verify the configuration by running the sample client.

   For the sample client to work with your telemetry channel, the channel must authorize the client to publish, subscribe, and receive publications. The sample client connects to the telemetry channel on port 1883 by default. See also MQ Telemetry Transport sample programs.

Creating the SYSTEM.MQXR.SERVICE

Use the **runMQXRService** command to create the SYSTEM.MQXR.SERVICE.

```
DEF    SERVICE(SYSTEM.MQXR.SERVICE) +
CONTROL(QMGR) +
DESCR('Manages clients using MQXR protocols such as MQTT') +
SERVTYPE(SERVER) +
STARTCMD('+MQ_INSTALL_PATH+/mqxr/bin/runMQXRService.sh') +
STARTARG('-m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+" -g "+MQ_DATA_PATH+" -sf "/home/keyFileLocation/
keyFile.txt"') +
STOPCMD('+MQ_INSTALL_PATH+/mqxr/bin/endMQXRService.sh') +
STOPARG('-m +QMNAME+') +
STDOUT('+MQ_Q_MGR_DATA_PATH+/mqxr.stdout') +
STDERR('+MQ_Q_MGR_DATA_PATH+/mqxr.stderr')
```

Windows Configuring a queue manager for telemetry on Windows

Follow these steps to configure IBM MQ Telemetry manually. If you only need a simple configuration that uses the guest user ID, you can instead run the IBM MQ Telemetry support wizard in IBM MQ Explorer.

Before you begin

If you only need a simple configuration, consider using the IBM MQ Telemetry support in IBM MQ Explorer. This support includes a wizard and a sample command procedure `sampleMQM`. These resources set up an initial configuration using the guest user ID. See Verifying the installation of IBM MQ Telemetry by using IBM MQ Explorer and MQ Telemetry Transport sample programs.

If you need a more complex configuration that uses a different authentication method, use the steps in this task. Begin with the following initial steps:

1. See Installation considerations for IBM MQ Telemetry for information on how to install IBM MQ, and the IBM MQ Telemetry feature.
2. Create and start a queue manager. The queue manager is referred to as *qMgr* in this task.
3. As part of this task you configure the telemetry (MQXR) service. The MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties`. You do not normally need to edit the MQXR properties file directly, because almost all settings can be configured through MQSC admin commands or IBM MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes. See MQXR properties.

About this task

Follow the steps in this task to configure IBM MQ Telemetry manually, using different authorization schemes.

Procedure

1. Open a command window at the telemetry samples directory.

   The telemetry samples directory is *WMQ program installation directory*\mqxr\samples.
2. Create the telemetry transmission queue.

   If SYSTEM.MQTT.TRANSMIT.QUEUE does not exist, it is created automatically when the telemetry (MQXR) service is first started, and set to use the guest user ID. However this task configures IBM MQ Telemetry to use a different authorization scheme. For this task you create SYSTEM.MQTT.TRANSMIT.QUEUE, and configure access to it, before you start the telemetry (MQXR) service.

   Run the following command:

   ```
   echo DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000) | runmqsc qMgr
   ```
3. Set the default transmission queue.

   It is easier to send messages directly to MQTT clients if SYSTEM.MQTT.TRANSMIT.QUEUE is the default transmission queue. Otherwise, you have to add a remote queue definition for every client that receives IBM MQ messages; see "Sending a message to a client directly" on page 338. Note that altering the default transmission queue might interfere with your existing configuration.

   When the telemetry (MQXR) service is first started, it does not set SYSTEM.MQTT.TRANSMIT.QUEUE as the default transmission queue for the queue manager. To configure this setting, you alter the default transmission queue property. You do this either by using the IBM MQ Explorer, or by running the following command:

   ```
   echo ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE') | runmqsc qMgr
   ```
4. Follow a procedure in "Authorizing MQTT clients to access IBM MQ objects" on page 340 to create one or more user IDs. The user IDs have the authority to publish, subscribe, and send publications to MQTT clients.
5. Edit the installMQXRService_win.mqsc file to configure the key file that is used to encrypt passphrase for MQTT TLS channels:
   a) Open the *WMQ program installation directory*\mqxr\samples\installMQXRService_win.mqsc file.
   b) Locate the line that includes the **STARTARG**parameter and edit the **-sf** option to specify the location of your credentials key file.

      By default, the installMQXRService_win.mqsc file uses a default key file named [DEFAULT]. The default key file is the same for all IBM MQ installations, so you must supply a key file that is unique to your installation when you encrypt passphrases.

      See also the example code in "Creating SYSTEM.MQXR.SERVICE" on page 337.
6. Install the telemetry (MQXR) service by running the following command:

   ```
   type installMQXRService_win.mqsc | runmqsc qMgr
   ```
7. Start the service.

   ```
   echo START SERVICE(SYSTEM.MQXR.SERVICE) | runmqsc qMgr
   ```

   The telemetry (MQXR) service is started automatically when the queue manager is started. It is started manually in this task, because the queue manager is already running.
8. Using IBM MQ Explorer, configure telemetry channels to accept connections from MQTT clients.

   The telemetry channels must be configured such that their identities are one of the user IDs defined in step "4" on page 336.

   See also DEFINE CHANNEL (MQTT).
9. Verify the configuration by running the sample client.

For the sample client to work with your telemetry channel, the channel must authorize the client to publish, subscribe, and receive publications. The sample client connects to the telemetry channel on port 1883 by default. See also MQ Telemetry Transport sample programs.

Creating SYSTEM.MQXR.SERVICE

Use the **runMQXRService** command to create the SYSTEM.MQXR.SERVICE.

```
DEF    SERVICE(SYSTEM.MQXR.SERVICE) +
CONTROL(QMGR) +
DESCR('Manages clients using MQXR protocols such as MQTT') +
SERVTYPE(SERVER) +
STARTCMD('+MQ_INSTALL_PATH+\mqxr\bin\runMQXRService.bat') +
STARTARG('-m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+\." -g "+MQ_DATA_PATH+\." -sf
"c:\keyFileLocation\keyFile.txt"') +
STOPCMD('+MQ_INSTALL_PATH+\mqxr\bin\endMQXRService.bat') +
STOPARG('-m +QMNAME+') +
STDOUT('+MQ_Q_MGR_DATA_PATH+\mqxr.stdout') +
STDERR('+MQ_Q_MGR_DATA_PATH+\mqxr.stderr')
```

ALW Configuring distributed queuing to send messages to MQTT clients

IBM MQ applications can send MQTT v3 clients messages by publishing to subscription created by a client, or by sending a message directly. Whichever method is used, the message is placed on SYSTEM.MQTT.TRANSMIT.QUEUE, and sent to the client by the telemetry (MQXR) service. There are a number of ways to place a message on SYSTEM.MQTT.TRANSMIT.QUEUE.

Publishing a message in response to an MQTT client subscription

The telemetry (MQXR) service creates a subscription on behalf of the MQTT client. The client is the destination for any publications that match the subscription sent by the client. The IBM MQ Telemetry services forwards matching publications back to the client.

An MQTT client is connected to IBM MQ as a queue manager, with its queue manager name set to its ClientIdentifier. The destination for publications to be sent to the client is a transmission queue, SYSTEM.MQTT.TRANSMIT.QUEUE. The IBM MQ Telemetry service forwards messages on SYSTEM.MQTT.TRANSMIT.QUEUE to MQTT clients, using the target queue manager name as the key to a specific client.

The telemetry (MQXR) service opens the transmission queue using ClientIdentifier as the queue manager name. The telemetry (MQXR) service passes the object handle of the queue to the MQSUB call, to forward publications that match the client subscription. In the object name resolution, the ClientIdentifier is created as the remote queue manager name, and the transmission queue must resolve to SYSTEM.MQTT.TRANSMIT.QUEUE. Using standard IBM MQ object name resolution, *ClientIdentifier* is resolved as follows; see Table 16 on page 338.

1. *ClientIdentifier* matches nothing.

   ClientIdentifier is a remote queue manager name. It does not match the local queue manager name, a queue manager alias, or a transmission queue name.

   The queue name is not defined. Currently, the telemetry (MQXR) service sets SYSTEM.MQTT.PUBLICATION.QUEUE as the name of the queue. An MQTT v3 client does not support queues, so the resolved queue name is ignored by the client.

   The local queue manager property, Default transmission queue, name must be set to SYSTEM.MQTT.TRANSMIT.QUEUE, so that the publication is put on SYSTEM.MQTT.TRANSMIT.QUEUE to be sent to the client.

2. *ClientIdentifier* matches a queue manager alias named *ClientIdentifier*.

   ClientIdentifier is a remote queue manager name. It matches the name of a queue manager alias.

   The queue manager alias must be defined with *ClientIdentifier* as the remote queue manager name.

By setting the transmission queue name in the queue manager alias definition it is not necessary for the default transmission to be set to SYSTEM.MQTT.TRANSMIT.QUEUE.

Table 16. Name resolution of an MQTT queue manager alias

	Input		Output		
ClientIdentifier	**Queue manager name**	**Queue name**	**Queue manager name**	**Queue name**	**Transmission queue**
Matches nothing	*ClientIdentifier*	*undefined*	*ClientIdentifier*	*undefined*	Default transmission queue. SYSTEM.MQTT.TRANSMIT.QUEUE
Matches a queue manager alias named *ClientIdentifier*	*ClientIdentifier*	*undefined*	*ClientIdentifier*	*undefined*	SYSTEM.MQTT.TRANSMIT.QUEUE

For further information about name resolution, see Name resolution.

Any IBM MQ program can publish to the same topic. The publication is sent to its subscribers, including MQTT v3 clients that have a subscription to the topic.

If an administrative topic is created in a cluster, with the attribute CLUSTER(*clusterName*), any application in the cluster can publish to the client; for example:

```
echo DEFINE TOPIC('MQTTExamples') TOPICSTR('MQTT Examples') CLUSTER(MQTT) REPLACE | runmqsc qMgr
```

Note: Do not give SYSTEM.MQTT.TRANSMIT.QUEUE a cluster attribute.

MQTT client subscribers and publishers can connect to different queue managers. The subscribers and publishers can be part of the same cluster, or connected by a publish/subscribe hierarchy. The publication is delivered from the publisher to the subscriber using IBM MQ.

Sending a message to a client directly

An alternative to a client creating a subscription and receiving a publication that matches the subscription topic, send a message to an MQTT v3 client directly. MQTT V3 client applications cannot send messages directly, but other application, such as IBM MQ applications, can.

The IBM MQ application must know the ClientIdentifier of the MQTT v3 client. As MQTT v3 clients do not have queues, the target queue name is passed to the MQTT v3 application client messageArrived method as a topic name. For example, in an MQI program, create an object descriptor with the client as the ObjectQmgrName:

```
MQOD.ObjectQmgrName = ClientIdentifier ;
MQOD.ObjectName = name ;
```

If the application is written using JMS, create a point-to-point destination; for example:

JM 3.0

```
jakarta.jms.Destination jmsDestination =
(jakarta.jms.Destination)jmsFactory.createQueue
("queue://ClientIdentifier/name");
```

JMS 2.0

```
javax.jms.Destination jmsDestination =
(javax.jms.Destination)jmsFactory.createQueue
("queue://ClientIdentifier/name");
```

To send an unsolicited message to an MQTT client use a remote queue definition. The remote queue manager name must resolvedto the `ClientIdentifier` of the client. The transmission queue must resolve to `SYSTEM.MQTT.TRANSMIT.QUEUE` ; see Table 17 on page 339. The remote queue name can be anything. The client receives it as a topic string.

Table 17. Name resolution of an MQTT client remote queue definition

Input		Output		
Queue name	**Queue manager name**	**Queue name**	**Queue manager name**	**Transmission queue**
Name of remote queue definition	Blank or local queue manager name	Remote queue name used as a topic string	*ClientIdentifier*	SYSTEM.MQTT.TRANSMIT.QUEUE

If the client is connected, the message is sent directly to the MQTT client, which calls the `messageArrived` method; see `messageArrived` method.

If the client has disconnected with a persistent session, the message is stored in `SYSTEM.MQTT.TRANSMIT.QUEUE` ; see MQTT stateless and stateful sessions. It is forwarded to the client when the client reconnects to the session again.

If you send a non-persistent message it is sent to the client with `at most once` quality of service, `QoS=0`. If you send a persistent message directly to a client, by default, it is sent with `exactly once` quality of service, `QoS=2`. As the client might not have a persistence mechanism, the client can reduce the quality of service that it accepts for messages sent directly. To reduce the quality of service for messages sent directly to a client, make a subscription to the topic `DEFAULT.QoS`. Specify the maximum quality of service the client can support.

ALW MQTT client identification, authorization, and authentication

The telemetry (MQXR) service publishes, or subscribes to, IBM MQ topics on behalf of MQTT clients, using MQTT channels. The IBM MQ administrator configures the MQTT channel identity that is used for IBM MQ authorization. The administrator can define a common identity for the channel, or use the `Username` or `ClientIdentifier` of a client connected to the channel.

The telemetry (MQXR) service can authenticate the client using the `Username` supplied by the client, or by using a client certificate. The `Username` is authenticated using a password provided by the client.

To summarize: Client identification is the selection of the client identity. Depending on the context, the client is identified by the `ClientIdentifier`, `Username`, a common client identity created by the administrator, or a client certificate. The client identifier used for authenticity checking does not have to be the same identifier that is used for authorization.

MQTT client programs set the `Username` and `Password` that are sent to the server using an MQTT channel. They can also set the TLS properties that are required to encrypt and authenticate the connection. The administrator decides whether to authenticate the MQTT channel, and how to authenticate the channel.

To authorize an MQTT client to access IBM MQ objects, authorize the `ClientIdentifier`, or `Username` of the client, or authorize a common client identity. To permit a client to connect to IBM MQ, authenticate the `Username`, or use a client certificate. Configure JAAS to authenticate the `Username`, and configure TLS to authenticate a client certificate.

If you set a `Password` at the client, either encrypt the connection using VPN, or configure the MQTT channel to use TLS, to keep the password private.

It is difficult to manage client certificates. For this reason, if the risks associated with password authentication are acceptable, password authentication is often used to authenticate clients.

If there is a secure way to manage and store the client certificate it is possible to rely on certificate authentication. However, it is rarely the case that certificates can be managed securely in the types of environments that telemetry is used in. Instead, the authentication of devices using client certificates is complemented by authenticating client passwords at the server. Because of the additional complexity, the use of client certificates is restricted to highly sensitive applications. The use of two forms of authentication is called two-factor authentication. You must know one of the factors, such as a password, and have the other, such as a certificate.

In a highly sensitive application, such as a chip-and-pin device, the device is locked down during manufacture to prevent tampering with the internal hardware and software. A trusted, time-limited, client certificate is copied to the device. The device is deployed to the location where it is to be used. Further authentication is performed each time the device is used, either using a password, or another certificate from a smart card.

ALW MQTT client identity and authorization

Use the client ID, `Username`, or a common client identity for authorization to access IBM MQ objects.

The IBM MQ administrator has three choices for selecting the identity of the MQTT channel. The administrator makes the choice when defining or modifying the MQTT channel used by the client. The identity is used to authorize access to IBM MQ topics. The choice is made in the following order:

1. The client ID (see USECLNTID).
2. An identity the administrator provides for the channel (the `MCAUSER` of the channel. See MCAUSER).
3. If neither of the previous choices applies, the `Username` passed from the MQTT client (`Username` is an attribute of the `MqttConnectOptions` class. It must be set before the client connects to the service. Its default value is null).

Avoid trouble: The identity chosen by this process is thereafter referred to, for example by the DISPLAY CHSTATUS (MQTT) command, as the MCAUSER of the client. Be aware that this is not necessarily the same identity as the MCAUSER of the channel that is referred to in choice (2).

Use the IBM MQ **setmqaut** command to select which objects, and which actions, are authorized to be used by the identity associated with the MQTT channel. For example, the following code authorizes a channel identity `MQTTClient`, provided by the administrator of queue manager `QM1`:

```
 setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

ALW *Authorizing MQTT clients to access IBM MQ objects*

Follow these steps to authorize MQTT clients to publish and subscribe to IBM MQ Objects. The steps follow four alternative access control patterns.

Before you begin

MQTT clients are authorized to access objects in IBM MQ by being assigned an identity when they connect to a telemetry channel. The IBM MQ Administrator configures the telemetry channel using IBM MQ Explorer to give a client one of three types of identity:

1. `ClientIdentifier`
2. `Username`
3. A name the administrator assigns to the channel.

Whichever type is used, the identity must be defined to IBM MQ as a principal by the installed authorization service. The default authorization service on Windows or Linux is called the Object Authority Manager (OAM). If you are using the OAM, the identity must be defined as a user ID.

Use the identity to give a client, or collection of clients, permission to publish or subscribe to topics defined in IBM MQ. If an MQTT client has subscribed to a topic, use the identity to give it permission to receive the resulting publications.

It is hard to manage a system with tens of thousands of MQTT clients, each requiring individual access permissions. One solution is to define common identities, and associate individual MQTT clients with one of the common identities. Define as many common identities as you require to define different combinations of permissions. Another solution is to write your own authorization service that can deal more easily with thousands of users than the operating system.

You can combine MQTT clients into common identities in two ways, using the OAM:

1. Define multiple telemetry channels, each with a different user ID that the administrator allocates using IBM MQ Explorer. Clients connecting using different TCP/IP port numbers are associated with different telemetry channels, and are assigned different identities.
2. Define a single telemetry channel, but have each client select a `Username` from a small set of user IDs. The administrator configures the telemetry channel to select the client `Username` as its identity.

In this task, the identity of the telemetry channel is called *`mqttUser`*, regardless of how it is set. If collections of clients use different identities, use multiple *`mqttUsers`*, one for each collection of clients. As the task uses the OAM, each *`mqttUser`* must be a user ID.

About this task

In this task, you have a choice of four access control patterns that you can tailor to specific requirements. The patterns differ in their granularity of access control.

- "No access control" on page 341
- "Coarse-grained access control" on page 341
- "Medium-grained access control" on page 342
- "Fine-grained access control" on page 342

The result of the models is to assign *`mqttUsers`* sets of permissions to publish and subscribe to IBM MQ, and receive publications from IBM MQ.

No access control
MQTT clients are given IBM MQ administrative authority, and can perform any action on any object.

Procedure

1. Create a user ID *`mqttUser`* to act as the identity of all MQTT clients.
2. Add *`mqttUser`* to the `mqm` group; see Adding a user to a group on Windows , or Creating and managing groups on Linux

Coarse-grained access control
MQTT clients have authority to publish and subscribe, and to send messages to MQTT clients. They do not have authority to perform other actions, or to access other objects.

Procedure

1. Create a user ID *`mqttUser`* to act as the identity of all MQTT clients.
2. Authorize *`mqttUser`* to publish and subscribe to all topics and to send publications to MQTT clients.

```
setmqaut -m qMgr -t topic -n SYSTEM.BASE.TOPIC -p mqttUser -all +pub +sub
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqttUser -all +put
```

Medium-grained access control
MQTT clients are divided into different groups to publish and subscribe to different sets of topics, and to send messages to MQTT clients.

Procedure

1. Create multiple user IDs, *mqttUsers*, and multiple administrative topics in the publish/subscribe topic tree.
2. Authorize different *mqttUsers* to different topics.

```
setmqaut -m qMgr -t topic -n topic1 -p mqttUserA -all +pub +sub
setmqaut -m qMgr -t topic -n topic2 -p mqttUserB -all +pub +sub
```

3. Create a group *mqtt*, and add all *mqttUsers* to the group.
4. Authorize *mqtt* to send topics to MQTT clients.

```
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
```

Fine-grained access control
MQTT clients are incorporated into an existing system of access control, that authorizes groups to perform actions on objects.

About this task

A user ID is assigned to one or more operating system groups depending on the authorizations it requires. If IBM MQ applications are publishing and subscribing to the same topic space as MQTT clients, use this model. The groups are referred to as `Publish X`, `Subscribe Y`, and `mqtt`

`Publish X`
Members of `Publish X` groups can publish to *topicX*.

`Subscribe Y`
Members of `Subscribe Y` groups can subscribe to *topicY*.

`mqtt`
Members of the *mqtt* group can send publications to MQTT clients.

Procedure

1. Create multiple groups, `Publish X` and `Subscribe Y` that are allocated to multiple administrative topics in the publish/subscribe topic tree.
2. Create a group `mqtt`.
3. Create multiple user IDs, *mqttUsers*, and add the users to any of the groups, depending on what they are authorized to do.
4. Authorize different `Publish X` and `Subscribe X` groups to different topics, and authorize the *mqtt* group to send messages to MQTT clients.

```
setmqaut -m qMgr -t topic -n topic1 -p Publish X -all +pub
setmqaut -m qMgr -t topic -n topic1 -p Subscribe X -all +pub +sub
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
```

ALW MQTT client authentication using a password

Authenticate the `Username` using the client password. You can authenticate the client using a different identity to the identity used to authorize the client to publish and subscribe to topics.

The telemetry (MQXR) service uses JAAS to authenticate the client `Username`. JAAS uses the `Password` supplied by the MQTT client.

The IBM MQ administrator decides whether to authenticate the `Username`, or not to authenticate at all, by configuring the MQTT channel a client connects to. Clients can be assigned to different channels,

and each channel can be configured to authenticate its clients in different ways. Using JAAS, you can configure which methods must authenticate the client, and which can optionally authenticate the client.

The choice of identity for authentication does not affect the choice of identity for authorization. You might want to set up a common identity for authorization for administrative convenience, but authenticate each user to use that identity. The following procedure outlines the steps to authenticate individual users to use a common identity:

1. The IBM MQ administrator sets the MQTT channel identity to any name, such as `MQTTClientUser`, using IBM MQ Explorer.
2. The IBM MQ administrator authorizes `MQTTClient` to publish and subscribe to any topic:

```
 setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

3. The MQTT client application developer creates an `MqttConnectOptions` object and sets `Username` and `Password` before connecting to the server.
4. The security developer creates a JAAS `LoginModule` to authenticate the `Username` with the `Password` and includes it in the JAAS configuration file.
5. The IBM MQ administrator configures the MQTT channel to authenticate the `UserName` of the client using JAAS.

ALW MQTT client authentication using TLS

Connections between the MQTT client and the queue manager are always initiated by the MQTT client. The MQTT client is always the SSL client. Client authentication of the server and server authentication of the MQTT client are both optional.

By providing the client with a private signed digital certificate, you can authenticate the MQTT client to IBM MQ. The IBM MQ Administrator can force MQTT clients to authenticate themselves to the queue manager using TLS. You can only request client authentication as part of mutual authentication.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

Client authentication using TLS relies upon the client having a secret. The secret is the private key of the client in the case of a self-signed certificate, or a key provided by a certificate authority. The key is used to sign the digital certificate of the client. Anyone in possession of the corresponding public key can verify the digital certificate. Certificates can be trusted, or if they are chained, traced back through a certificate chain to a trusted root certificate. Client verification sends all the certificates in the certificate chain provided by the client to the server. The server checks the certificate chain until it finds a certificate it trusts. The trusted certificate is either the public certificate generated from a self-signed certificate, or a root certificate typically issued by a certificate authority. As a final, optional, step the trusted certificate can be compared with a "live" certificate revocation list.

The trusted certificate might be issued by a certificate authority and already included in the JRE certificate store. It might be a self-signed certificate, or any certificate that has been added to the telemetry channel keystore as a trusted certificate.

Note: The telemetry channel has a combined keystore/truststore that holds both the private keys to one or more telemetry channels, and any public certificates needed to authenticate clients. Because an SSL channel must have a keystore, and it is the same file as the channel truststore, the JRE certificate store is never referenced. The implication is that if authentication of a client requires a CA root certificate, you must place the root certificate in the keystore for the channel, even if the CA root certificate is already in the JRE certificate store. The JRE certificate store is never referenced.

Think about the threats that client authentication is intended to counter, and the roles the client and server play in countering the threats. Authenticating the client certificate alone is insufficient to prevent unauthorized access to a system. If someone else has got hold of the client device, the client device is

not necessarily acting with the authority of the certificate holder. Never rely on a single defense against unwanted attacks. At least use a two-factor authentication approach and supplement possession of a certificate with knowledge of private information. For example, use JAAS, and authenticate the client using a password issued by the server.

The primary threat to the client certificate is that it gets into the wrong hands. The certificate is held in a password protected keystore at the client. How does it get placed in the keystore? How does the MQTT client get the password to the keystore? How secure is the password protection? Telemetry devices are often easy to remove, and then can be hacked in private. Must the device hardware be tamper-proof? Distributing and protecting client-side certificates is recognized to be hard; it is called the key-management problem.

A secondary threat is that the device is misused to access servers in unintended ways. For example, if the MQTT application is tampered with, it might be possible to use a weakness in the server configuration using the authenticated client identity.

To authenticate an MQTT client using SSL, configure the telemetry channel, and the client.

Related concepts

"IBM MQ Telemetry channel configuration for MQTT client authentication using TLS" on page 344
The IBM MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. TLS channels are configured with passphrase protected access to key files. If a TLS channel is defined with no passphrase or key file, the channel does not accept TLS connections.

MQTT client configuration for client authentication using TLS

ALW *IBM MQ Telemetry channel configuration for MQTT client authentication using TLS*

The IBM MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. TLS channels are configured with passphrase protected access to key files. If a TLS channel is defined with no passphrase or key file, the channel does not accept TLS connections.

Set the property, `com.ibm.mq.MQTT.ClientAuth` of a TLS telemetry channel to `REQUIRED` to force all clients connecting on that channel to provide proof that they have verified digital certificates. The client certificates are authenticated using certificates from certificate authorities, leading to a trusted root certificate. If the client certificate is self-signed, or is signed by a certificate that is from a certificate authority, the publicly signed certificates of the client, or certificate authority, must be stored securely at the server.

Place the publicly signed client certificate or the certificate from the certificate authority in the telemetry channel keystore. At the server, publicly signed certificates are stored in the same key file as privately signed certificates, rather than in a separate truststore.

The server verifies the signature of any client certificates it is sent using all the public certificates and cipher suites it has. The server verifies the key chain. The queue manager can be configured to test the certificate against the certificate revocation list. The queue manager revocation namelist property is `SSLCRLNL`.

If any of the certificates a client sends is verified by a certificate in the server keystore, then the client is authenticated.

The IBM MQ administrator can configure the same telemetry channel to use JAAS to check the `UserName` or `ClientIdentifier` of the client with the client `Password`.

You can use the same keystore for multiple telemetry channels.

Verification of at least one digital certificate in the password protected client keystore on the device authenticates the client to the server. The digital certificate is only used for authentication by IBM MQ. It is not used to verify the TCP/IP address of the client, or set the identity of the client for authorization or accounting. The identity of the client adopted by the server is either the `Username` or `ClientIdentifier` of the client, or an identity created by the IBM MQ administrator.

You can also use TLS cipher suites for client authentication. If you plan to use SHA-2 cipher suites, see “System requirements for using SHA-2 cipher suites with MQTT channels” on page 346.

Related concepts

“IBM MQ Telemetry channel configuration for channel authentication using TLS” on page 345
The IBM MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. TLS channels are configured with passphrase protected access to key files. If a TLS channel is defined with no passphrase or key file, the channel does not accept TLS connections.

CipherSpecs and CipherSuites

Related reference

DEFINE CHANNEL (MQTT)

ALTER CHANNEL (MQTT)

ALW IBM MQ Telemetry channel authentication using TLS

Connections between the MQTT client and the queue manager are always initiated by the MQTT client. The MQTT client is always the SSL client. Client authentication of the server and server authentication of the MQTT client are both optional.

The client always attempts to authenticate the server, unless the client is configured to use a CipherSpec that supports anonymous connection. If the authentication fails, then the connection is not established.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

Server authentication using SSL authenticates the server to which you are about to send confidential information to. The client performs the checks matching the certificates sent from the server, against certificates placed in its truststore, or in its JRE `cacerts` store.

The JRE certificate store is a JKS file, `cacerts`. It is located in *JRE_LOCATION*`\lib\security\`. It is installed with the default password `changeit`. You can either store certificates you trust in the JRE certificate store, or in the client truststore. You cannot use both stores. Use the client truststore if you want to keep the public certificates the client trusts separate from certificates other Java applications use. Use the JRE certificate store if you want to use a common certificate store for all Java applications running on the client. If you decide to use the JRE certificate store review the certificates it contains, to make sure you trust them.

You can modify the JSSE configuration by supplying a different trust provider. You can customize a trust provider to perform different checks on a certificate. In some OGSi environments that have used the MQTT client, the environment provides a different trust provider.

To authenticate the telemetry channel using TLS, configure the server, and the client.

ALW IBM MQ Telemetry channel configuration for channel authentication using TLS

The IBM MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. TLS channels are configured with passphrase protected access to key files. If a TLS channel is defined with no passphrase or key file, the channel does not accept TLS connections.

Store the digital certificate of the server, signed with its private key, in the keystore that the telemetry channel is going to use at the server. Store any certificates in its key chain in the keystore, if you want to transmit the key chain to the client. Configure the telemetry channel using IBM MQ explorer to use TLS. Provide it with the path to the keystore, and the passphrase to access the keystore. If you do not set the TCP/IP port number of the channel, the TLS telemetry channel port number defaults to 8883.

System requirements for using SHA-2 cipher suites with MQTT channels

You can also use TLS cipher suites for channel authentication. If you use a version of Java that supports SHA-2 cipher suites, you can use these suites to secure your MQTT (telemetry) channels and client apps.

For IBM MQ 8.0, which includes the telemetry (MQXR) service, the minimum Java version is Java 7 from IBM, SR6. SHA-2 cipher suites are supported by default in Java 7 from IBM, SR4 onwards. You can therefore use SHA-2 cipher suites with the telemetry (MQXR) service to secure your MQTT (telemetry) channels.

If you are running an MQTT client with a different JRE, you need to ensure that it also supports the SHA-2 cipher suites.

Switching from using a CMS key repository to a PKCS #12 key repository

From IBM MQ 9.4.0, CMS key repositories and stash files are not supported with AMQP and MQTT channels that use SSL/TLS. Use PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system instead. You can create a PKCS #12 key repository by using the following command:

```
runmqakm -keydb -create -db filename.p12 -pw password -type pkcs12
```

This command creates a PKCS #12 key repository file named *filename*.p12 that is secured with the specified password.

Updating Java properties for enabling FIPS 140-3 cryptography

From IBM MQ 10.0.0, with the JRE upgrade to IBM Semeru Runtime 21, only FIPS 140-3 is supported for FIPS configuration. For the MQTT protocol, FIPS 140-3 is supported only on the following platforms:

- AIX
- Windows systems
- Linux on Power® Systems
- Linux for x86-64

To enable FIPS 140-3 cryptography for the MQTT channels, the following updates are required:

- Set the **SSLFIPS** attribute to YES in the queue manager.
- Update the `java.properties` file with the following:

```
-Dsemeru.fips=true
-Dsemeru.customprofile=OpenJCEPlusFIPS
```

You can also use the **setmqtt** property to update the `java.properties` by using the following command:

```
setmqtt properties -m -k -Dsemeru.fips -v true -k -Dsemeru.customprofile -v OpenJCEPlusFIPS.
```

If **SSLFIPS** is enabled in the queue manager without setting both the **Dsemeru** Java properties, the MQTT service does not start and the following error message is logged:

```
AMQXR2125E: SSLFIPS property is set to YES in the Queue Manager configuration but the FIPS 140-3
JVM system properties -Dsemeru.fips=true and -Dsemeru.customprofile=OpenJCEPlusFIPS are not set
in the java.properties file.
```

If the Java properties are set but **SSLFIPS** is not enabled, the MQTT service does not start and the following error message is logged:

```
AMQXR2126E: The FIPS 140-3 JVM system properties -Dsemeru.fips=true and
-Dsemeru.customprofile=OpenJCEPlusFIPS are set in the java.properties file but the SSLFIPS
property is not set in the Queue Manager configuration.
```

For more information, see setmqxr (set MQTT server properties).

FIPS 140-3 support not available on Linux s390

V 10.0.0

The MQTT service is available on the Linux s390 architecture but this system does not support FIPS 140-3. On this platform, the MQTT channel can still use FIPS-compliant cipher without enabling FIPS configuration. If FIPS is enabled, the MQTT service will not start and a general security exception is logged in the MQTT error log. The exception includes an error message.

```
AMQXR2127E: FIPS 140-3 support is not available on this operating system.
```

Allowing the MQTT service to start without FIPS support

V 10.0.0

To allow the MQTT service to start when FIPS is enabled but not supported on the platform, you can use the **com.ibm.mq.MQXR.AllowStartupWithoutFIPS** property. You configure this property in the `java.properties` file. The service starts, but an error message is logged indicating that FIPS is not supported. This feature is important for events where the queue manager sets **SSLFIPS** to YES as other channels require FIPS to be enabled. Disabling **SSLFIPS** in such cases can negatively impact those channels. By using this property, MQTT services can continue to run without compromising the broader FIPS configuration.

Related concepts

IBM MQ Telemetry (MQXR) service

“IBM MQ Telemetry channel configuration for MQTT client authentication using TLS” on page 344

The IBM MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. TLS channels are configured with passphrase protected access to key files. If a TLS channel is defined with no passphrase or key file, the channel does not accept TLS connections.

CipherSpecs and CipherSuites

Related reference

DEFINE CHANNEL (MQTT)

ALTER CHANNEL (MQTT)

Example MQTT channel configuration using TLS authentication

This example walks you through an example of configuring an MQTT channel that uses TLS authentication. It configures a channel between MQTT and Mosquitto.

The example uses a Docker container for both IBM MQ on Red Hat Enterprise Linux and Mosquitto on CentOS, but applies to any type of server. (CentOS was used for Mosquitto because of registry entitlements.)

This example assumes that you are aware of the configuration changes described in “IBM MQ Telemetry channel configuration for channel authentication using TLS” on page 345.

Configure IBM MQ keystore and channel for one-way TLS

Important: From IBM MQ 9.4.0, CMS key repositories and stash files are not supported with AMQP and MQTT channels that use SSL/TLS. Use PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system instead.

Complete the following steps:

1. Create an IBM MQ PKCS #12 keystore:

   ```
   runmqakm -keydb -create -db mqtt.p12 -pw "passw0rd" -type p12
   ```

2. Create a personal certificate:

```
runmqakm -cert -create -db mqtt.p12 -pw "passw0rd" -size 2048 -dn "CN= mqm, OU=MQTest,
O=MQSupport, C=US" -sig_alg SHA256_WITH_RSA -label ibmwebspheremqmqm
```

You can use the following command to confirm the creation of the certificate:

```
runmqakm -cert -list -v -db mqtt.p12 -pw "passw0rd"
```

3. Create the MQTT channel by entering the following command at the runmqsc prompt:

```
DEFINE CHANNEL(MQTTDEMO) CHLTYPE(MQTT) BACKLOG(4096) PORT(8883) MCAUSER('mqm')
PROTOCOL(MQTTV311,MQTTV3,HTTP) SSLCAUTH(OPTIONAL) SSLCIPH('SSL_RSA_WITH_AES_256_CBC_SHA256')
SSLKEYR('/var/mqm/mqtt/mqtt.p12') SSLKEYP('passw0rd') TRPTYPE(TCP)
```

Note that the channel uses Java cipher mappings, see TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS.

4. Extract the certificate:

```
runmqakm -cert -extract -db mqtt.p12 -pw "passw0rd" -label ibmwebspheremqmqm -target
serverCert.pem
```

Install Mosquitto on CentOS in a Docker container

Complete the following steps to create a Docker container with Mosquitto running on CentOS:

1. `docker pull centos`
2. `docker run -it centos /bin/bash`
3. `dnf -y install epel-release`
4. `dnf -y install mosquitto`

Move signer certificate to Mosquitto

Complete the following steps to move the certificate that you created in IBM MQ to Mosquitto. These steps are run on the Docker host machine.

1. View the container IDs on Docker:

```
docker container ls
```

2. Copy the file from the docker container to your local system docker

```
cp MQ_Container_ID:/var/mqm/mqtt/serverCert.pem serverCert.pem
```

3. Copy the file from your local machine to the root directory on the centOS Machine:

```
docker cp serverCert.pem CentOS_ContainerID:/serverCert.pem
```

Publish with Mosquitto

Publish a test message on Mosquitto using the following command:

```
mosquitto_pub -h 172.17.0.2 --cafile serverCert.pem --insecure -p 8883 -i mosquittoClient -t
test -m 'test message' -d
```

The command arguments have the following meanings:

-h
The Red Hat Enterprise Linux host IP address (can be found using **nslookup**).

--cafile
The file containing the signer certificate.

--insecure
This option is specified because the example is using a self-signed certificate. Do not use this option when using real CA certificates.

-p
Port number.

-i
Client ID.

-t
The topic being published to.

-m
The message being published.

-d
Enable debug messages.

Configure the MQTT channel for mutual TLS authentication

Enter the following command to reconfigure the MQTT channel as SSLCAUTH(REQUIRED).

```
ALTER CHANNEL(MQTTDEMO) CHLTYPE(MQTT) SSLCAUTH(REQUIRED)
```

Create a key/certificate pair on Mosquitto server and add to IBM MQ

Enter the following commands to create the key/certificate pair on Mosquitto:

1. Use **openSSL** to create the key/certificate pair for Mosquitto:

   ```
   openssl req -x509 -newkey rsa:4096 -keyout mosquittoKey.pem -out mosquittoCert.pem -subj "/
   CN=Mosquitto"
   ```

2. List the container IDs for the containers:

   ```
   docker container ls
   ```

3. Copy the Mosquitto certificate to your local system docker:

   ```
   docker cp CentOS_ContainerID:mosquittoCert.pem .
   ```

4. Copy the Mosquitto certificate to IBM MQ:

   ```
   docker cp mosquittoCert.pem MQ_Container_ID:/var/mqm/mqtt
   ```

5. Add the certificate to the IBM MQ keystore:

   ```
   runmqakm -cert -add -db mqtt.p12 -pw "passw0rd" -file mosquittoCert.pem
   ```

6. Restart the MQTT channel.

Publish with Mosquitto and mutual authentication

Complete the following steps to publish with Mosquitto using mutual authentication.

1. The following command should successfully publish a test message:

   ```
   mosquitto_pub -h 172.17.0.2 --cafile serverCert.pem --insecure -p 8883 -i mosquittoClient -t
   test -m 'test message' -d --cert mosquittoCert.pem --key mosquittoKey.pem
   ```

2. The following command should fail to publish a test message and generate an error message because it does not send a personal certificate from Mosquitto:

   ```
   mosquitto_pub -h 172.17.0.2 --cafile serverCert.pem --insecure -p 8883 -i mosquittoClient -t
   test -m 'test message' -d /var/mqm/qmgrs/mqttDemoQM/errors/ mqxr_0.log
   ```

Related information

Managing keys and certificates

ALW Publication privacy on telemetry channels

The privacy of MQTT publications sent in either direction across telemetry channels is secured by using TLS to encrypt transmissions over the connection.

MQTT clients that connect to telemetry channels use TLS to secure the privacy of publications transmitted on the channel using symmetric key cryptography. Because the endpoints are not authenticated, you cannot trust channel encryption alone. Combine securing privacy with server or mutual authentication.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

For a typical configuration, which encrypts the channel and authenticates the server, consult “IBM MQ Telemetry channel authentication using TLS” on page 345.

Encrypting TLS connections without authenticating the server exposes the connection to man-in-the-middle attacks. Although the information you exchange is protected against eavesdropping, you do not know who you are exchanging it with. Unless you control the network, you are exposed to someone intercepting your IP transmissions, and masquerading as the endpoint.

You can create an encrypted TLS connection, without authenticating the server, by using a Diffie-Hellman key exchange CipherSpec that supports anonymous TLS. The master secret, shared between the client and server, and used to encrypt TLS transmissions, is established without exchanging a privately signed server certificate.

Because anonymous connections are insecure, most TLS implementations do not default to using anonymous CipherSpecs. If a client request for TLS connection is accepted by a telemetry channel, the channel must have a keystore protected by a passphrase. By default, since TLS implementations do not use anonymous CipherSpecs, the keystore must contain a privately signed certificate that the client can authenticate.

If you use anonymous CipherSpecs, the server keystore must exist, but it need not contain any privately signed certificates.

Another way to establish an encrypted connection is to replace the trust provider at the client with your own implementation. Your trust provider would not authenticate the server certificate, but the connection would be encrypted.

Attention: When using TLS with MQTT you can use large messages, however, there might be a possible performance impact when doing so. MQTT is optimized for processing small messages (typically between 1KB and 1MB in size).

ALW TLS configuration of MQTT Java clients and telemetry channels

Configure TLS to authenticate the telemetry channel and the MQTT Java client, and encrypt the transfer of messages between them. MQTT Java clients use Java Secure Socket Extension (JSSE) to connect telemetry channels using TLS. As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

You can configure the connection between a Java MQTT client and a telemetry channel to use the TLS protocol over TCP/IP. What is secured depends on how you configure TLS to use JSSE. Starting with the most secured configuration, you can configure three different levels of security:

1. Permit only trusted MQTT clients to connect. Connect an MQTT client only to a trusted telemetry channel. Encrypt messages between the client and the queue manager; see “MQTT client authentication using TLS” on page 343

2. Connect an MQTT client only to a trusted telemetry channel. Encrypt messages between the client and the queue manager; see “IBM MQ Telemetry channel authentication using TLS” on page 345.
3. Encrypt messages between the client and the queue manager; see “Publication privacy on telemetry channels” on page 350.

JSSE configuration parameters

Modify JSSE parameters to alter the way an TLS connection is configured. The JSSE configuration parameters are arranged into three sets:

1. IBM MQ Telemetry channel
2. MQTT Java client
3. JRE

Configure the telemetry channel parameters using IBM MQ Explorer. Set the MQTT Java Client parameters in the `MqttConnectionOptions.SSLProperties` attribute. Modify JRE security parameters by editing files in the JRE security directory on both the client and server.

IBM MQ Telemetry channel

Set all the telemetry channel TLS parameters using IBM MQ Explorer.

ChannelName

`ChannelName` is a required parameter on all channels.

The channel name identifies the channel associated with a particular port number. Name channels to help you administer sets of MQTT clients.

PortNumber

`PortNumber` is an optional parameter on all channels. It defaults to 1883 for TCP channels, and 8883 for TLS channels.

The TCP/IP port number associated with this channel. MQTT clients are connected to a channel by specifying the port defined for the channel. If the channel has TLS properties, the client must connect using the TLS protocol; for example:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

`KeyFileName` is a required parameter for TLS channels. It must be omitted for TCP channels.

`KeyFileName` is the path to the Java keystore containing digital certificates that you provide. Use JKS, JCEKS or PKCS12 as the type of keystore on the server.

Identify the keystore type by using one of the following file extensions:

```
.jks
.jceks
.p12
.pkcs12
```

A keystore with any other file extension is assumed to be a JKS keystore.

You can combine one type of keystore at the server with other types of keystore at the client.

Place the private certificate of the server in the keystore. The certificate is known as the server certificate. The certificate can be self-signed, or part of a certificate chain that is signed by a signing authority.

If you are using a certificate chain, place the associated certificates in the server keystore.

The server certificate, and any certificates in its certificate chain, are sent to clients to authenticate the identity of the server.

If you have set `ClientAuth` to `Required`, the keystore must contain any certificates necessary to authenticate the client. The client sends a self-signed certificate, or a certificate chain, and the client is authenticated by the first verification of this material against a certificate in the keystore. Using a certificate chain, one certificate can verify many clients, even if they are issued with different client certificates.

PassPhrase

`PassPhrase` is a required parameter for TLS channels. It must be omitted for TCP channels.

The passphrase is used to protect the keystore.

ClientAuth

`ClientAuth` is an optional TLS parameter. It defaults to no client authentication. It must be omitted for TCP channels.

Set `ClientAuth` if you want the telemetry (MQXR) service to authenticate the client, before permitting the client to connect to the telemetry channel.

If you set `ClientAuth`, the client must connect to the server using TLS, and authenticate the server. In response to setting `ClientAuth`, the client sends its digital certificate to the server, and any other certificates in its keystore. Its digital certificate is known as the client certificate. These certificates are authenticated against certificates held in the channel keystore, and in the JRE `cacerts` store.

CipherSuite

`CipherSuite` is an optional TLS parameter. It defaults to try all the enabled CipherSpecs. It must be omitted for TCP channels.

If you want to use a particular CipherSpec, set `CipherSuite` to the name of the CipherSpec that must be used to establish the TLS connection.

The IBM MQ Telemetry service and MQTT client negotiate a common CipherSpec, or compatible CipherSpecs, from all the CipherSpecs that are enabled at each end. If a specific CipherSpec is specified at one end of the connection, it must be compatible with the CipherSpec at the other end.

Install additional ciphers by adding additional providers to JSSE.

FIPS

FIPS is an optional setting. By default it is not set.

Either in the properties panel of the queue manager, or using **runmqsc**, set **SSLFIPS**. **SSLFIPS** specifies whether only FIPS-certified algorithms are to be used.

Revocation namelist

Revocation namelist is an optional setting. By default it is not set.

Either in the properties panel of the queue manager, or using **runmqsc**, set SSLCRLNL. SSLCRLNL specifies a namelist of authentication information objects which are used to provide certificate revocation locations.

No other queue manager parameters that set TLS properties are used.

MQTT Java client

Set TLS properties for the Java client in `MqttConnectionOptions.SSLProperties` ; for example:

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

The names and values of specific properties are described in the `MqttConnectOptions` class. For links to client API documentation for the MQTT client libraries, see MQTT client programming reference.

`Protocol`

`Protocol` is optional.

The protocol is selected in negotiation with the telemetry server. If you require a specific protocol you can select one. If the telemetry server does not support the protocol the connection fails.

`ContextProvider`

`ContextProvider` is optional.

`KeyStore`

`KeyStore` is optional. Configure it if `ClientAuth` is set at the server to force authentication of the client.

Place the digital certificate of the client, signed using its private key, into the keystore. Specify the keystore path and password. The type and provider are optional. JKS is the default type, and IBMJCE is the default provider.

Specify a different keystore provider to reference a class that adds a new keystore provider. Pass the name of the algorithm used by the keystore provider to instantiate the `KeyManagerFactory` by setting the key manager name.

`TrustStore`

`TrustStore` is optional. You can place all the certificates you trust in the JRE `cacerts` store.

Configure the truststore if you want to have a different truststore for the client. You might not need to configure the truststore if the server is using a certificate issued by a well known CA that already has its root certificate stored in `cacerts`.

Add the publicly signed certificate of the server or the root certificate to the truststore, and specify the truststore path and password. JKS is the default type, and IBMJCE is the default provider.

Specify a different truststore provider to reference a class that adds a new truststore provider. Pass the name of the algorithm used by the truststore provider to instantiate the `TrustManagerFactory` by setting the trust manager name.

JRE

Other aspects of Java security that affect the behavior of TLS on both the client and server are configured in the JRE. The configuration files on Windows are in *`Java Installation Directory`*`\jre\lib\security`. If you are using the JRE shipped with IBM MQ the path is as shown in the following table:

Table 18. Filepaths by platform for JRE TLS configuration files

Platform	Filepath
Windows	*`WMQ Installation Directory`*`\java\jre\lib\security`
AIX and Linux platforms	*`WMQ Installation Directory`*`/java/jre64/jre/lib/security`

Well-known certificate authorities

The `cacerts` file contains the root certificates of well-known certificate authorities. The `cacerts` is used by default, unless you specify a truststore. If you use the `cacerts` store, or do not provide a truststore, you must review and edit the list of signers in `cacerts` to meet your security requirements.

You can use the **runmqktool** command to manage the `cacerts` certificates file. `cacerts` is a JKS file. Specify the parameter **-storetype jks** when the **runmqktool** command is used to manage the certificates file.

The default password for the `cacerts` file is `changeit`. Change the password by using the **runmqktool -storepasswd** command to secure the file.

Configuring security classes

Use the `java.security` file to register additional security providers and other default security properties.

Permissions

Use the `java.policy` file to modify the permissions granted to resources. `javaws.policy` grants permissions to `javaws.jar`

Encryption strength

Some JREs ship with reduced strength encryption. If you cannot import keys into keystores, reduced strength encryption might be the cause. If necessary, download strong, but limited jurisdiction files from IBM developer kits, Security information.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country. Check its regulations, and its policies concerning the import, possession, use, and re-export of encryption software, to determine if it is permitted.

Modify the trust provider to permit the client to connect to any server

The example illustrates how to add a trust provider and reference it from the MQTT client code. The example performs no authentication of the client or server. The resulting TLS connection is encrypted without being authenticated.

The code snippet in Figure 16 on page 354 sets the `AcceptAllProviders` trust provider and trust manager for the MQTT client.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

Figure 16. MQTT Client code snippet

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
private static final long serialVersionUID = 1L;
public AcceptAllProvider() {
super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
put("TrustManagerFactory.TrustAllCertificates",
AcceptAllTrustManagerFactory.class.getName());
}
```

Figure 17. AcceptAllProvider.java

```
protected static class AcceptAllTrustManagerFactory extends
javax.net.ssl.TrustManagerFactorySpi {
public AcceptAllTrustManagerFactory() {}
protected void engineInit(java.security.KeyStore keystore) {}
protected void engineInit(
javax.net.ssl.ManagerFactoryParameters parameters) {}
protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
}
```

Figure 18. AcceptAllTrustManagerFactory.java

```
protected static class AcceptAllX509TrustManager implements
javax.net.ssl.X509TrustManager {
public void checkClientTrusted(
java.security.cert.X509Certificate[] certificateChain,
String authType) throws java.security.cert.CertificateException {
report("Client authtype=" + authType);
for (java.security.cert.X509Certificate certificate : certificateChain) {
report("Accepting:" + certificate);
}
}
public void checkServerTrusted(
java.security.cert.X509Certificate[] certificateChain,
String authType) throws java.security.cert.CertificateException {
report("Server authtype=" + authType);
for (java.security.cert.X509Certificate certificate : certificateChain) {
report("Accepting:" + certificate);
}
}
public java.security.cert.X509Certificate[] getAcceptedIssuers() {
return new java.security.cert.X509Certificate[0];
}
private static void report(String string) {
System.out.println(string);
}
}
```

Figure 19. AcceptAllX509TrustManager.java

ALW CipherSuites supported by the IBM MQ Telemetry service from IBM MQ 10.0.0

You can use TLS with IBM MQ Telemetry (MQTT) to encrypt message traffic between the IBM MQ Telemetry service and MQTT clients. From IBM MQ 10.0.0 (or IBM MQ 9.4.4 for Continuous Delivery), you must ensure that your MQTT channels are configured to use CipherSuites that are supported by IBM Semeru Runtime 25.

From IBM MQ 10.0.0, the Java runtime environment (JRE) shipped with the product is updated from IBM Semeru Runtime 21 JRE to IBM Semeru Runtime 25. IBM Semeru Runtime 25 does not support FIPS 140-2 cryptography. If you are using any FIPS 140-2 CipherSpecs, you will need to move to using FIPS 140-3 CipherSpecs.

FIPS 140-2 support is still available in IBM SDK, Java Technology Edition 8 to IBM Semeru Runtime 21.

Table 19 on page 355 shows which CipherSuites the IBM MQ Telemetry service supports from IBM MQ 10.0.0.

Table 19. CipherSuites you can use with MQTT channels

CipherSuite	Protocol	FIPS 140-3 compatible
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	Yes
TLS_RSA_WITH_AES_128_CBC_SHA "1" on page 356	TLS 1.0	No
TLS_RSA_WITH_AES_128_CBC_SHA256 "1" on page 356	TLS 1.2	No

Table 19. CipherSuites you can use with MQTT channels (continued)		
CipherSuite	**Protocol**	**FIPS 140-3 compatible**
TLS_RSA_WITH_AES_128_GCM_SHA256 "1" on page 356	TLS 1.2	No
TLS_RSA_WITH_AES_256_CBC_SHA "1" on page 356	TLS 1.0	No
TLS_RSA_WITH_AES_256_CBC_SHA256 "1" on page 356	TLS 1.2	No
TLS_RSA_WITH_AES_256_GCM_SHA384 "1" on page 356	TLS 1.2	No
SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	No
SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	No
*TLS12	TLS 1.2	No
TLS_AES_128_GCM_SHA256	TLS 1.3	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	Yes
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	No
TLS_AES_128_CCM_SHA256	TLS 1.3	No
TLS_AES_128_CCM_8_SHA256	TLS 1.3	No
*ANY	Multiple	No
*TLS13	TLS 1.3	No
*TLS12ORHIGHER	TLS 1.2 and above	No
*TLS13ORHIGHER	TLS 1.3 and above	No

Notes:

1. The IBM Java 8 JRE removed support for RSA key exchange when operating in FIPS mode. TLS_RSA_* CipherSpecs are disabled by default in IBM Semeru Runtime 21 and IBM Semeru Runtime 25.

ALW IBM MQ Telemetry channel JAAS configuration

Configure JAAS to authenticate the `Username` sent by the client.

The IBM MQ administrator configures which MQTT channels require client authentication using JAAS. Specify the name of a JAAS configuration for each channel that is to perform JAAS authentication. Channels can all use the same JAAS configuration, or they can use different JAAS configurations. The configurations are defined in *`WMQData directory`*`\qmgrs\`*`qMgrName`*`\mqxr\jaas.config`.

The `jaas.config` file is organized by JAAS configuration name. Under each configuration name is a list of Login configurations; see "Sample jaas.config file" on page 357.

JAAS provides four standard Login modules. The standard NT and UNIX Login modules are of limited value.

JndiLoginModule
Authenticates against a directory service configured under JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Authenticates using Kerberos protocols.

NTLoginModule
Authenticates using the NT security information for the current user.

UnixLoginModule
Authenticates using the UNIX security information for the current user.

The problem with using `NTLoginModule` or `UnixLoginModule` is that the telemetry (MQXR) service runs with the `mqm` identity, and not the identity of the MQTT channel. `mqm` is the identity passed to `NTLoginModule` or `UnixLoginModule` for authentication, and not the identity of the client.

To overcome this problem, write your own Login module, or use the other standard Login modules. A sample `JAASLoginModule.java` is supplied with IBM MQ Telemetry. It is an implementation of the `javax.security.auth.spi.LoginModule` interface. Use it to develop your own authentication method.

Any new LoginModule classes you provide must be on the class path of the telemetry (MQXR) service. Do not place your classes in IBM MQ directories that are in the class path. Create your own directories, and define the whole class path for the telemetry (MQXR) service.

You can augment the class path used by the telemetry (MQXR) service by setting class path in the `service.env` file. `CLASSPATH` must be capitalized, and the class path statement can only contain literals. You cannot use variables in the CLASSPATH; for example `CLASSPATH=%CLASSPATH%` is incorrect. The telemetry (MQXR) service sets its own classpath. The CLASSPATH defined in `service.env` is added to it.

The telemetry (MQXR) service provides two callbacks that return the `Username` and the `Password` for a client connected to the MQTT channel. The `Username` and `Password` are set in the `MqttConnectOptions` object. See “Sample JAASLoginModule.Login() method” on page 357 for an example of how to access `Username` and `Password`.

Sample `jaas.config` file

An example of a JAAS configuration file with one named configuration, `MQXRConfig`

```
MQXRConfig {
samples.JAASLoginModule required debug=true;
//com.ibm.security.auth.module.NTLoginModule required;
//com.ibm.security.auth.module.Krb5LoginModule required
//         principal=principal@your_realm
//         useDefaultCcache=TRUE
//         renewTGT=true;
//com.sun.security.auth.module.NTLoginModule required;
//com.sun.security.auth.module.UnixLoginModule required;
//com.sun.security.auth.module.Krb5LoginModule required
//         useTicketCache="true"
//         ticketCache="${user.home}${/}tickets";
};
```

Sample `JAASLoginModule.Login()` method

An example of a JAAS Login module coded to receive the `Username` and `Password` provided by an MQTT client.

```
public boolean login()
throws javax.security.auth.login.LoginException {
javax.security.auth.callback.Callback[] callbacks =
new javax.security.auth.callback.Callback[2];
callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] = new javax.security.auth.callback.PasswordCallback(
"PasswordCallback", false);
try {
callbackHandler.handle(callbacks);
String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
.getName();
char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
.getPassword();
 // Accept everything.
if (true) {
loggedIn = true;
} else
throw new javax.security.auth.login.FailedLoginException("Login failed");

principal= new JAASPrincipal(username);
```

```
} catch (java.io.IOException exception) {
throw new javax.security.auth.login.LoginException(exception.toString());
} catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
throw new javax.security.auth.login.LoginException(exception.toString());
}

return loggedIn;
}
```

Related tasks

Resolving problem: JAAS login module not called by the IBM MQ Telemetry service

Related reference

AuthCallback MQXR class

ALW Administering an AMQP client

You can administer an AMQP client using IBM MQ Explorer or at a command line. Use the Explorer to configure channels and monitor AMQP clients that are connected to IBM MQ. Configure the security of AMQP clients using TLS and JAAS.

Before you begin

For information about installing AMQP on your platform, see Choosing what to install.

Administering using IBM MQ Explorer

Use the Explorer to configure AMQP channels and monitor the AMQP clients that are connected to IBM MQ. You can configure the security of AMQP clients using TLS and JAAS.

Administering using the command line

You can administer an AMQP client at the command line using MQSC commands.

ALW AMQP Service does not start automatically on queue manager startup

From IBM MQ 9.4.0, the default behavior of the setting of the **CONTROL** attribute for starting the AMQP service has changed. When creating and starting a new queue manager, the AMQP service does not automatically start as part of the queue manager startup process.

Between IBM MQ 9.0.4 and IBM MQ 9.4.0, the default behavior of the setting of the **CONTROL** attribute for starting the AMQP service is QMGR.

If the AMQP component was installed, the AMQP service started automatically, even if not utilized. To avoid the default startup of the AMQP Java Virtual Machine (JVM), you had two options:

- Not installing the AMQP component, or
- Changing the AMQP service **CONTROL** attribute to MANUAL after the queue manager is started.

From IBM MQ 9.4.0, newly created queue managers have reverted the setting of the **CONTROL** attribute of the SYSTEM.AMQP.SERVICE to MANUAL, which was the default setting prior to IBM MQ 9.0.4.

Migrated queue managers, if utilizing AMQP, continue to auto-start the service during queue manager startup. To determine if AMQP was utilized the following are checked:

- Existing AMQP channels
- Channel started messages in the AMQP error log.

Attention:

- This happens once only; the first time the queue manager is started after an upgrade.

- During migration, if the **CONTROL** attribute is changed from QMGR to MANUAL, an information message is logged in the IBM MQ error log to indicate the change. See Location of AMQP logs, error logs, and configuration files for more information.

If you want the AMQP service to auto start, change the service **CONTROL** attribute to QMGR and restart the queue manager. Subsequent restarts of the queue manager start the AMQP service.

Viewing IBM MQ objects in use by AMQP clients

You can view the different IBM MQ resources in use by AMQP clients, for example connections and subscriptions.

Connections

When the AMQP service is started new Hconns are created and connected to the queue manager. This pool of Hconns is used when AMQP clients publish messages. You can view the Hconns by using the **DISPLAY CONN** command. For example:

```
DISPLAY CONN(*) TYPE(CONN) WHERE (APPLDESC LK 'IBM MQ Advanced Message Queuing Protocol*')
```

This command also shows any client-specific Hconns. The Hconns that have a blank client ID attribute are the Hconns used in the pool

When an AMQP client connects to an AMQP channel, a new Hconn is connected to the queue manager. This Hconn is used to consume messages asynchronously for the subscriptions that the AMQP client has created. You can view the Hconn used by a particular AMQP client using the **DISPLAY CONN** command. For example:

```
DISPLAY CONN(*) TYPE(CONN) WHERE (CLIENTID EQ 'recv_abcd1234')
```

Subscriptions created by clients

When an AMQP client subscribes to a topic, a new IBM MQ subscription is created. The subscription name includes the following information:

- The name of the client. If the client joined a shared subscription, the name of the share is used
- The topic pattern that the client subscribed to
- A prefix. The prefix is `private` if the client created a non-shared subscription, or `share` if the client joined a shared subscription

To view the subscriptions in use by a particular AMQP client, run the **DISPLAY SUB** command and filter on the `private` prefix:

```
DISPLAY SUB(':private:*')
```

To view the shared subscriptions that are in use by multiple clients, run the **DISPLAY SUB** command and filter on the `share` prefix:

```
DISPLAY SUB(':share:*')
```

Because shared subscriptions can be used by multiple AMQP clients, you might want to view the clients currently consuming messages from the shared subscription. You can do this by listing the Hconns that currently have a handle open on the subscription queue. To view the clients currently using a share, complete the following steps:

1. Find the queue name that the shared subscription uses as a destination. For example:

```
DISPLAY SUB(':private:recv_e298452:public') DEST
   5 : DISPLAY SUB(':private:recv_e298452:public') DEST
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D312020202020202020707E0A565C2D0020)
```

```
   SUB(:private:recv_e298452:public)
   DEST(SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)
```

2. Run the **DISPLAY CONN** command to find the handles open on that queue:

```
DISPLAY CONN(*) TYPE(HANDLE) WHERE  (OBJNAME
EQ SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)
   21 : DISPLAY CONN(*) TYPE(HANDLE) WHERE(OBJNAME EQ
SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)

AMQ8276: Display Connection details.
   CONN(707E0A56642B0020)
   EXTCONN(414D5143514D31202020202020202020)
   TYPE(HANDLE)

   OBJNAME(SYSTEM.BASE.TOPIC)      OBJTYPE(TOPIC)

   OBJNAME(SYSTEM.MANAGED.DURABLE.560A7E7020002961)
   OBJTYPE(QUEUE)
```

3. For each of the handles, view the AMQP client ID that has the handle open:

```
DISPLAY CONN(707E0A56642B0020) CLIENTID
   23 : DISPLAY CONN(707E0A56642B0020) CLIENTID

AMQ8276: Display Connection details.
   CONN(707E0A56642B0020)
   EXTCONN(414D5143514D31202020202020202020)
   TYPE(CONN)
   CLIENTID(recv_8f02c9d)
DISPLAY CONN(707E0A565F290020) CLIENTID
   24 : DISPLAY CONN(707E0A565F290020) CLIENTID
AMQ8276: Display Connection details.
   CONN(707E0A565F290020)
   EXTCONN(414D5143514D31202020202020202020)
   TYPE(CONN)
   CLIENTID(recv_86d8888)
```

AMQP client identification, authorization, and authentication

Like other IBM MQ client applications, you can secure AMQP connections in a number of ways.

You can use the following security features to secure AMQP connections to IBM MQ:

- Channel authentication records
- Connection authentication
- Channel MCA user configuration
- IBM MQ authority definitions
- TLS connectivity

From a security perspective, establishing a connection consists of the following two steps:

- Deciding whether the connection should continue
- Deciding which IBM MQ identity the application assumes for later authority checks

The following information outlines different IBM MQ configurations and the steps that are worked through when an AMQP client tries to make a connection. Not all IBM MQ configurations use all of the steps described. For example, some configurations do not use TLS for connections inside the company firewall and some configurations use TLS but do not use client certificates for authentication. Many environments do not use custom or custom JAAS modules.

Establishing a connection

The following steps describe what happens when a connection is being established by an AMQP client. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. If the client opens a TLS connection to IBM MQ and provides a certificate, the queue manager attempts to validate the client certificate.
2. If the client provides user name and password credentials, an AMQP SASL frame is received by the queue manager and MQ CONNAUTH configuration is checked.
3. MQ channel authentication rules are checked (for example, whether the IP address and TLS certificate DN are valid)
4. Channel MCAUSER is asserted, unless channel authentication rules determine otherwise.
5. If a JAAS module has been configured, it is invoked
6. MQ CONNECT authority check applied to resulting MQ user ID.
7. Connection established with an assumed IBM MQ identity.

Publishing a message

The following steps describe what happens when a message is being published by an AMQP client. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. AMQP link attach frame arrives at queue manager. IBM MQ publish authority for the specified topic string is checked for the MQ user identity established during connection.
2. Message is published to specified topic string.

Subscribing to a topic pattern

The following steps describe what happens when an AMQP client subscribes to a topic pattern. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. AMQP link attach frame arrives at queue manager. IBM MQ subscribe authority for the specified topic pattern is checked for the MQ user identity established during connection.
2. Subscription is created.

AMQP client identity and authorization

Use the AMQP client ID, the AMQP user name, or a common client identity defined on the channel or in a channel authentication rule, for authorization to access IBM MQ objects.

The administrator makes the choice when defining or modifying the AMQP channel, by configuring the queue manager CONNAUTH setting, or by defining channel authentication rules. The identity is used to authorize access to IBM MQ topics. The choice is made based on the following:

1. The channel USECLNTID attribute.
2. The ADOPTCTX attribute of the queue manager CONNAUTH rule.
3. The MCAUSER attribute defined on the channel.
4. The USERSRC attribute of a matching channel authentication rule.

Avoid trouble: The identity chosen by this process is thereafter referred to, for example by the DISPLAY CHSTATUS (AMQP) command, as the MCAUSER of the client. Be aware that this is not necessarily the same identity as the MCAUSER of the channel that is referred to in choice (2).

Use the IBM MQ **setmqaut** command to select which objects, and which actions, are authorized to be used by the identity associated with the AMQP channel. For example, the following commands authorize a channel identity `AMQPClient`, provided by the administrator of queue manager `QM1`:

```
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p AMQPClient -all +pub +sub
```

and

```
setmqaut -m QM1 -t qmgr -p AMQPClient -all +connect
```

AMQP client authentication using a password

Authenticate the AMQP client user name using the client password. You can authenticate the client using a different identity from the identity used to authorize the client to publish and subscribe to topics.

The AMQP service can use MQ CONNAUTH or JAAS to authenticate the client user name. If one of these is configured, the password provided by the client is verified by the MQ CONNAUTH configuration or the JAAS module.

The following procedure outlines example steps to authenticate individual users against the local OS users and passwords and, if successful, adopt the common identity `AMQPUser`:

1. The IBM MQ administrator sets the AMQP channel MCAUSER identity to any name, such as `AMQPUser`, using IBM MQ Explorer.
2. The IBM MQ administrator authorizes `AMQPUser` to publish and subscribe to any topic:

   ```
   setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p AMQPUser -all +pub +sub +connect
   ```

3. The IBM MQ administrator configures an IDPWOS CONNAUTH rule to check the user name and password presented by the client. The CONNAUTH rule should set CHCKCLNT(REQUIRED) and ADOPTCTX(NO).

Note: You are recommended to use channel authentication rules and to set the MCAUSER channel attribute to a user who has no privileges, to allow more control over connections to the queue manager.

Publication privacy on channels

The privacy of AMQP publications sent in either direction across AMQP channels is secured by using TLS to encrypt transmissions over the connection.

AMQP clients that connect to AMQP channels use TLS to secure the privacy of publications transmitted on the channel using symmetric key cryptography. Because the endpoints are not authenticated, you cannot trust channel encryption alone. Combine securing privacy with server or mutual authentication.

As an alternative to using TLS, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect AMQP clients to AMQP channels using TCP/IP over the VPN network.

Encrypting TLS connections without authenticating the server exposes the connection to man-in-the-middle attacks. Although the information you exchange is protected against eavesdropping, you do not know who you are exchanging it with. Unless you control the network, you are exposed to someone intercepting your IP transmissions, and masquerading as the endpoint.

You can create an encrypted TLS connection, without authenticating the server, by using a Diffie-Hellman key exchange CipherSpec that supports anonymous TLS. The master secret, shared between the client and server, and used to encrypt TLS transmissions, is established without exchanging a privately signed server certificate.

Because anonymous connections are insecure, most TLS implementations do not default to using anonymous CipherSpecs. If a client request for TLS connection is accepted by an AMQP channel, the channel must have a keystore protected by a passphrase. By default, since TLS implementations do not use anonymous CipherSpecs, the keystore must contain a privately signed certificate that the client can authenticate.

If you use anonymous CipherSpecs, the server keystore must exist, but it need not contain any privately signed certificates.

Another way to establish an encrypted connection is to replace the trust provider at the client with your own implementation. Your trust provider would not authenticate the server certificate, but the connection would be encrypted.

ALW Configuring AMQP clients with TLS

You can configure AMQP clients to use TLS to protect data flowing across the network and to authenticate the identity of the queue manager the client connects to.

To use TLS for the connection from an AMQP client to an AMQP channel, you must ensure the queue manager has been configured to use TLS. Configuring TLS on queue managers describes how to configure the keystore that a queue manager reads TLS certificates from.

When the queue manager has been configured with a keystore, you must configure the TLS attributes on the AMQP channel that clients will connect to. AMQP channels have four attributes related to TLS configuration as follows:

SSLCAUTH
The **SSLCAUTH** attribute is used to specify whether the queue manager should require an AMQP client to present a client certificate to verify its identity.

SSLCIPH
The **SSLCIPH** attribute specifies the cipher the channel should use to encode data in the TLS flow.

- V 10.0.0 For the list of supported ciphers when using IBM MQ 10.0.0 and later, see "CipherSuites supported by the AMQP service from IBM MQ 10.0.0" on page 363.

From IBM MQ 9.4.0, AMQP channels support ANY* generic CipherSpecs. For more information about CipherSpecs, see Enabling CipherSpecs.

SSLPEER
The **SSLPEER** attribute is used to specify the distinguished name (DN) a client certificate must match if a connection is to be allowed.

CERTLABL
The **CERTLABL** attribute specifies the certificate the queue manager should present to the client. The queue manager's keystore can contain multiple certificates. This attribute allows you to specify the certificate to be used for connections to this channel. If no **CERTLABL** is specified, the certificate in the queue manager key repository with the label corresponding to the queue manager **CERTLABL** attribute is used.

When you have configured your AMQP channel with the TLS attributes, you must restart the AMQP service using the following command:

```
STOP SERVICE(SYSTEM.AMQP.SERVICE)
START SERVICE(SYSTEM.AMQP.SERVICE)
```

When an AMQP client connects to an AMQP channel protected by TLS, the client verifies the identity of the certificate presented by the queue manager. To do this, you must configure your AMQP client with a truststore containing the queue manager's certificate. The steps to do this vary depending on the AMQP client you are using. For more information about the various AMQP clients and APIs, see the respective AMQP client documentation.

Related reference
DEFINE CHANNEL (define a new channel)
STOP SERVICE (stop a service) on Multiplatforms
START SERVICE (start a service) on Multiplatforms

V 10.0.0 ALW CipherSuites supported by the AMQP service from IBM MQ 10.0.0

You can use TLS with IBM MQ AMQP channels to encrypt message traffic between the IBM MQ AMQP service and AMQP clients, and to authenticate the identity of the queue manager a client connects to. From IBM MQ 10.0.0 (or IBM MQ 9.4.4 for Continuous Delivery), you must ensure that your AMQP channels are configured to use CipherSuites that are supported by IBM Semeru Runtime 25.

V 10.0.0 V 10.0.0 From IBM MQ 10.0.0, the Java runtime environment (JRE) shipped with the product is updated from IBM Semeru Runtime 21 JRE to IBM Semeru Runtime 25. IBM Semeru Runtime 25 does not support

FIPS 140-2 cryptography. If you are using any FIPS 140-2 CipherSpecs, you will need to move to using FIPS 140-3 CipherSpecs.

FIPS 140-2 support is still available in IBM SDK, Java Technology Edition 8 to IBM Semeru Runtime 21.

Table 20 on page 364 shows which CipherSuites the AMQP service supports from IBM MQ 10.0.0.

Table 20. CipherSuites you can use with AMQP channels

CipherSuite	Protocol	FIPS 140-3 compatible
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	Yes
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	Yes
TLS_RSA_WITH_AES_128_CBC_SHA "1" on page 364	TLS 1.0	No
TLS_RSA_WITH_AES_128_CBC_SHA256 "1" on page 364	TLS 1.2	No
TLS_RSA_WITH_AES_128_GCM_SHA256 "1" on page 364	TLS 1.2	No
TLS_RSA_WITH_AES_256_CBC_SHA "1" on page 364	TLS 1.0	No
TLS_RSA_WITH_AES_256_CBC_SHA256 "1" on page 364	TLS 1.2	No
TLS_RSA_WITH_AES_256_GCM_SHA384 "1" on page 364	TLS 1.2	No
SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	No
SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	No
*TLS12	TLS 1.2	No
TLS_AES_128_GCM_SHA256	TLS 1.3	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	Yes
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	No
TLS_AES_128_CCM_SHA256	TLS 1.3	No
TLS_AES_128_CCM_8_SHA256	TLS 1.3	No
*ANY	Multiple	No
*TLS13	TLS 1.3	No
*TLS12ORHIGHER	TLS 1.2 and above	No
*TLS13ORHIGHER	TLS 1.3 and above	No

Notes:

1. The IBM Java 8 JRE removed support for RSA key exchange when operating in FIPS mode. TLS_RSA_* CipherSpecs are disabled by default in IBM Semeru Runtime 21 and IBM Semeru Runtime 25.

Disconnecting AMQP clients from the queue manager

If you want to disconnect AMQP clients from the queue manager, either run the PURGE CHANNEL command or stop the connection to the AMQP client.

- Run the **PURGE CHANNEL** command. For example:

```
PURGE CHANNEL(MYAMQP) CLIENTID('recv_28dbb7e')
```

- Alternatively, stop the connection that the AMQP client is using to disconnect the client by completing the following steps:

  1. Find the connection that the client is using by running the **DISPLAY CONN** command. For example:

     ```
     DISPLAY CONN(*) TYPE(CONN) WHERE (CLIENTID EQ 'recv_28dbb7e')
     ```

     The command output is as follows:

     ```
     DISPLAY CONN(*)  TYPE(CONN) WHERE(CLIENTID EQ 'recv_28dbb7e')
       40 : DISPLAY CONN(*) TYPE(CONN) WHERE(CLIENTID EQ 'recv_28dbb7e')
     AMQ8276: Display  Connection details.
       CONN(707E0A565F2D0020)
       EXTCONN(414D5143514D31202020202020202020)
       TYPE(CONN)
       CLIENTID(recv_28dbb7e)
     ```

  2. Stop the connection. For example:

     ```
     STOP CONN(707E0A565F2D0020)
     ```

Administering multicast

Use this information to learn about the IBM MQ Multicast administration tasks such as reducing the size of multicast messages and enabling data conversion.

Getting started with multicast

Use this information to get started with IBM MQ Multicast topics and communication information objects.

About this task

IBM MQ Multicast messaging uses the network to deliver messages by mapping topics to group addresses. The following tasks are a quick way to test if the required IP address and port are correctly configured for multicast messaging.

Creating a COMMINFO object for multicast

The communication information (COMMINFO) object contains the attributes associated with multicast transmission. For more information about the COMMINFO object parameters, see DEFINE COMMINFO.

Use the following command-line example to define a COMMINFO object for multicast:

```
DEFINE COMMINFO(MC1) GRPADDR(group address) PORT(port number)
```

where *MC1* is the name of your COMMINFO object, *group address* is your group multicast IP address or DNS name, and the *port number* is the port to transmit on (The default value is 1414).

A new COMMINFO object called *MC1* is created; This name is the name that you must specify when defining a TOPIC object in the next example.

Creating a TOPIC object for multicast

A topic is the subject of the information that is published in a publish/subscribe message, and a topic is defined by creating a TOPIC object. TOPIC objects have two parameters which define whether they can be used with multicast or not. These parameters are: **COMMINFO** and **MCAST**.

- **COMMINFO** This parameter specifies the name of the multicast communication information object. For more information about the COMMINFO object parameters, see DEFINE COMMINFO.
- **MCAST** This parameter specifies whether multicast is allowable at this position in the topic tree.

Use the following command-line example to define a TOPIC object for multicast:

```
DEFINE TOPIC(ALLSPORTS) TOPICSTR('Sports') COMMINFO(MC1) MCAST(ENABLED)
```

A new TOPIC object called *ALLSPORTS* is created. It has a topic string *Sports*, its related communication information object is called *MC1* (which is the name you specified when defining a COMMINFO object in the previous example), and multicast is enabled.

Testing the multicast publish/subscribe

After the TOPIC and COMMINFO objects have been created, they can be tested using the `amqspubc` sample and the `amqssubc` sample. For more information about these samples see The Publish/Subscribe sample programs.

1. Open two command-line windows; The first command line is for the `amqspubc` publish sample, and the second command line is for the `amqssubc` subscribe sample.
2. Enter the following command at command line 1:

   ```
   amqspubc Sports QM1
   ```

   where *Sports* is the topic string of the TOPIC object defined in an earlier example, and *QM1* is the name of the queue manager.
3. Enter the following command at command line 2:

   ```
   amqssubc Sports QM1
   ```

   where *Sports* and *QM1* are the same as used in step “2” on page 366.
4. Enter `Hello world` at command line 1. If the port and IP address that are specified in the COMMINFO object are configured correctly; the `amqssubc` sample, which is listening on the port for publications from the specified address, outputs `Hello world` at command line 2.

IBM MQ Multicast topic topology

Use this example to understand the IBM MQ Multicast topic topology.

IBM MQ Multicast support requires that each subtree has its own multicast group and data stream within the total hierarchy.

The *classful network* IP addressing scheme has designated address space for multicast address. The full multicast range of IP address is `224.0.0.0` to `239.255.255.255`, but some of these addresses are reserved. For a list of reserved address either contact your system administrator or see https://www.iana.org/assignments/multicast-addresses for more information. It is recommended that you use the locally scoped multicast address in the range of `239.0.0.0` to `239.255.255.255`.

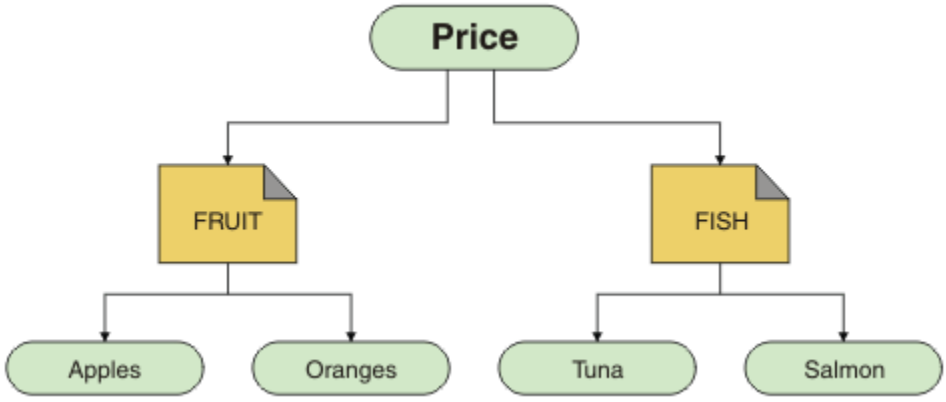
In the following diagram, there are two possible multicast data streams:

```
DEF COMMINFO(MC1) GRPADDR(239.XXX.XXX.XXX)
DEF COMMINFO(MC2) GRPADDR(239.YYY.YYY.YYY)
```

where *239.XXX.XXX.XXX* and *239.YYY.YYY.YYY* are valid multicast addresses.

These topic definitions are used to create a topic tree as shown in the following diagram:

```
DEFINE TOPIC(FRUIT) TOPICSTRING('Price/FRUIT') MCAST(ENABLED) COMMINFO(MC1)
DEFINE TOPIC(FISH) TOPICSTRING('Price/FISH') MCAST(ENABLED) COMMINFO(MC2)
```



Each multicast communication information (COMMINFO) object represents a different stream of data because their group addresses are different. In this example, the FRUIT topic is defined to use COMMINFO object MC1 , the FISH topic is defined to use COMMINFO object MC2 , and the Price node has no multicast definitions.

IBM MQ Multicast has a 255 character limit for topic strings. This limitation means that care must be taken with the names of nodes and leaf-nodes within the tree; if the names of nodes and leaf-nodes are too long, the topic string might exceed 255 characters and return the 2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR reason code. It is recommended to make topic strings as short as possible because longer topic strings might have a detrimental effect on performance.

Controlling the size of multicast messages

Use this information to learn about the IBM MQ message format, and reduce the size of IBM MQ messages.

IBM MQ messages have a number of attributes associated with them which are contained in the message descriptor. For small messages, these attributes might represent most of the data traffic and can have a significant detrimental effect on the transmission rate. IBM MQ Multicast enables the user to configure which, if any, of these attributes are transmitted along with the message.

The presence of message attributes, other than topic string, depends on whether the COMMINFO object states that they must be sent or not. If an attribute is not transmitted, the receiving application applies a default value. The default MQMD values are not necessarily the same as the MQMD_DEFAULT value, and are described in Table 21 on page 368.

The COMMINFO object contains the MCPROP attribute which controls how many of the MQMD fields and user properties flow with the message. By setting the value of this attribute to an appropriate level, you can control the size of the IBM MQ Multicast messages:

MCPROP
The multicast properties control how many of the MQMD properties and user properties flow with the message.

ALL
All user properties and all the fields of the MQMD are transmitted.

REPLY
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

USER
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM, which allows some inter-operation with the current XMS applications and IBM Integration Bus RMM applications.

Removed XMS .NET Multicast messaging (using RMM) was deprecated from IBM MQ 9.2 and removed at IBM MQ 9.3.

Multicast message attributes

Message attributes can come from various places, such as the MQMD, the fields in the MQRFH2, and message properties.

The following table shows what happens when messages are sent subject to the value of MCPROP (described previously in this section), and the default value used when an attribute is not sent.

Table 21. Messaging attributes and how they relate to multicast

Attribute	Action when using multicast	Default if not transmitted
TopicString	Always Included	Not applicable
MQMQ StrucId	Not transmitted	Not applicable
MQMD Version	Not transmitted	Not applicable
Report	Included if not default	0
MsgType	Included if not default	MQMT_DATAGRAM
Expiry	Included if not default	0
Feedback	Included if not default	0
Encoding	Included if not default	MQENC_NORMAL(equiv)
CodedCharSetId	Included if not default	1208
Format	Included if not default	MQRFH2
Priority	Included if not default	4
Persistence	Included if not default	MQPER_NOT_PERSISTENT
MsgId	Included if not default	Null
CorrelId	Included if not default	Null
BackoutCount	Included if not default	0
ReplyToQ	Included if not default	Blank
ReplyToQMgr	Included if not default	Blank
UserIdentifier	Included if not default	Blank
AccountingToken	Included if not default	Null
PutAppIType	Included if not default	MQAT_JAVA
PutAppIName	Included if not default	Blank
PutDate	Included if not default	Blank

Table 21. Messaging attributes and how they relate to multicast (continued)		
Attribute	**Action when using multicast**	**Default if not transmitted**
PutTime	Included if not default	Blank
ApplOriginData	Included if not default	Blank
GroupID	Excluded	Not applicable
MsgSeqNumber	Excluded	Not applicable
Offset	Excluded	Not applicable
MsgFlags	Excluded	Not applicable
OriginalLength	Excluded	Not applicable
UserProperties	Included	Not applicable

Related reference

Multi ALTER COMMINFO

DEFINE COMMINFO

Enabling data conversion for Multicast messaging

Use this information to understand how data conversion works for IBM MQ Multicast messaging.

IBM MQ Multicast is a shared, connectionless protocol, and so it is not possible for each client to make specific requests for data conversion. Every client subscribed to the same multicast stream receives the same binary data; therefore, if IBM MQ data conversion is required, the conversion is performed locally at each client.

In a mixed platform installation, it might be that most of the clients require the data in a format that is not the native format of the transmitting application. In this situation the **CCSID** and **ENCODING** values of the multicast `COMMINFO` object can be used to define the encoding of the message transmission for efficiency.

IBM MQ Multicast supports data conversion of the message payload for the following built in formats:

- MQADMIN
- MQEVENT
- MQPCF
- MQRFH
- MQRFH2
- MQSTR

In addition to these formats, you can also define your own formats and use an MQDXP - Data-conversion exit parameter data conversion exit.

For information about programming data conversions, see Data conversion in the MQI for multicast messaging.

For more information about data conversion, see Data conversion.

For more information about data conversion exits and `ClientExitPath`, see ClientExitPath stanza of the client configuration file.

Multicast application monitoring

Use this information to learn about administering and monitoring IBM MQ Multicast.

The status of the current publishers and subscribers for multicast traffic (for example, the number of messages sent and received, or the number of messages lost) is periodically transmitted to the server

from the client. When status is received, the COMMEV attribute of the COMMINFO object specifies whether or not the queue manager puts an event message on the SYSTEM.ADMIN.PUBSUB.EVENT. The event message contains the status information received. This information is an invaluable diagnostic aid in finding the source of a problem.

Use the MQSC command **DISPLAY CONN** to display connection information about the applications connected to the queue manager. For more information on the **DISPLAY CONN** command, see DISPLAY CONN.

Use the MQSC command **DISPLAY TPSTATUS** to display the status of your publishers and subscribers. For more information on the **DISPLAY TPSTATUS** command, see DISPLAY TPSTATUS.

COMMEV and the multicast message reliability indicator

The *reliability indicator*, used in conjunction with the **COMMEV** attribute of the COMMINFO object, is a key element in the monitoring of IBM MQ Multicast publishers and subscribers. The reliability indicator (the **MCASTREL** field that is returned on the Publish or Subscribe status commands) is an IBM MQ indicator that illustrates the percentage of transmissions that have no errors Sometimes messages have to be retransmitted due to a transmission error, which is reflected in the value of **MCASTREL**. Potential causes of transmission errors include slow subscribers, busy networks, and network outages. **COMMEV** controls whether event messages are generated for multicast handles that are created using the COMMINFO object and is set to one of three possible values:

DISABLED
Event messages are not written.

ENABLED
Event messages are always written, with a frequency defined in the COMMINFO **MONINT** parameter.

EXCEPTION
Event messages are written if the message reliability is under the reliability threshold. A message reliability level of 90% or less indicates that there might be a problem with the network configuration, or that one or more of the Publish/Subscribe applications is running too slowly:

- A value of **MCASTREL(100,100)** indicates that there have been no issues in either the short term, or the long-term time frame.
- A value of **MCASTREL(80,60)** indicates that 20% of the messages are currently having issues, but that it is also an improvement on the long-term value of 60.

Clients might continue transmitting and receiving multicast traffic even when the unicast connection to the queue manager is broken, therefore the data might be out of date.

Multicast message reliability

Use this information to learn how to set the IBM MQ Multicast subscription and message history.

A key element of overcoming transmission failure with multicast is the buffering of transmitted data (a history of messages to be kept at the transmitting end of the link) by IBM MQ. This process means that no buffering of messages is required in the putting application process because IBM MQ provides the reliability. The size of this history is configured via the communication information (COMMINFO) object, as described in the following information. A bigger transmission buffer means that there is more transmission history to be retransmitted if needed, but due to the nature of multicast, 100% assured delivery cannot be supported.

The IBM MQ Multicast message history is controlled in the communication information (COMMINFO) object by the **MSGHIST** attribute:

MSGHIST
This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

A value of 0 gives the least level of reliability. The default value is 100 KB.

The IBM MQ Multicast new subscription history is controlled in the communication information (COMMINFO) object by the **NSUBHIST** attribute:

NSUBHIST
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

NONE
A value of NONE causes the transmitter to transmit only publication made from the time of the subscription. NONE is the default value.

ALL
A value of ALL causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this situation can give a similar behavior to retained publications.

Note: Using the value of ALL might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

Related reference
DEFINE COMMINFO
Multi ALTER COMMINFO

Advanced multicast tasks

Use this information to learn about advanced IBM MQ Multicast administration tasks such as configuring `.ini` files and interoperability with IBM MQ LLM.

For considerations for security in a Multicast installation, see Multicast security.

Bridging between multicast and non-multicast publish/subscribe domains

Use this information to understand what happens when a non-multicast publisher publishes to an IBM MQ Multicast enabled topic.

If a non-multicast publisher publishes to a topic that is defined as **MCAST** enabled and **BRIDGE** enabled, the queue manager transmits the message out over multicast directly to any subscribers that might be listening. A multicast publisher cannot publish to topics that are not multicast enabled.

Existing topics can be multicast enabled by setting the **MCAST** and **COMMINFO** parameters of a topic object. See Initial multicast concepts for more information about these parameters.

The COMMINFO object **BRIDGE** attribute controls publications from applications that are not using multicast. If **BRIDGE** is set to ENABLED and the **MCAST** parameter of the topic is also set to ENABLED, publications from applications that are not using multicast are bridged to applications that do. For more information on the **BRIDGE** parameter, see DEFINE COMMINFO.

Configuring the .ini files for Multicast

Use this information to understand the IBM MQ Multicast fields in the `.ini` files.

Additional IBM MQ Multicast configuration can be made in an `ini` file. The specific `ini` file that you must use is dependent on the type of applications:

- Client: Configure the *MQ_DATA_PATH* `/mqclient.ini` file.
- Queue manager: Configure the *MQ_DATA_PATH* `/qmgrs/`*QMNAME*`/qm.ini` file.

where *MQ_DATA_PATH* is the location of the IBM MQ data directory (`/var/mqm/mqclient.ini`), and *QMNAME* is the name of the queue manager to which the `.ini` file applies.

The `.ini` file contains fields used to fine-tune the behavior of IBM MQ Multicast:

```
Multicast:
Protocol       = IP | UDP
```

```
IPVersion       = IPv4 | IPv6 | ANY | BOTH
LimitTransRate    = DISABLED | STATIC | DYNAMIC
TransRateLimit    = 100000
SocketTTL       = 1
Batch         = NO
Loop         = 1
Interface       = <IPaddress>
FeedbackMode     = ACK | NACK | WAIT1
HeartbeatTimeout   = 20000
HeartbeatInterval   = 2000
```

Protocol

UDP
In this mode, packets are sent using the UDP protocol. Network elements cannot provide assistance in the multicast distribution as they do in IP mode however. The packet format remains compatible with PGM. This is the default value.

IP
In this mode, the transmitter sends raw IP packets. Network elements with PGM support assist in the reliable multicast packet distribution. This mode is fully compatible with the PGM standard.

IPVersion

IPv4
Communicate using the IPv4 protocol only. This is the default value.

IPv6
Communicate using the IPv6 protocol only.

ANY
Communicate using IPv4, IPv6, or both, depending on which protocol is available.

BOTH
Supports communication using both IPv4 and IPv6.

LimitTransRate

DISABLED
There is no transmission rate control. This is the default value.

STATIC
Implements static transmission rate control. The transmitter would not transmit at a rate exceeding the rate specified by the TransRateLimit parameter.

DYNAMIC
The transmitter adapts its transmission rate according to the feedback it gets from the receivers. In this case the transmission rate limit cannot be more than the value specified by the TransRateLimit parameter. The transmitter tries to reach an optimal transmission rate.

TransRateLimit
The transmission rate limit in Kbps.

SocketTTL
The value of SocketTTL determines if the multicast traffic can pass through a router, or the number of routers it can pass through.

Batch
Controls whether messages are batched or sent immediately There are 2 possible values:

- *NO* The messages are not batched, they are sent immediately.
- *YES* The messages are batched.

Loop
Set the value to 1 to enable multicast loop. Multicast loop defines whether the data sent is looped back to the host or not.

Interface
The IP address of the interface on which multicast traffic flows. For more information and troubleshooting, see: Testing multicast applications on a non-multicast network and Setting the appropriate network for multicast traffic

FeedbackMode

NACK
Feedback by negative acknowledgments. This is the default value.

ACK
Feedback by positive acknowledgments.

WAIT1
Feedback by positive acknowledgments where the transmitter waits for only 1 ACK from any of the receivers.

HeartbeatTimeout
The heartbeat timeout in milliseconds. A value of 0 indicates that the heartbeat timeout events are not raised by the receiver or receivers of the topic. The default value is 20000.

HeartbeatInterval
The heartbeat interval in milliseconds. A value of 0 indicates that no heartbeats are sent. The heartbeat interval must be considerably smaller than the **HeartbeatTimeout** value to avoid false heartbeat timeout events. The default value is 2000.

Multicast interoperability with IBM MQ Low Latency Messaging

Use this information to understand the interoperability between IBM MQ Multicast and IBM MQ Low Latency Messaging (LLM).

Basic payload transfer is possible for an application using LLM, with another application using multicast to exchange messages in both directions. Although multicast uses LLM technology, the LLM product itself is not embedded. Therefore it is possible to install both LLM and IBM MQ Multicast, and operate and service the two products separately.

LLM applications that communicate with multicast might need to send and receive message properties. The IBM MQ message properties and MQMD fields are transmitted as LLM message properties with specific LLM message property codes as shown in the following table:

Table 22. IBM MQ message properties to IBM MQ LLM property mappings

IBM MQ property	IBM MQ LLM property type	LLM property kind	LLM property code
MQMD.Report	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1001
MQMD.MsgType	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1002
MQMD.Expiry	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1003
MQMD.Feedback	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1004
MQMD.Encoding	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1005
MQMD.CodedCharSetId	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1006
MQMD.Format	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1007
MQMD.Priority	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1008
MQMD.Persistence	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1009
MQMD.MsgId	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_ByteArray	-1010
MQMD.BackoutCount	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1012
MQMD.ReplyToQ	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1013
MQMD.ReplyToQMger	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1014
MQMD.PutDate	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1020
MQMD.PutTime	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1021

Table 22. IBM MQ message properties to IBM MQ LLM property mappings (continued)			
IBM MQ property	**IBM MQ LLM property type**	**LLM property kind**	**LLM property code**
MQMD.ApplOriginData	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1022
MQPubOptions	RMM_MSG_PROP_INT32	LLM_PROP_KIND_int32	-1053

For more information about LLM, see the LLM product documentation: IBM MQ Low Latency Messaging.

IBM i Administering IBM MQ for IBM i

CL commands are the preferred method to administer IBM MQ on IBM i. You can also use MQSC commands, PCF commands, control commands, and remote administration.

About this task

Administration tasks include creating, starting, altering, viewing, stopping and deleting clusters, processes and IBM MQ objects (queue managers, queues, namelists, process definitions, channels, client connection channels, listeners, services and authentication information objects).

Related concepts

Understanding IBM MQ for IBM i queue manager library names

Installable services and components on IBM i

Related tasks

Changing IBM MQ configuration information on Multiplatforms

Setting up security on IBM i

“Invoking the dead-letter queue handler on IBM i” on page 170

On IBM MQ for IBM i, you invoke the DLQ handler by setting the **STRMQMDLQ** command.

Determining problems with IBM MQ for IBM i applications

Related reference

System and default objects

IBM i Managing IBM MQ for IBM i using CL commands

Use this information to understand the IBM MQ IBM i commands.

Most groups of IBM MQ commands, including those associated with queue managers, queues, topics, channels, namelists, process definitions, and authentication information objects can be accessed using the relevant **WRK*** command.

The principal command in the set is **WRKMQM**. This command allows you, for example, to display a list of all the queue managers on the system, together with status information. Alternatively, you can process all queue manager specific commands using various options against each entry.

From the **WRKMQM** command you can select specific areas of each queue manager, for example, working with channels, topics or queues, and from there select individual objects.

Recording IBM MQ application definitions

When you create or customize IBM MQ applications, it is useful to keep a record of all IBM MQ definitions created. This record can be used for:

- Recovery purposes
- Maintenance
- Rolling out IBM MQ applications

You can record IBM MQ application definitions in 1 of 2 ways:

1. Creating CL programs to generate your IBM MQ definitions for the server.
2. Creating MQSC text files as SRC members to generate your IBM MQ definitions using the cross-platform IBM MQ command language.

For further details about defining queue objects, see “Administering IBM MQ using MQSC commands” on page 12 and “Using IBM MQ Programmable Command Formats” on page 26.

Related reference
IBM MQ for IBM i CL commands reference

IBM i Before you start using the IBM MQ for IBM i using CL commands

Use this information to start the IBM MQ subsystem and create a local queue manager.

Before you begin
Ensure that the IBM MQ subsystem is running (using the command STRSBS QMQM/QMQM), and that the job queue associated with that subsystem is not held. By default, the IBM MQ subsystem and job queue are both named QMQM in library QMQM.

About this task
Using the IBM i command line to start a queue manager

Procedure
1. Create a local queue manager by issuing the CRTMQM command from an IBM i command line.

   When you create a queue manager, you have the option of making that queue manager the default queue manager. The default queue manager (of which there can only be one) is the queue manager to which a CL command applies, if the queue manager name parameter (MQMNAME) is omitted.
2. Start a local queue manager by issuing the STRMQM command from an IBM i command line.

   If the queue manager startup takes more than a few seconds IBM MQ will show status messages intermittently detailing the startup progress. For more information on these messages see Messages and reason codes.

What to do next

You can stop a queue manager by issuing the ENDMQM command from the IBM i command line, and control a queue manager by issuing other IBM MQ commands from an IBM i command line.

Remote queue managers cannot be started remotely but must be created and started in their systems by local operators. An exception to this is where remote operating facilities (outside IBM MQ for IBM i) exist to enable such operations.

The local queue administrator cannot stop a remote queue manager.

Note: As part of quiescing an IBM MQ system, you have to quiesce the active queue managers. This is described in “Quiescing IBM MQ for IBM i” on page 440.

IBM i Creating IBM MQ for IBM i objects

Use this information to understand the methods for creating IBM MQ objects for IBM i.

Before you begin
The following tasks suggest various ways in which you can use IBM MQ for IBM i from the command line.

About this task

There are two online methods to create IBM MQ objects, which are:

Procedure

1. Using a Create command, for example: The **Create MQM Queue** command: **CRTMQMQ**
2. Using a Work with MQM object command, followed by F6, for example: The **Work with MQM Queues** command: **WRKMQMQ**

What to do next

For a list of all commands see IBM MQ for IBM i CL commands.

Note: All MQM commands can be submitted from the Message Queue Manager Commands menu. To display this menu, type `GO CMDMQM` on the command line and press the `Enter` key.

The system displays the prompt panel automatically when you select a command from this menu. To display the prompt panel for a command that you have typed directly on the command line, press F4 before pressing the `Enter` key.

Creating a local queue using the CRTMQMQ command

Procedure

1. Type `CHGMQM` on the command line and press the F4 key.
2. On the **Create MQM Queue panel**, type the name of the queue that you want to create in the `Queue name` field. To specify a mixed case name, you enclose the name in apostrophes.
3. Type `*LCL` in the `Queue type` field.
4. Specify a queue manager name, unless you are using the default queue manager, and press the `Enter` key. You can overtype any of the values with a new value. Scroll forward to see further fields. The options used for clusters are at the end of the list of options.
5. When you have changed any values, press the `Enter` key to create the queue.

Creating a local queue using the WRKMQMQ command

Procedure

1. Type `WRKMQMQ` on the command line.
2. Enter the name of a queue manager.
3. If you want to display the prompt panel, press F4. The prompt panel is useful to reduce the number of queues displayed, by specifying a generic queue name or queue type.
4. Press `Enter` and the **Work with MQM Queues panel** is displayed. You can overtype any of the values with a new value. Scroll forward to see further fields. The options used for clusters are at the end of the list of options.
5. Press F6 to create a new queue; this takes you to the **CRTMQMQ** panel. See "Creating a local queue using the CRTMQMQ command" on page 376 for instructions on how to create the queue. When you have created the queue, the **Work with MQM Queues panel** is displayed again. The new queue is added to the list when you press `F5=Refresh`.

Altering queue manager attributes

About this task

To alter the attributes of the queue manager specified on the **CHGMQM** command, specifying the attributes and values that you want to change. For example, use the following options to alter the attributes of `jupiter.queue.manager`:

Procedure

Type **CHGMQM** on the command line and press the F4 key.

Results

The command changes the dead-letter queue used, and enables inhibit events.

IBM i Working with local queues on IBM i

This section contains examples of some of the commands that you can use to manage local queues. All the commands shown are also available using options from the **WRKMQMQ command panel**.

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues that are managed by the local queue manager are said to be local to that queue manager.

Use the command **CRTMQMQ QTYPE *LCL** to create a definition of a local queue and also to create the data structure that is called a queue. You can also modify the queue characteristics from those of the default local queue.

In this example, the queue we define, `orange.local.queue`, is specified to have these characteristics:

- It is enabled for gets, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an *ordinary* queue, that is, it is not an initiation queue or a transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

The following command does this on the default queue manager:

```
CRTMQMQ QNAME('orange.local.queue') QTYPE(*LCL)
TEXT('Queue for messages from other systems')
PUTENBL(*NO)
GETENBL(*YES)
TRGENBL(*NO)
MSGDLYSEQ(*FIFO)
MAXDEPTH(1000)
MAXMSGLEN(2000)
USAGE(*NORMAL)
```

Note:

1. USAGE *NORMAL indicates that this queue is not a transmission queue.
2. If you already have a local queue with the name `orange.local.queue` on the same queue manager, then this command fails. Use the REPLACE *YES attribute if you want to overwrite the existing definition of a queue, but see also “Changing local queue attributes” on page 378.

Defining a dead-letter queue

Each queue manager must have a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You must explicitly tell the queue manager about the dead-letter queue. You can do this by specifying a dead-letter queue on the **CRTMQM** command, or you can use the **CHGMQM** command to specify one later. You must also define the dead-letter queue before it can be used.

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is supplied with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required. There is no need to rename it, although you can if you like.

A dead-letter queue has no special requirements except that:

- It must be a local queue.
- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH).

IBM MQ provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see "Invoking the dead-letter queue handler on IBM i" on page 170.

Displaying default object attributes

When you define an IBM MQ object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called SYSTEM.DEFAULT.LOCAL.QUEUE. To see exactly what these attributes are, use the following command:

```
DSPMQMQ QNAME(SYSTEM.DEFAULT.LOCAL.QUEUE) MQMNAME(MYQUEUEMANAGER)
```

Copying a local queue definition

You can copy a queue definition using the CPYMQMQ command. For example:

```
CPYMQMQ FROMQ('orange.local.queue') TOQ('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

This command creates a queue with the same attributes as our original queue `orange.local.queue`, rather than those of the system default local queue.

You can also use the **CPYMQMQ** command to copy a queue definition, but substituting one or more changes to the attributes of the original. For example:

```
CPYMQMQ FROMQ('orange.local.queue') TOQ('third.queue') MQMNAME(MYQUEUEMANAGER)
MAXMSGLEN(1024)
```

This command copies the attributes of the queue `orange.local.queue` to the queue `third.queue`, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Note: When you use the **CPYMQMQ** command, you copy the queue attributes only, not the messages on the queue.

Changing local queue attributes

You can change queue attributes in two ways, using either the **CHGMQMQ** command or the **CPYMQMQ** command with the REPLACE *YES attribute. In "Defining a local queue" on page 377, you defined the queue `orange.local.queue`. If, for example, you need to increase the maximum message length on this queue to 10,000 bytes.

- Using the **CHGMQMQ** command:

  ```
  CHGMQMQ QNAME('orange.local.queue') MQMNAME(MYQUEUEMANAGER) MAXMSGLEN(10000)
  ```

  This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.
- Using the **CRTMQMQ** command with the REPLACE *YES option, for example:

  ```
  CRTMQMQ QNAME('orange.local.queue') QTYPE(*LCL) MQMNAME(MYQUEUEMANAGER)
  MAXMSGLEN(10000) REPLACE(*YES)
  ```

  This command changes not only the maximum message length, but all the other attributes, which are given their default values. The queue is now put enabled whereas previously it was put inhibited. Put enabled is the default, as specified by the queue SYSTEM.DEFAULT.LOCAL.QUEUE, unless you have changed it.

  If you ***decrease*** the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Clearing a local queue

To delete all the messages from a local queue called `magenta.queue`, use the following command:

```
CLRMQMQ QNAME('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under syncpoint.
- An application currently has the queue open.

Deleting a local queue

Use the command **DLTMQMQ** to delete a local queue.

A queue cannot be deleted if it has uncommitted messages on it, or if it is in use.

Enabling large queues

IBM MQ supports queues larger than 2 GB. See your operating system documentation for information on how to enable IBM i to support large files.

The IBM i product information can be found in IBM Documentation.

Some utilities might not be able to cope with files greater than 2 GB. Before enabling large file support, check your operating system documentation for information on restrictions on such support.

IBM i Working with alias queues on IBM i

This section contains examples of some of the commands that you can use to manage alias queues. All the commands shown are also available using options from the **WRKMQMQ command panel**.

An alias queue (sometimes known as a queue alias) provides a method of redirecting MQI calls. An alias queue is not a real queue but a definition that resolves to a real queue. The alias queue definition contains a target queue name, which is specified by the TGTQNAME attribute.

When an application specifies an alias queue in an MQI call, the queue manager resolves the real queue name at run time.

For example, an application has been developed to put messages on a queue called `my.alias.queue`. It specifies the name of this queue when it makes an **MQOPEN** request and, indirectly, if it puts a message on this queue. The application is not aware that the queue is an alias queue. For each MQI call using this alias, the queue manager resolves the real queue name, which could be either a local queue or a remote queue defined at this queue manager.

By changing the value of the TGTQNAME attribute, you can redirect MQI calls to another queue, possibly on another queue manager. This is useful for maintenance, migration, and load-balancing.

Defining an alias queue

The following command creates an alias queue:

```
CRTMQMQ QNAME('my.alias.queue') QTYPE(*ALS) TGTQNAME('yellow.queue')
MQMNAME(MYQUEUEMANAGER)
```

This command redirects MQI calls that specify `my.alias.queue` to the queue `yellow.queue`. The command does not create the target queue; the MQI calls fail if the queue `yellow.queue` does not exist at run time.

If you change the alias definition, you can redirect the MQI calls to another queue. For example:

```
CHGMQMQ QNAME('my.alias.queue') TGTQNAME('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

This command redirects MQI calls to another queue, `magenta.queue`.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

- Application ALPHA can put messages on `yellow.queue`, but is not allowed to get messages from it.
- Application BETA can get messages from `yellow.queue`, but is not allowed to put messages on it.

You can do this using the following commands:

```
/* This alias is put enabled and get disabled for application ALPHA */

CRTMQMQ QNAME('alphas.alias.queue') QTYPE(*ALS)     TGTQNAME('yellow.queue')
PUTENBL(*YES) GETENBL(*NO) MQMNAME(MYQUEUEMANAGER)

/* This alias is put disabled and get enabled for application BETA */

CRTMQMQ QNAME('betas.alias.queue') QTYPE(*ALS) TGTQNAME('yellow.queue')
PUTENBL(*NO) GETENBL(*YES) MQMNAME(MYQUEUEMANAGER)
```

ALPHA uses the queue name `alphas.alias.queue` in its MQI calls; BETA uses the queue name `betas.alias.queue`. They both access the same queue, but in different ways.

You can use the REPLACE *YES attribute when you define alias queues, in the same way that you use these attributes with local queues.

Using other commands with alias queues

You can use the appropriate commands to display or change alias queue attributes. For example:

```
* Display the alias queue's attributes */

DSPMQMQ QNAME('alphas.alias.queue') MQMNAME(MYQUEUEMANAGER)

/* ALTER the base queue name, to which the alias resolves. */
/* FORCE = Force the change even if the queue is open. */

CHQMQMQ QNAME('alphas.alias.queue') TGTQNAME('orange.local.queue') FORCE(*YES)
MQMNAME(MYQUEUEMANAGER)
```

IBM i Working with model queues on IBM i

This section contains examples of some of the commands that you can use to manage model queues. All the commands shown are also available using options from the **WRKMQMQ command panel**.

A queue manager creates a dynamic queue if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A model queue is a template that specifies the attributes of any dynamic queues created from it.

Model queues provide a convenient method for applications to create queues as they are required.

Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes, except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not). For example:

```
CRTMQMQ QNAME('green.model.queue') QTYPE(*MDL) DFNTYPE(*PERMDYN)
```

This command creates a model queue definition. From the DFNTYPE attribute, the actual queues created from this template are permanent dynamic queues. The attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the REPLACE *YES attribute when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

You can use the appropriate commands to display or alter a model queue's attributes. For example:

```
/* Display the model queue's attributes */

DSPMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('green.model.queue')

/* ALTER the model queue to enable puts on any */
/* dynamic queue created from this model. */

CHGMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('blue.model.queue') PUTENBL(*YES)
```

IBM i Working with triggering on IBM i

Use this information to learn about triggering and process definitions.

IBM MQ provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in Triggering channels.

What is triggering?

The queue manager defines certain conditions as constituting trigger events. If triggering is enabled for a queue and a trigger event occurs, the queue manager sends a trigger message to a queue called an initiation queue. The presence of the trigger message on the initiation queue indicates that a trigger event has occurred.

Trigger messages generated by the queue manager are not persistent. This has the effect of reducing logging (thereby improving performance), and minimizing duplicates during restart, so improving restart time.

What is the trigger monitor?

The program which processes the initiation queue is called a trigger-monitor application, and its function is to read the trigger message and take appropriate action, based on the information contained in the trigger message. Normally this action would be to start some other application to process the queue which caused the trigger message to be generated. From the point of view of the queue manager, there is nothing special about the trigger-monitor application - it is another application that reads messages from a queue (the initiation queue).

Altering the job submission attributes of the trigger monitor

The trigger monitor supplied as command **STRMQMTRM** submits a job for each trigger message using the system default job description, QDFTJOBD. This has limitations in that the submitted jobs are always called QDFTJOBD and have the attributes of the default job description including the library list, *SYSVAL. IBM MQ provides a method for overriding these attributes. For example, it is possible to customize the submitted jobs to have more meaningful job names as follows:

1. In the job description specify the description you want, for example logging values.
2. Specify the Environment Data of the process definition used in the triggering process:

   ```
   CHGMQMPRC PRCNAME(MY_PROCESS) MQMNAME(MHA3) ENVDATA ('JOBD(MYLIB/TRIGJOBD)')
   ```

   The Trigger Monitor performs a SBMJOB using the specified description.

It is possible to override other attributes of the SBMJOB by specifying the appropriate keyword and value in the Environment Data of the process definition. The only exception to this is the CMD keyword because

this attribute is filled by the trigger monitor. An example of the command to specify the Environment Data of the process definition where both the job name and description are to be altered follows:

```
CHGMQMPRC PRCNAME(MY_PROCESS) MQMNAME(MHA3) ENVDATA ('JOBD(MYLIB/TRIGJOB)
JOB(TRIGGER)')
```

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the TRGENBL attribute.

In this example, a trigger event is to be generated when there are 100 messages of priority 5 or higher on the local queue `motor.insurance.queue`, as follows:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.insurance.queue') QTYPE(*LCL)
PRCNAME('motor.insurance.quote.process') MAXMSGLEN(2000)
DFTMSGPST(*YES) INITQNAME('motor.ins.init.queue')
TRGENBL(*YES) TRGTYPE(*DEPTH) TRGDEPTH(100) TRGMSGPTY(5)
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
The name of the queue manager.

QNAME('motor.insurance.queue')
The name of the application queue being defined.

PRCNAME('motor.insurance.quote.process')
The name of the application to be started by a trigger monitor program.

MAXMSGLEN(2000)
The maximum length of messages on the queue.

DFTMSGPST(*YES)
Messages on this queue are persistent by default.

INITQNAME('motor.ins.init.queue')
The name of the initiation queue on which the queue manager is to put the trigger message.

TRGENBL(*YES)
The trigger attribute value.

TRGTYPE(*DEPTH)
A trigger event is generated when the number of messages of the required priority (**TRGMSGPTY**) reaches the number specified in **TRGDEPTH**.

TRGDEPTH(100)
The number of messages required to generate a trigger event.

TRGMSGPTY(5)
The priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue `motor.ins.init.queue` for guidance:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.ins.init.queue') QTYPE(*LCL)
GETENBL(*YES) SHARE(*NO) TRGTYPE(*NONE)
MAXMSGL(2000)
MAXDEPTH(1000)
```

Creating a process definition

Use the **CRTMQMPRC** command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PRCNAME attribute on the application queue `motor.insurance.queue`. The following command creates the required process, `motor.insurance.quote.process`, identified in this example:

```
CRTMQMPRC MQMNAME(MYQUEUEMANAGER) PRCNAME('motor.insurance.quote.process')
TEXT('Insurance request message processing')
APPTYPE(*OS400) APPID(MQTEST/TESTPROG)
USRDATA('open, close, 235')
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
The name of the queue manager.

PRCNAME('motor.insurance.quote.process')
The name of the process definition.

TEXT('Insurance request message processing')
A description of the application program to which this definition relates. This text is displayed when you use the **DSPMQMPRC** command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPTYPE(*OS400)
The type of application to be started.

APPID(MQTEST/TESTPROG)
The name of the application executable file, specified as a fully qualified file name.

USRDATA('open, close, 235')
User-defined data, which can be used by the application.

Displaying your process definition

Use the **DSPMQMPRC** command to examine the results of your definition. For example:

```
MQMNAME(MYQUEUEMANAGER) DSPMQMPRC('motor.insurance.quote.process')
```

You can also use the **CHGMQMPRC** command to alter an existing process definition, and the **DLTMQMPRC** command to delete a process definition.

IBM i Communicating between two IBM MQ systems on IBM i

This coding example illustrates how to set up two IBM MQ for IBM i systems, using CL commands, so that they can communicate with each other.

The systems are called SYSTEMA and SYSTEMB, and the communications protocol used is TCP/IP.

Carry out the following procedure:

1. Create a queue manager on SYSTEMA, calling it QMGRA1.

   ```
   CRTMQM   MQMNAME(QMGRA1) TEXT('System A - Queue +
   Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   ```

2. Start this queue manager.

   ```
   STRMQM   MQMNAME(QMGRA1)
   ```

3. Define the IBM MQ objects on SYSTEMA that you need to send messages to a queue manager on SYSTEMB.

   ```
   /* Transmission queue */
   CRTMQMQ  QNAME(XMITQ.TO.QMGRB1) QTYPE(*LCL) +
   MQMNAME(QMGRA1) TEXT('Transmission Queue +
   to QMGRB1') MAXDEPTH(5000) USAGE(*TMQ)
   ```

```
/* Remote queue that points to a queue called TARGETB        */
/* TARGETB belongs to queue manager QMGRB1 on SYSTEMB        */
CRTMQMQ  QNAME(TARGETB.ON.QMGRB1) QTYPE(*RMT) +
MQMNAME(QMGRA1) TEXT('Remote Q pointing +
at Q TARGETB on QMGRB1 on Remote System +
SYSTEMB') RMTQNAME(TARGETB) +
RMTMQMNAME(QMGRB1) TMQNAME(XMITQ.TO.QMGRB1)

/* TCP/IP sender channel to send messages to the queue manager on SYSTEMB*/
CRTMQMCHL CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*SDR) +
MQMNAME(QMGRA1) TRPTYPE(*TCP) +
TEXT('Sender Channel From QMGRA1 on +
SYSTEMA to QMGRB1 on SYSTEMB') +
CONNAME(SYSTEMB) TMQNAME(XMITQ.TO.QMGRB1)
```

4. Create a queue manager on SYSTEMB, calling it QMGRB1.

```
CRTMQM   MQMNAME(QMGRB1) TEXT('System B - Queue +
Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
```

5. Start the queue manager on SYSTEMB.

```
STRMQM   MQMNAME(QMGRB1)
```

6. Define the IBM MQ objects that you need to receive messages from the queue manager on SYSTEMA.

```
/* Local queue to receive messages on */
CRTMQMQ  QNAME(TARGETB) QTYPE(*LCL) MQMNAME(QMGRB1) +
TEXT('Sample Local Queue for QMGRB1')

/* Receiver channel of the same name as the sender channel on SYSTEMA */
CRTMQMCHL CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*RCVR) +
MQMNAME(QMGRB1) TRPTYPE(*TCP) +
TEXT('Receiver Channel from QMGRA1 to +
QMGRB1')
```

7. Finally, start a TCP/IP listener on SYSTEMB so that the channel can be started. This example uses the default port of 1414.

```
STRMQMLSR MQMNAME(QMGRB1)
```

You are now ready to send test messages between SYSTEMA and SYSTEMB. Using one of the supplied samples, put a series of messages to your remote queue on SYSTEMA.

Start the channel on SYSTEMA, either by using the command **STRMQMCHL**, or by using the command **WRKMQMCHL** and entering a start request (Option 14) against the sender channel.

The channel should go to RUNNING status and the messages are sent to queue TARGETB on SYSTEMB.

Check your messages by issuing the command:

```
WRKMQMMSG QNAME(TARGETB) MQMNAME(QMGRB1).
```

IBM i Creating sample queues using AMQSAMP4

You can run the AMQSAMP4 sample IBM i CL program to create some sample queues. The source for this program is included in file QCLSRC in library QMQMSAMP. You can compile it by using the CRTCLPGM command.

```
/*******************************************************************/
/*                                                                 */
/* Program name: AMQSAMP4                                          */
/*                                                                 */
/* Description: Sample CL program defining MQM queues              */
/*       to use with the sample programs                           */
/*       Can be run, with changes as needed, after                 */
/*       starting the MQM                                          */
/*                                                                 */
/* <N_OCO_COPYRIGHT>                                              */
/* Licensed Materials - Property of IBM                            */
/*                                                                 */
```

```
/* 63H9336                                                          */
/* (c) Copyright IBM Corp. 1993, 2026. All Rights Reserved.         */
/*                                                                  */
/* US Government Users Restricted Rights - Use, duplication or      */
/* disclosure restricted by GSA ADP Schedule Contract with          */
/* IBM Corp.                                                        */
/* <NOC_COPYRIGHT>                                                 */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/* Function:                                                        */
/*                                                                  */
/*                                                                  */
/*  AMQSAMP4 is a sample CL program to create or reset the          */
/*  MQI resources to use with the sample programs.                  */
/*                                                                  */
/*  This program, or a similar one, can be run when the MQM         */
/*  is started - it creates the objects if missing, or resets       */
/*  their attributes to the prescribed values.                      */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*  Exceptions signaled: none                                       */
/*  Exceptions monitored: none                                      */
/*                                                                  */
/*  AMQSAMP4 takes a single parameter, the Queue Manager name       */
/*                                                                  */
/********************************************************************/
QSYS/PGM PARM(&QMGRNAME)

/********************************************************************/
/*  Queue Manager Name Parameter                                    */
/********************************************************************/
QSYS/DCL VAR(&QMGRNAME) TYPE(*CHAR)

/********************************************************************/
/*    EXAMPLES OF DIFFERENT QUEUE TYPES                             */
/*                                                                  */
/*  Create local, alias and remote queues                           */
/*                                                                  */
/*  Uses system defaults for most attributes                        */
/*                                                                  */
/********************************************************************/
/*  Create a local queue   */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.LOCAL')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('Sample local queue') /* description */+
SHARE(*YES)        /* Shareable  */+
DFTMSGPST(*YES) /* Persistent messages OK */

/*  Create an alias queue  */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.ALIAS')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*ALS) REPLACE(*YES)         +
+
TEXT('Sample alias queue')         +
DFTMSGPST(*YES) /* Persistent messages OK */+
TGTQNAME('SYSTEM.SAMPLE.LOCAL')

/*  Create a remote queue - in this case, an indirect reference  */
/*   is made to the sample local queue on OTHER queue manager  */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.REMOTE')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*RMT) REPLACE(*YES)         +
+
TEXT('Sample remote queue')/* description */+
DFTMSGPST(*YES) /* Persistent messages OK */+
RMTQNAME('SYSTEM.SAMPLE.LOCAL')       +
RMTMQMNAME(OTHER)  /* Queue is on OTHER  */

/*  Create a transmission queue for messages to queues at OTHER  */
/*  By default, use remote node name                */
CRTMQMQ   QNAME('OTHER') /* transmission queue name */+
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES) +
TEXT('Transmision queue to OTHER') +
USAGE(*TMQ)  /* transmission queue   */

/********************************************************************/
```

```
/*    SPECIFIC QUEUES AND PROCESS USED BY SAMPLE PROGRAMS          */
/*                                                                */
/*  Create local queues used by sample programs                   */
/*  Create MQI process associated with sample initiation queue    */
/*                                                                */
/******************************************************************/
/*  General reply queue  */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.REPLY')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('General reply queue')         +
DFTMSGPST(*NO) /* Not Persistent     */

/*  Queue used by AMQSINQ4                     */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.INQ')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('Queue for AMQSINQ4')          +
SHARE(*YES)        /* Shareable  */+
DFTMSGPST(*NO) /* Not Persistent     */+
+
TRGENBL(*YES) /* Trigger control on   */+
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.INQPROCESS')     +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Queue used by AMQSSET4                     */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.SET')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('Queue for AMQSSET4')          +
SHARE(*YES)        /* Shareable */ +
DFTMSGPST(*NO)/* Not Persistent     */ +
+
TRGENBL(*YES) /* Trigger control on   */ +
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.SETPROCESS')     +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Queue used by AMQSECH4                      */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.ECHO')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('Queue for AMQSECH4')          +
SHARE(*YES)        /* Shareable */ +
DFTMSGPST(*NO)/* Not Persistent     */ +
+
TRGENBL(*YES) /* Trigger control on   */ +
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')    +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Initiation Queue used by AMQSTRG4, sample trigger process   */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.TRIGGER') +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)  +
TEXT('Trigger queue for sample programs')

/*  MQI Processes associated with triggered sample programs    */
/*                                 */
/*****  Note - there are versions of the triggered samples ******/
/*****   in different languages - set APPID for these   ******/
/*****   process to the variation you want to trigger   ******/
/*                                 */
CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.INQPROCESS')     +
MQMNAME(&QMGRNAME)             +
REPLACE(*YES)               +
+
TEXT('Trigger process for AMQSINQ4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSINQ4')  /* C       +
/*    APPID('QMQM/AMQ0INQ4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3INQ4')  /* RPG - ILE */

CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.SETPROCESS')     +
MQMNAME(&QMGRNAME)             +
REPLACE(*YES)               +
```

```
+
TEXT('Trigger process for AMQSSET4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSSET4')  /* C   */    +
/*    APPID('QMQM/AMQ0SET4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3SET4')  /* RPG - ILE */

CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')    +
MQMNAME(&QMGRNAME)             +
REPLACE(*YES)               +
+
TEXT('Trigger process for AMQSECH4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSECH4')  /* C   */    +
/*    APPID('QMQM/AMQ0ECH4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3ECH4')  /* RPG - ILE */

/*********************************************************************/
/*                                                                   */
/*  Normal return.                                                   */
/*                                                                   */
/*********************************************************************/
SNDPGMMSG  MSG('AMQSAMP4 Completed creating sample +
objects for ' *CAT &QMGRNAME)
RETURN
ENDPGM

/*********************************************************************/
/*                                                                   */
/* END OF AMQSAMP4                                                   */
/*                                                                   */
/*********************************************************************/
```

IBM i Alternative ways of administering IBM MQ for IBM i

Using CL commands is the preferred method of administering IBM MQ for IBM i. However, you can use various other administration methods including MQSC commands, PCF commands, control commands, and remote administration.

About this task

You normally use IBM i CL commands to administer IBM MQ for IBM i. For an overview of these commands, see “Managing IBM MQ for IBM i using CL commands” on page 374.

You can also use MQSC commands and PCF commands as described in the subtopics, and you can use control commands as described in “Administering IBM MQ for Multiplatforms using control commands” on page 10.

You can use IBM MQ instrumentation events to monitor the operation of queue managers. See Instrumentation events for information about IBM MQ instrumentation events and how to use them.

Use any of the administration methods described in the following subtopics as an alternative to using IBM i CL commands:

IBM i Local and remote administration on IBM i

You administer IBM MQ for IBM i objects locally or remotely.

About this task

Local administration means carrying out administration tasks on any queue managers that you have defined on your local system. In IBM MQ, you can consider this as local administration because no IBM MQ channels are involved, that is, the communication is managed by the operating system. To perform this type of task, you must either log on to the remote system and issue the commands from there, or create a process that can issue the commands for you.

IBM MQ supports administration from a single point through what is known as *remote administration*. Remote administration consists of sending programmable command format (PCF) control messages to the SYSTEM.ADMIN.COMMAND.QUEUE on the target queue manager.

There are a number of ways of generating PCF messages. These are described in the following steps.

Procedure

- Write a program using PCF messages. See "Administration using PCF commands on IBM i" on page 389.
- Write a program using the MQAI, which sends out PCF messages. See "Using the MQAI to simplify the use of PCFs" on page 37.
- Use the IBM MQ Explorer, available with IBM MQ for Windows, which allows you to use a graphical user interface (GUI) and generates the correct PCF messages. See "Using the IBM MQ Explorer with IBM MQ for IBM i " on page 390.
- Use **STRMQMMQSC** to send commands indirectly to a remote queue manager. See "Administering using MQSC commands on IBM i" on page 388.

  For example, you can issue a remote command to change a queue definition on a remote queue manager.

  Some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log on to the remote system and issue the commands from there or create a process that can issue the commands for you.

IBM i Administering using MQSC commands on IBM i

On IBM i, you create a list of commands in a Script file, then run the file by using the **STRMQMMQSC** command. You use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, namelists, channels, client connection channels, listeners, services, topics, and authentication information objects.

About this task

IBM MQ script (MQSC) commands are written in human-readable form, in EBCDIC text. You issue MQSC commands to a queue manager using the **STRMQMMQSC** IBM MQ CL command. This method is a batch method only, taking its input from a source physical file in the server library system. The default name for this source physical file is QMQSC.

Attention: Do not use the QTEMP library as the source library to STRMQMMQSC, as the usage of the QTEMP library is limited. You must use another library as an input file to the command.

For portability among IBM MQ environments, limit the line length in MQSC command files to 72 characters. Use the plus sign to indicate that the command is continued on the next line.

Object attributes specified in MQSC are shown in this topic in uppercase (for example, RQMNAME), although they are not case-sensitive.

Note:

1. The format of an MQSC file does not depend on its location in the file system.
2. MQSC attribute names are limited to eight characters.
3. MQSC commands are available on all IBM MQ platforms.

For a description of each MQSC command and its syntax, see MQSC commands.

Procedure

1. Create the QMQSC source file.

IBM MQ for IBM i does not supply a source file called QMQSC. To process MQSC commands you must create the QMQSC source file in a library of your choice, by issuing the following command:

```
CRTSRCPF FILE(MYLIB/QMQSC) RCDLEN(240) TEXT('IBM MQ - MQSC Source')
```

2. Work with the members.

   MQSC source is held in members within this source file. To work with the members enter the following command:

   ```
   WRKMBRPDM MYLIB/QMQSC
   ```

   You can now add new members and maintain existing ones.

   ```
   .
   .
   DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE +
   DESCR(' ') +
   PUT(ENABLED) +
   DEFPRTY(0) +
   DEFPSIST(NO) +
   GET(ENABLED) +
   MAXDEPTH(5000) +
   MAXMSGL(1024) +
   DEFSOPT(SHARED) +
   NOHARDENBO +
   USAGE(NORMAL) +
   NOTRIGGER;
   .
   .
   ```

   Figure 20. Extract from an MQSC command file, `myprog.in`, showing an MQSC command (DEFINE QLOCAL) with its attributes.

Related information

Administering IBM MQ using MQSC commands

IBM i Administration using PCF commands on IBM i

The purpose of IBM MQ programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way you can create queues and process definitions, and change queue managers, from a program.

PCF commands cover the same range of functions provided by MQSC commands. However, unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an IBM MQ message. Each command is sent to the target queue manager using the MQI function MQPUT in the same way as any other message. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned in another data structure. The application can then process the reply and act accordingly.

Briefly, these are some of the things the application programmer must specify to create a PCF command message:

Message descriptor

This is a standard IBM MQ message descriptor, in which:

- Message type (*MsgType*) is MQMT_REQUEST.
- Message format (*Format*) is MQFMT_ADMIN.

Application data

Contains the PCF message including the PCF header, in which:

- The PCF message type (*Type*) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. See "Using the MQAI to simplify the use of PCFs" on page 37 for further information.

For a complete description of the PCF data structures and how to implement them, see Structures for commands and responses.

IBM i Using the IBM MQ Explorer with IBM MQ for IBM i

Use this information to administer IBM MQ for IBM i using the IBM MQ Explorer.

IBM MQ for Windows (x86 platform), and IBM MQ for Linux (x86 and x86-64 platforms) provide an administration interface called the IBM MQ Explorer to perform administration tasks as an alternative to using CL, control, or MQSC commands.

The IBM MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows (x86 platform), or Linux (x86 and x86-64 platforms), by pointing the IBM MQ Explorer at the queue managers and clusters you are interested in.

With the IBM MQ Explorer, you can:

- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of IBM MQ objects such as queues, topics, and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel.
- View queue managers in a cluster.
- Check to see which applications, users, or channels have a particular queue open.
- Create a new queue manager cluster using the **Create New Cluster** wizard.
- Add a queue manager to a cluster using the **Add Queue Manager to Cluster** wizard.
- Manage the authentication information object, used with Transport Layer Security (TLS) channel security.

Using the online guidance, you can:

- Define and control various resources including queue managers, queues, channels, process definitions, client connection channels, listeners, topics, services, namelists, and clusters.
- Start or stop a queue manager and its associated processes.
- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.

Ensure that you have satisfied the following requirements before attempting to use the IBM MQ Explorer to manage IBM MQ on a server machine. Check that:

1. A command server is running for any queue manager being administered, started on the server by the CL command **STRMQMCSVR**.
2. A suitable TCP/IP listener exists for every remote queue manager. This is the IBM MQ listener started by the **STRMQMLSR** command.
3. The server connection channel, called `SYSTEM.ADMIN.SVRCONN`, exists on every remote queue manager. You must create this channel yourself. It is mandatory for every remote queue manager being administered. Without it, remote administration is not possible.
4. Verify that the `SYSTEM.MQEXPLORER.REPLY.MODEL` queue exists.

IBM i Managing the command server for remote administration on IBM i

Use this information to learn about the remote administration of IBM MQ for IBM i command server.

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command.

A command server is mandatory for all administration involving PCFs, the MQAI, and also for remote administration.

Note: For remote administration, you must ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager. Avoid this situation if at all possible.

There are separate control commands for starting and stopping the command server. You can perform the operations described in the following sections using the IBM MQ Explorer.

Starting and stopping the command server

To start the command server, use this CL command:

```
STRMQMCSVR MQMNAME('saturn.queue.manager')
```

where `saturn.queue.manager` is the queue manager for which the command server is being started.

To stop the command server, use one of the following CL commands:

1. ```
   ENDMQMCSVR MQMNAME('saturn.queue.manager') OPTION(*CNTRLD)
   ```

   to perform a controlled stop, where `saturn.queue.manager` is the queue manager for which the command server is being stopped. This is the default option, which means that the `OPTION(*CNTRLD)` can be omitted.

2. ```
   ENDMQMCSVR MQMNAME('saturn.queue.manager') OPTION(*IMMED)
   ```

   to perform an immediate stop, where `saturn.queue.manager` is the queue manager for which the command server is being stopped.

Displaying the status of the command server

For remote administration, ensure that the command server on the target queue manager is running. If it is not running, remote commands cannot be processed. Any messages containing commands are queued in the target queue manager's command queue `SYSTEM.ADMIN.COMMAND.QUEUE`.

To display the status of the command server for a queue manager, called here `saturn.queue.manager`, the CL command is:

```
DSPMQMCSVR MQMNAME('saturn.queue.manager')
```

Issue this command on the target machine. If the command server is running, the panel shown in Figure 21 on page 392 appears:

```
Display MQM Command Server (DSPMQMCSVR)


Queue manager name . . . . . > saturn.queue.manager

MQM Command Server Status. . . . > RUNNING




F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 21. Display MQM Command Server panel

IBM i Running web console commands

You must configure your environment as described in the following text, for the web console related Qshell commands to run correctly on IBM MQ for IBM i.

About this task

When the Qshell starts, it initializes internal tables for processing commands based on the CCSID of the job. For the web console related Qshell commands to run correctly on IBM i, you must configure your environment.

A locale is set by setting the LANG environment variable to the path name to a locale object. For example, to set the locale for US English, the LANG environment variable is set as follows:

```
LANG=/QSYS.LIB/EN_US.LOCALE
```

In the Qshell, you can check the setting by issue command set to list all the environment variables. Usually it is LANG which may impact the locale for the runtime environment. It may also has LC_ALL.

To run Qshell commands correctly, the locale environment setting must be consistent with your job setting.

Procedure

Use CL command DSPJOB JOB(JobNumber/USERProfile/JobName)

a) Select option 2 to display the job definition attributes.

b) The following attributes should be consistent with the LANG or LC_ALL environment setting

- Language identifier
- Country or region identifier
- Coded character set identifier

For example, if

```
LANG=/QSYS.LIB/FR_FR.LOCALE
```

your job attributes should be:

- Language ID . . . . . FRA
- Country or region ID . . . . . FR
- Coded character set ID 297

What to do next

For more information on national language support, see the IBM Documentation topic National language support (NLS) considerations.

IBM i Work management for IBM i

This information describes the way in which IBM MQ handles work requests, and details the options available for prioritizing and controlling the jobs associated with IBM MQ.

Warning

Do not alter IBM MQ work management objects unless you fully understand the concepts of IBM i and IBM MQ work management.

Additional information regarding subsystems and job descriptions can be found under Work Management in the IBM i product documentation. Pay particular attention to the sections on Starting jobs and Batch jobs.

IBM MQ for IBM i incorporates the IBM i UNIX environment and IBM i threads. Do **not** make any changes to the objects in the Integrated File System (IFS).

During normal operations, an IBM MQ queue manager starts a number of batch jobs to perform different tasks. By default these batch jobs run in the QMQM subsystem that is created when IBM MQ is installed.

Work management refers to the process of tailoring IBM MQ tasks to obtain the optimum performance from your system, or to make administration simpler.

For example, you can:

- Change the run-priority of jobs to make one queue manager more responsive than another.
- Redirect the output of a number of jobs to a particular output queue.
- Make all jobs of a certain type run in a specific subsystem.
- Isolate errors to a subsystem.

Work management is carried out by creating or changing the job descriptions associated with the IBM MQ jobs. You can configure work management for:

- An entire IBM MQ installation.
- Individual queue managers.
- Individual jobs for individual queue managers.

IBM i IBM MQ tasks for IBM i

This is a table of the IBM MQ for IBM i jobs and a brief description of each.

When a queue manager is running, you see some or all of the following batch jobs running under the QMQM user profile in the IBM MQ subsystem. The jobs are described briefly in Table 23 on page 393.

You can view all jobs connected to a queue manager using option 22 on the **Work with Queue Manager** (WRKMQM) panel. You can view listeners using the WRKMQMLSR command.

Table 23. IBM MQ tasks.

Job name	Function
AMQZMUC0	Utility manager. This job executes critical queue manager utilities, for example the journal chain manager.
AMQZXMA0	The execution controller that is the first job started by the queue manager. It handles MQCONN requests, and starts agent processes to process IBM MQ API calls.
AMQZFUMA	Object authority manager (OAM).

Table 23. IBM MQ tasks. (continued)	
Job name	**Function**
AMQZLAA0	Queue manager agents that perform most of the work for applications that connect to the queue manager using MQCNO_STANDARD_BINDING.
AMQZLSA0	Queue manager agent.
AMQZMUF0	Utility Manager
AMQZMGR0	Process controller. This job is used to start and manage listeners and services.
AMQZMUR0	Utility manager. This job executes critical queue manager utilities, for example the journal chain manager.
AMQFQPUB	Queued publish/subscribe daemon.
AMQFCXBA	Broker worker job.
RUNMQBRK	Broker control job.
AMQRMPPA	Channel process pooling job.
AMQCRSTA	TCP/IP-invoked channel responder.
AMQCRS6B	LU62 receiver channel and client connection (see note).
AMQRRMFA	Repository manager for clusters.
AMQCLMAA	Non-threaded TCP/IP listener.
AMQPCSEA	PCF command processor that handles PCF and remote administration requests.
RUNMQTRM	Trigger monitor.
RUNMQDLQ	Dead letter queue handler.
RUNMQCHI	The channel initiator.
RUNMQCHL	Sender channel job that is started for each sender channel.
RUNMQLSR	Threaded TCP/IP listener.
AMQRCMLA	Channel MQSC and PCF command processor.

Note: The LU62 receiver job runs in the communications subsystem and takes its runtime properties from the routing and communications entries that are used to start the job. See Initiated end (Receiver) for more information.

IBM i Work management objects on IBM i

When IBM MQ is installed, various objects are supplied in the QMQM library to assist with work management. These objects are the ones necessary for IBM MQ jobs to run in their own subsystem.

Sample job descriptions are provided for two of the IBM MQ batch jobs. If no specific job description is provided for an IBM MQ job, it runs with the default job description QMQMJOBD.

The work management objects that are supplied when you install IBM MQ are listed in Table 24 on page 395 and the objects created for a queue manager are listed in Table 25 on page 395

Note: The work management objects can be found in the QMQM library and the queue manager objects can be found in the queue manager library.

Table 24. Work management objects		
Name	**Type**	**Description**
AMQZLAA0	*JOBD	The job description that is used by the IBM MQ agent processes
AMQZLSA0	*JOBD	The isolated bindings queue manager agent
AMQZXMA0	*JOBD	The job description that is used by IBM MQ execution controllers
QMQM	*SBSD	The subsystem in which all IBM MQ jobs run
QMQM	*JOBQ	The job queue attached to the supplied subsystem
QMQMJOBD	*JOBD	The default IBM MQ job description, used if there is not a specific job description for a job
QMQMMSG	*MSGQ	The default message queue for IBM MQ jobs.
QMQMRUN20	*CLS	A class description for high priority IBM MQ jobs
QMQMRUN35	*CLS	A class description for medium priority IBM MQ jobs
QMQMRUN50	*CLS	A class description for low priority IBM MQ jobs

Table 25. Work management objects created for a queue manager		
Name	**Type**	**Description**
AMQA000000	*JRNRCV	Local journal receiver
AMQAJRN	*JRN	Local journal
AMQJRNINF	*USRSPC	User space that is updated with the latest journal receivers required for startup and media recovery of a queue manager. This user space can be queried by an application to determine which journal receivers require archiving and which can be safely deleted.
AMQAJRNMSG	*MSGQ	Local journal message queue
AMQCRC6B	*PGM	Program to start the LU6.2 connection
AMQRFOLD	*FILE	Migrated queue manager channel definition file
QMQMMSG	*MSGQ	Queue manager message queue

IBM i How IBM MQ uses work management objects on IBM i

This information describes the way in which IBM MQ uses the work management objects, and provides configuration examples.

Attention: Do not alter the job queue entry settings in the QMQM subsytem to limit the number of jobs allowed in the subsystem by priority. If you attempt to do this, you can stop essential IBM MQ jobs from running after they are submitted and cause the queue manager startup to fail.

To understand how to configure work management, you must first understand how IBM MQ uses job descriptions.

The job description used to start the job controls many attributes of the job. For example:

- The job queue on which the job is queued and on which subsystem the job runs.
- The routing data used to start the job and class that the job uses for its runtime parameters.
- The output queue that the job uses for print files.

The process of starting an IBM MQ job can be considered in three steps:

1. IBM MQ selects a job description.

   IBM MQ uses the following technique to determine which job description to use for a batch job:

   a. Look in the queue manager library for a job description with the same name as the job. See Understanding IBM MQ for IBM i queue manager library names for further details about the queue manager library.

   b. Look in the queue manager library for the default job description QMQMJOBD.

   c. Look in the QMQM library for a job description with the same name as the job.

   d. Use the default job description, QMQMJOBD, in the QMQM library.

2. The job is submitted to the job queue.

   Job descriptions supplied with IBM MQ have been set up, by default, to put jobs on to job queue QMQM in library QMQM. The QMQM job queue is attached to the supplied QMQM subsystem, so by default the jobs start running in the QMQM subsystem.

3. The job enters the subsystem and goes through the routing steps.

   When the job enters the subsystem, the routing data specified on the job description is used to find routing entries for the job.

   The routing data must match one of the routing entries defined in the QMQM subsystem, and this defines which of the supplied classes (QMQMRUN20, QMQMRUN35, or QMQMRUN50) is used by the job.

Note: If IBM MQ jobs do not appear to be starting, make sure that the subsystem is running and the job queue is not held,

If you have modified the IBM MQ work management objects, make sure everything is associated correctly. For example, if you specify a job queue other than QMQM/QMQM on the job description, make sure that an ADDJOBQE is performed for the subsystem, that is, QMQM.

You can create a job description for each job documented in Table 23 on page 393 using the following worksheet as an example:

```
What is the queue manager library name? __________
Does job description AMQZXMA0 exist in the queue manager library? Yes   No
Does job description QMQMJOBD exist in the queue manager library? Yes   No
Does job description AMQZXMA0 exist in the QMQM library?     Yes   No
Does job description QMQMJOBD exist in the QMQM library?     Yes   No
```

If you answer No to all these questions, create a global job description QMQMJOBD in the QMQM library.

The IBM MQ message queue

An IBM MQ message queue, QMQMMSG, is created in each queue manager library. Operating system messages are sent to this queue when queue manager jobs end and IBM MQ sends messages to the queue. For example, to report which journal receivers are needed at startup. Keep the number of messages in this message queue at a manageable size to make it easier to monitor.

IBM i Default system examples for IBM i

These examples show how an unmodified IBM MQ installation works when some of the standard jobs are submitted at queue manager startup time.

First, the AMQZXMA0 execution controller job starts.

1. Issue the **STRMQM** command for queue manager TESTQM.
2. IBM MQ searches the queue manager library QMTESTQM, firstly for job description AMQZXMA0, and then job description QMQMJOBD.

Neither of these job descriptions exist, so IBM MQ looks for job description AMQZXMA0 in the product library QMQM. This job description exists, so it is used to submit the job.

3. The job description uses the IBM MQ default job queue, so the job is submitted to job queue QMQM/QMQM.
4. The routing data on the AMQZXMA0 job description is QMQMRUN20, so the system searches the subsystem routing entries for one that matches that data.

   By default, the routing entry with sequence number 9900 has comparison data that matches QMQMRUN20, so the job is started with the class defined on that routing entry, which is also called QMQMRUN20.
5. The QMQM/QMQMRUN20 class has run priority set to 20, so the AMQZXMA0 job runs in subsystem QMQM with the same priority as most interactive jobs on the system.

IBM i Configuring work management examples on IBM i

Use this information to learn how you can change and create IBM MQ job descriptions to change the runtime attributes of IBM MQ jobs.

The key to the flexibility of IBM MQ work management lies in the two-tier way that IBM MQ searches for job descriptions:

- If you create or change job descriptions in a queue manager library, those changes override the global job descriptions in QMQM, but the changes are local and affect that particular queue manager alone.
- If you create or change global job descriptions in the QMQM library, those job descriptions affect all queue managers on the system, unless overridden locally for individual queue managers.

1. The following example increases the priority of channel control jobs for an individual queue manager.

   To make the repository manager and channel initiator jobs, AMQRRMFA and RUNMQCHI, run as quickly as possible for queue manager TESTQM, carry out the following steps:

   a. Create local duplicates of the QMQM/QMQMJOBD job description with the names of the IBM MQ processes that you want to control in the queue manager library. For example:

   ```
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
   NEWOBJ(RUNMQCHI)
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
   NEWOBJ(AMQRRMFA)
   ```

   b. Change the routing data parameter on the job description to ensure that the jobs use the QMQMRUN20 class.

   ```
   CHGJOBD JOBD(QMTESTQM/RUNMQCHI) RTGDTA('QMQMRUN20')
   CHGJOBD JOBD(QMTESTQM/AMQRRMFA) RTGDTA('QMQMRUN20')
   ```

   The AMQRRMFA and RUNMQCHI jobs for queue manager TESTQM now:

   - Use the new local job descriptions in the queue manager library
   - Run with priority 20, because the QMQMRUN20 class is used when the jobs enter the subsystem.
2. The following example defines a new run priority class for the QMQM subsystem.

   a. Create a duplicate class in the QMQM library, to allow other queue managers to access the class, by issuing the following command:

   ```
   CRTDUPOBJ OBJ(QMQMRUN20) FROMLIB(QMQM) OBJTYPE(*CLS) TOLIB(QMQM)
   NEWOBJ(QMQMRUN10)
   ```

   b. Change the class to have the new run priority by issuing the following command:

   ```
   CHGCLS CLS(QMQM/QMQMRUN10) RUNPTY(10)
   ```

   c. Add the new class definition to the subsystem by issuing the following command:

```
ADDRTGE SBSD(QMQM/QMQM) SEQNBR(8999) CMPVAL('QMQMRUN10') PGM(QSYS/QCMD)
CLS(QMQM/QMQMRUN10)
```

Note: You can specify any numeric value for the routing sequence number, but the values must be in sequential order. This sequence number tells the subsystem the order in which routing entries are to be searched for a routing data match.

d. Change the local or global job description to use the new priority class by issuing the following command:

```
CHGJOBD JOBD(QMQMlibname/QMQMJOBD) RTGDTA('QMQMRUN10')
```

Now all the queue manager jobs associated with the QMlibraryname use a run priority of 10.

3. The following example runs a queue manager in its own subsystem

To make all the jobs for queue manager TESTQM run in the QBATCH subsystem, carry out the following steps:

a. Create a local duplicate of the QMQM/QMQMJOBD job description in the queue manager library with the command

```
CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
```

b. Change the job queue parameter on the job description to ensure that the jobs use the QBATCH job queue.

```
CHGJOBD JOBD(QMTESTQM/QMQMJOBD) JOBQ(*LIBL/QBATCH)
```

Note: The job queue is associated with the subsystem description. If you find that the jobs are staying on the job queue, verify that the job queue definition is defined on the SBSD. Use the DSPSBSD command for the subsystem and take option 6, Job queue entries.

All jobs for queue manager TESTQM now:

- Use the new local default job description in the queue manager library
- Are submitted to job queue QBATCH.

To ensure that jobs are routed and prioritized correctly:

- Either create routing entries for the IBM MQ jobs in subsystem QBATCH, or
- Rely on a catch-all routing entry that calls QCMD, irrespective of what routing data is used.

  This option works only if the maximum active jobs option for job queue QBATCH is set to *NOMAX. The system default is 1.

4. The following example creates another IBM MQ subsystem

a. Create a duplicate subsystem in the QMQM library by issuing the following command:

```
CRTDUPOBJ OBJ(QMQM) FROMLIB(QMQM) OBJTYPE(*SBSD) TOLIB(QMQM) NEWOBJ(QMQM2)
```

b. Remove the QMQM job queue by issuing the following command:

```
RMVJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM)
```

c. Create a new job queue for the subsystem by issuing the following command:

```
CRTJOBQ JOBQ(QMQM/QMQM2) TEXT('Job queue for IBM MQ Queue Manager')
```

d. Add a job queue entry to the subsystem by issuing the following command:

```
ADDJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM2) MAXACT(*NOMAX)
```

e. Create a duplicate QMQMJOBD in the queue manager library by issuing the following command:

```
CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMlibraryname)
```

f. Change the job description to use the new job queue by issuing the following command:

```
CHGJOBD JOBD(QMlibraryname/QMQMJOBD) JOBQ(QMQM/QMQM2)
```

g. Start the subsystem by issuing the following command:

```
STRSBS SBSD(QMQM/QMQM2)
```

Note:

a. You can specify the subsystem in any library. If for any reason the product is reinstalled, or the QMQM library is replaced, any changes you made are removed.

b. All the queue manager jobs associated with the QMlibraryname now run under subsystem QMQM2.

IBM i Availability, backup, recovery, and restart on IBM i

Use this information to understand how IBM MQ for IBM i uses the IBM i journaling support to help its backup and restore strategy.

You must be familiar with standard IBM i backup and recovery methods, and with the use of journals and their associated journal receivers on IBM i, before reading this section. For information on these topics, see Backup and recovery.

To understand the backup and recovery strategy, you first need to understand how IBM MQ for IBM i organizes its data in the IBM i file system and the integrated file system (IFS).

IBM MQ for IBM i holds its data in an individual library for each queue manager instance, and in stream files in the IFS file system.

The queue manager specific libraries contain journals, journal receivers, and objects required to control the work management of the queue manager. The IFS directories and files contain IBM MQ configuration files, the descriptions of IBM MQ objects, and the data they contain.

Every change to these objects, that is recoverable across a system failure, is recorded in a journal *before* it is applied to the appropriate object. This has the effect that such changes can be recovered by replaying the information recorded in the journal.

You can configure IBM MQ for IBM i to use multiple queue manager instances on different servers to provide increased queue manager availability and speed up recovery in the case of a server or queue manager failure.

IBM i Queue manager journals on IBM i

Use this information to understand how IBM MQ for IBM i uses journals in its operation to control updates to local objects.

Each queue manager library contains a journal for that queue manager, and the journal has the name QM *GRLIB*/AMQ *A* JRN, where QM *GRLIB* is the name of the queue manager library, and *A* is a letter, A in the case of a single instance queue manager, that is unique to the queue manager instance.

QM *GRLIB* takes the name QM, followed by the name of the queue manager in a unique form. For example, a queue manager named TEST has a queue manager library named QMTEST. The queue manager library can be specified when creating a queue manager using the **CRTMQM** command.

Journals have associated journal receivers that contain the information being journaled. The receivers are objects to which information can only be appended and will fill up eventually.

Journal receivers use up valuable disk space with out-of-date information. However, you can place the information in permanent storage to minimize this problem. One journal receiver is attached to the journal at any particular time. If the journal receiver reaches its predetermined threshold size, it is detached and replaced by a new journal receiver. You can specify the threshold of journal receivers when you create a queue manager using **CRTMQM** and the **THRESHOLD** parameter.

The journal receivers associated with the local IBM MQ for IBM i journal exist in each queue manager library, and adopt a naming convention as follows:

```
AMQ Arnnnnn
```

where

A
is a letter A-Z. It is A for single instance queue managers. It varies for different instances of a multi-instance queue manager.

nnnnn
is decimal `00000 to 99999` that is incremented by 1 for the next journal in the sequence.

r
is decimal `0 to 9`, that is incremented by 1 each time a receiver is restored.

The sequence of the journals is based on date. However, the naming of the next journal is based on the following rules:

1. AMQArnnnnn goes to `AMQAr(nnnnn+1)`, and nnnnn wraps when it reaches 99999. For example, AMQA099999 goes to AMQA000000, and AMQA999999 goes to AMQA900000.
2. If a journal with a name generated by rule 1 already exists, the message CPI70E3 is sent to the QSYSOPR message queue and automatic receiver switching stops.

   The currently-attached receiver continues to be used until you investigate the problem and manually attach a new receiver.
3. If no new name is available in the sequence (that is, all possible journal names are on the system) you need to do both of the following:

   a. Delete journals no longer needed (see "Journal management on IBM i" on page 404).
   b. Record the journal changes into the latest journal receiver using (**RCDMQMIMG**) and then repeat the previous step. This allows the old journal receiver names to be reused.

The AMQAJRN journal uses the `MNGRCV(*SYSTEM)` option to enable the operating system to automatically change journal receivers when the threshold is reached. For more information on how the system manages receivers, see *IBM i Backup and Recovery*.

The journal receiver's default threshold value is 100,000 KB. You can set this to a larger value when you create the queue manager. The initial value of the `LogReceiverSize` attribute is written to the `LogDefaults` stanza of the `mqs.ini` file.

When a journal receiver extends beyond its specified threshold, the receiver is detached and a new journal receiver is created, inheriting attributes from the previous receiver. Changes to the `LogReceiverSize` or LogASP attributes after a queue manager has been created are ignored when the system automatically attaches a new journal receiver

See Changing IBM MQ configuration information on Multiplatforms for further details on configuring the system.

If you need to change the size of journal receivers after the queue manager has been created, create a new journal receiver and set its owner to QMQM using the following commands:

```
CRTJRNRCV JRNRCV(QM GRLIB/AMQ Arnnnnn) THRESHOLD(xxxxxx) +
TEXT('MQM LOCAL JOURNAL RECEIVER')
CHGOBJOWN OBJ(QM GRLIB/AMQ Arnnnnn) OBJTYPE(*JRNRCV) NEWOWN(QMQM)
```

where

QMGRLIB
Is the name of your queue manager library

A
Is the instance identifier (usually A).

rnnnnn
Is the next journal receiver in the naming sequence described previously

xxxxxx
Is the new receiver threshold (in KB)

Note: The maximum size of the receiver is governed by the operating system. To check this value look at the THRESHOLD keyword on the **CRTJRNRCV** command.

Now attach the new receiver to the AMQAJRN journal with the command:

```
CHGJRN JRN(QMGRLIB/AMQ A JRN) JRNRCV(QMGRLIB/AMQ Annnnnn)
```

See "Journal management on IBM i" on page 404 for details on how to manage these journal receivers.

IBM i *Queue manager journal usage on IBM i*

Use this information to understand how IBM MQ for IBM i uses journals in its operation to control updates to local objects.

Persistent updates to message queues happen in two stages. The records representing the update are first written to the journal, then the queue file is updated.

The journal receivers can therefore become more up to date than the queue files. To ensure that restart processing begins from a consistent point, IBM MQ uses checkpoints.

A checkpoint is a point in time when the record described in the journal is the same as the record in the queue. The checkpoint itself consists of the series of journal records needed to restart the queue manager. For example, the state of all transactions (that is, units of work) active at the time of the checkpoint.

Checkpoints are generated automatically by IBM MQ. They are taken when the queue manager starts and shuts down, and after a certain number of operations are logged.

You can force a queue manager to take a checkpoint by issuing the RCDMQMIMG command against all objects on a queue manager and displaying the results, as follows:

```
RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(Q_MGR_NAME) DSPJRNDTA(*YES)
```

As the queues handle further messages, the checkpoint record becomes inconsistent with the current state of the queues.

When IBM MQ is restarted, it locates the latest checkpoint record in the log. This information is held in the checkpoint file that is updated at the end of every checkpoint. The checkpoint record represents the most recent point of consistency between the log and the data. The data from this checkpoint is used to rebuild the queues as they existed at the checkpoint time. When the queues are re-created, the log is then played forward to bring the queues back to the state they were in before system failure or close down.

To understand how IBM MQ uses the journal, consider the case of a local queue called TESTQ in the queue manager TEST. This is represented by the IFS file:

```
/QIBM/UserData/mqm/qmgrs/TEST/queues
```

If a specified message is put on this queue, and then retrieved from the queue, the actions that take place are shown in Figure Figure 22 on page 402.

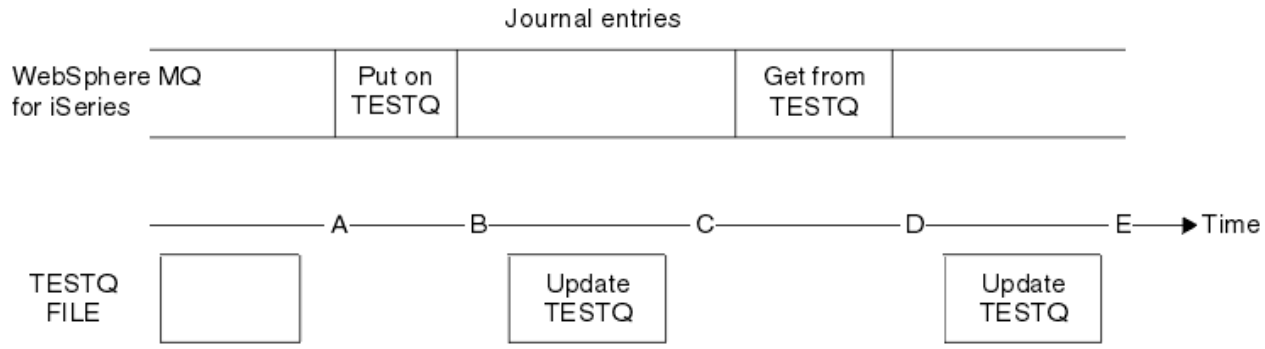


Figure 22. Sequence of events when updating MQM objects

The five points, A through E, shown in the diagram represent points in time that define the following states:

A
The IFS file representation of the queue is consistent with the information contained in the journal.

B
A journal entry is written to the journal defining a `Put` operation on the queue.

C
The appropriate update is made to the queue.

D
A journal entry is written to the journal defining a `Get` operation from the queue.

E
The appropriate update is made to the queue.

The key to the recovery capabilities of IBM MQ for IBM i is that the user can save the IFS file representation of TESTQ as at time A, and subsequently recover the IFS file representation of TESTQ as at time E, by restoring the saved object and replaying the entries in the journal from time A onwards.

This strategy is used by IBM MQ for IBM i to recover persistent messages after system failure. IBM MQ remembers a particular entry in the journal receivers, and ensures that on startup it replays the entries in the journals from this point onwards. This startup entry is periodically recalculated so that IBM MQ only has to perform the minimum necessary replay on the next startup.

IBM MQ provides individual recovery of objects. All persistent information relating to an object is recorded in the local IBM MQ for IBM i journals. Any IBM MQ object that becomes damaged or corrupt can be completely rebuilt from the information held in the journal.

For more information on how the system manages receivers, see "Availability, backup, recovery, and restart on IBM i" on page 399.

IBM i *Media images on IBM i*

On IBM i, a media image is a complete copy of an IBM MQ object that is recorded in the journal. Some corrupt or damaged objects can be automatically recovered from their media image.

An IBM MQ object of long duration can represent a large number of journal entries, going back to the point at which it was created. To avoid this, IBM MQ for IBM i has the concept of a media image of an object.

This media image is a complete copy of the IBM MQ object recorded in the journal. If an image of an object is taken, the object can be rebuilt by replaying journal entries from this image onwards. The entry in the journal that represents the replay point for each IBM MQ object is referred to as its media recovery entry. IBM MQ keeps track of the:

- Media recovery entry for each queue manager object.
- Oldest entry from within this set (see error message AMQ7462 in "Journal management on IBM i" on page 404 for details.

Images of the *CTLG object and the *MQM object are taken regularly because these objects are crucial to queue manager restart.

Images of other objects are taken when convenient. By default, images of all objects are taken when a queue manager is shut down using the **ENDMQM** command with parameter ENDCCTJOB(*YES). This operation can take a considerable amount of time for very large queue managers. If you need to shut down quickly, specify parameter RCDMQMIMG(*NO) with ENDCCTJOB(*YES). In such cases, you are recommended to record a complete media image in the journals after the queue manager has been restarted, using the following command:

```
RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(Q_MGR_NAME)
```

IBM MQ automatically records an image of an object, if it finds a convenient point at which an object can be compactly described by a small entry in the journal. However, this might never happen for some objects, for example, queues that consistently contain large numbers of messages.

Rather than allow the date of the oldest media recovery entry to continue for an unnecessarily long period, use the IBM MQ command RCDMQMIMG, which enables you to take an image of selected objects manually.

Recovery from media images

IBM MQ automatically recovers some objects from their media image if it is found that they are corrupt or damaged. In particular, this applies to the special *MQM and *CTLG objects as part of the normal queue manager startup. If any syncpoint transaction was incomplete at the time of the last shutdown of the queue manager, any queue affected is also recovered automatically, in order to complete the startup operation.

You must recover other objects manually, using the IBM MQ command RCRMQMOBJ. This command replays the entries in the journal to re-create the IBM MQ object. Should an IBM MQ object become damaged, the only valid actions are to delete it or re-create it by this method. Note, however, that nonpersistent messages cannot be recovered in this fashion.

IBM i *Checkpoints on IBM MQ for IBM i*

Checkpoints are taken at various times to provide a known consistent start point for recovery.

The checkpoint thread within process AMQZMUC0 is responsible for taking the checkpoint at the following points:

- Queue manager startup (STRMQM).
- Queue manager shutdown (ENDMQM).
- After a period of time has elapsed since the last checkpoint (the default period is 30 minutes) and a minimum number of log records have been written since the previous checkpoint (the default value is 100).
- After a number of log records have been written. The default value is 10 000.
- After the journal threshold size has been exceeded and a new journal receiver has been automatically created.
- When a full media image is taken with:

  ```
  RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(Q_MGR_NAME) DSPJRNDTA(*YES)
  ```

IBM i Backups of IBM MQ for IBM i data

Use this information to understand the two types of IBM MQ backup for each queue manager.

For each queue manager, there are two types of IBM MQ backup to consider:

- Data and journal backup.

  To ensure that both sets of data are consistent, do this only after shutting down the queue manager.

- Journal backup.

  You can do this while the queue manager is active.

For both methods, you need to find the names of the queue manager IFS directory and the queue manager library. You can find these in the IBM MQ configuration file (mqs.ini). For more information, see Changing IBM MQ configuration information on Multiplatforms.

Use the following procedures to do both types of backup:

Data and journal backup of a particular queue manager

Note: Do not use a save-while-active request when the queue manager is running. Such a request cannot complete unless all commitment definitions with pending changes are committed or rolled back. If this command is used when the queue manager is active, the channel connections might not end normally. Always use the following procedure.

1. Create an empty journal receiver, using the command:

   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```

2. Use the **RCDMQMIMG** command to record an MQM image for all IBM MQ objects, and then force a checkpoint using the command:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```

3. End channels and ensure that the queue manager is not running. If your queue manager is running, stop it with the **ENDMQM** command.
4. Backup the queue manager library by issuing the following command:

   ```
   SAVLIB LIB(QMTEST)
   ```

5. Back up the queue manager IFS directories by issuing the following command:

   ```
   SAV DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/test'))
   ```

Journal backup of a particular queue manager

Because all relevant information is held in the journals, as long as you perform a full save at some time, partial backups can be performed by saving the journal receivers. These record all changes since the time of the full backup and are performed by issuing the following commands:

1. Create an empty journal receiver, using the command:

   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```

2. Use the **RCDMQMIMG** command to record an MQM image for all IBM MQ objects, and then force a checkpoint using the command:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```

3. Save the journal receivers using the command:

   ```
   SAVOBJ OBJ(AMQ*) LIB(QMTEST) OBJTYPE(*JRNRCV) .........
   ```

A simple backup strategy is to perform a full backup of the IBM MQ libraries every week, and perform a daily journal backup. This, of course, depends on how you have set up your backup strategy for your enterprise.

IBM i *Journal management on IBM i*

As part of your backup strategy, take care of your journal receivers. It is useful to remove journal receivers from the IBM MQ libraries for various reasons:

- To release space; this applies to all journal receivers
- To improve the performance when starting (STRMQM)

- To improve the performance of recreating objects (RCRMQMOBJ)

Before deleting a journal receiver, you must take care that you have a backup copy and that you no longer need the journal receiver.

Journal receivers can be removed from the queue manager library *after* they have been detached from the journals and saved, provided that they are available for restoration if needed for a recovery operation.

The concept of journal management is shown in Figure 23 on page 405.

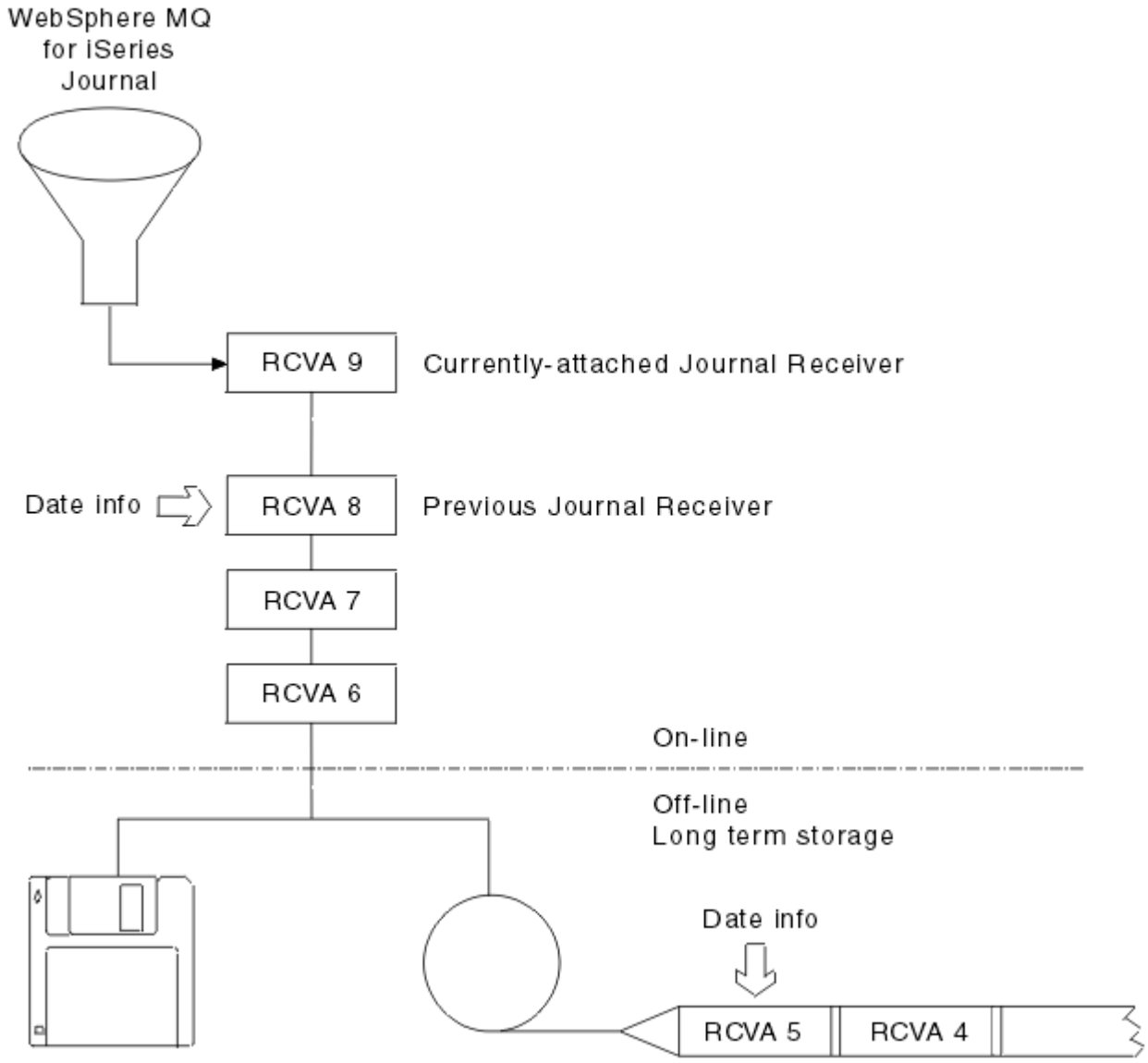


Figure 23. Journaling on IBM i

It is important to know how far back in the journals IBM MQ is likely to need to go, in order to determine when a journal receiver that has been backed up can be removed from the queue manager library, and when the backup itself can be discarded.

IBM MQ issues two messages to the queue manager message queue (QMQMMSG in the queue manager library) to help determine this time. These messages are issued when it starts, when it changes a local journal receiver, and you use RCDMQIMG to force a checkpoint. The two messages are:

AMQ7460
Startup recovery point. This message defines the date and time of the startup entry from which IBM MQ replays the journal in the event of a startup recovery pass. If the journal receiver that contains this

record is available in the IBM MQ libraries, this message also contains the name of the journal receiver containing the record.

AMQ7462
Oldest media recovery entry. This message defines the date and time of the oldest entry to use to re-create an object from its media image.

The journal receiver identified is the oldest one required. Any other IBM MQ journal receivers with older creation dates are no longer needed. If only stars are displayed, you need to restore backups from the date indicated to determine which is the oldest journal receiver.

When these messages are logged, IBM MQ also writes a user space object to the queue manager library that contains only one entry: the name of the oldest journal receiver that needs to be kept on the system. This user space is called AMQJRNINF, and the data is written in the format:

```
JJJJJJJJJJLLLLLLLLLLYYYYMMDDHHMMSSmmm
```

where:

JJJJJJJJJJ
Is the oldest receiver name that IBM MQ still needs.

LLLLLLLLLL
Is the journal receiver library name.

YYYY
Is the year of the oldest journal entry that IBM MQ needs.

MM
Is the month of the oldest journal entry that IBM MQ needs.

DD
Is the day of the oldest journal entry that IBM MQ needs.

HH
Is the hour of the oldest journal entry that IBM MQ needs.

SS
Is the seconds of the oldest journal entry that IBM MQ needs.

mmm
Is the milliseconds of the oldest journal entry that IBM MQ needs.

When the oldest journal receiver has been deleted from the system, this user space contains asterisks (*) for the journal receiver name.

Note: Periodically performing `RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES)` can save startup time for IBM MQ and reduce the number of local journal receivers you need to save and restore for recovery.

IBM MQ for IBM i does not refer to the journal receivers unless it is performing a recovery pass either for startup, or for recreating an object. If it finds that a journal it requires is not present, it issues message AMQ7432 to the queue manager message queue (QMQMMSG), reporting the time and date of the journal entry it requires to complete the recovery pass.

If this happens, restore all journal receivers that were detached after this date from the backup, to allow the recovery pass to succeed.

Keep the journal receiver that contains the startup entry, and any subsequent journal receivers, available in the queue manager library.

Keep the journal receiver containing the oldest `Media Recovery Entry`, and any subsequent journal receivers, available at all times, and either present in the queue manager library or backed-up.

When you force a checkpoint:

- If the journal receiver named in AMQ7460 is not advanced, this indicates that there is an incomplete unit of work that needs to be committed or rolled back.

- If the journal receiver named in AMQ7462 is not advanced, this indicates that there are one or more damaged objects.

IBM i *Restoring a complete queue manager (data and journals) on IBM i*

Use this information to restore one or more queue managers from a backup or from a remote machine.

If you need to recover one or more IBM MQ queue managers from a backup, perform the following steps.

1. Quiesce the IBM MQ queue managers.
2. Locate your latest backup set, consisting of your most recent full backup and subsequently backed up journal receivers.
3. Perform a RSTLIB operation, from the full backup, to restore the IBM MQ data libraries to their state at the time of the full backup, by issuing the following commands:

   ```
   RSTLIB LIB(QMQRLIB1) .........
   RSTLIB LIB(QMQRLIB2) .........
   ```

   If a journal receiver was partially saved in one journal backup, and fully saved in a subsequent backup, restore only the fully saved one. Restore journals individually, in chronological order.
4. Perform an RST operation to restore the IBM MQ IFS directories to the IFS file system, using the following command:

   ```
   RST DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/testqm')) ...
   ```
5. Start the message queue manager. This replays all journal records written since the full backup and restores all the IBM MQ objects to the consistent state at the time of the journal backup.

If you want to restore a complete queue manager on a different machine, use the following procedure to restore everything from the queue manager library. (We use TEST as the sample queue manager name.)

1. `CRTMQM TEST`
2. `DLTLIB LIB(QMTEST)`
3. `RSTLIB SAVLIB(QMTEST) DEV(*SAVF) SAVF(QMGRLIBSAV)`
4. Delete the following IFS files:

   ```
   /QIBM/UserData/mqm/qmgrs/TEST/QMQMCHKPT
   /QIBM/UserData/mqm/qmgrs/TEST/qmanager/QMQMOBJCAT
   /QIBM/UserData/mqm/qmgrs/TEST/qmanager/QMANAGER
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.AUTH.DATA.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CHANNEL.INITQ/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.COMMAND.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.REPOSITORY.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.TRANSMIT.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.PENDING.DATA.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.ADMIN.COMMAND.QUEUE/q
   ```
5. `STRMQM TEST`
6. `RCRMQMOBJ OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(TEST)`

IBM i *Restoring journal receivers for a particular queue manager on IBM i*

Use this information to understand the different ways to restore journal receivers.

The most common action is to restore a backed-up journal receiver to a queue manager library, if a receiver that has been removed is needed again for a subsequent recovery function.

This is a simple task, and requires the journal receivers to be restored using the standard IBM i RSTOBJ command:

```
RSTOBJ OBJ(QMQMDATA/AMQA000005) OBJTYPE(*JRNRCV) .........
```

A series of journal receivers might need to be restored, rather than a single receiver. For example, AMQA000007 is the oldest receiver in the IBM MQ libraries, and both AMQA000005 and AMQA000006 need to be restored.

In this case, restore the receivers individually in reverse chronological order. This is not always necessary, but is good practice. In severe situations, you might need to use the IBM i command WRKJRNA to associate the restored journal receivers with the journal.

When restoring journals, the system automatically creates an attached journal receiver with a new name in the journal receiver sequence. However, the new name generated might be the same as a journal receiver you need to restore. Manual intervention is needed to overcome this problem; to create a new name journal receiver in sequence, and new journal before restoring the journal receiver.

For example, consider the problem with saved journal AMQAJRN and the following journal receivers:

- AMQA000000
- AMQA100000
- AMQA200000
- AMQA300000
- AMQA400000
- AMQA500000
- AMQA600000
- AMQA700000
- AMQA800000
- AMQA900000

When restoring journal AMQAJRN to a queue manager library, the system automatically creates journal receiver AMQA000000. This automatically generated receiver conflicts with one of the existing journal receivers (AMQA000000) you want to restore, which you cannot restore.

The solution is:

1. Manually create the next journal receiver (see “Queue manager journals on IBM i” on page 399):

   ```
   CRTJRNRCV JRNRCV(QMQRLIB/AMQA900001) THRESHOLD(XXXXX)
   ```

2. Manually create the journal with the journal receiver:

   ```
   CRTJRN JRN(QMGRLIB/AMQAJRN) MNGRCV(*SYSTEM) +
   JRNRCV(QMGRLIB/AMQA9000001) MSGQ(QMGRLIB/AMQAJRNMSG)
   ```

3. Restore the local journal receivers AMQA000000 to AMQA900000.

IBM i Multi-instance queue managers on IBM i

Multi-instance queue managers improve availability by automatically switching to a standby server if the active server fails. The active and standby servers are multiple instances of the same queue manager; they share the same queue manager data. If the active instance fails you need to transfer its journal to the standby that takes over so that the queue manager can rebuild its queues.

Configure the IBM i systems you are running multi-instance queue managers on so that, if the active queue manager instance fails, the journal it is using is available to the standby instance that takes over. You can design your own configuration and administration tasks to make the journal from the active instance available to the instance that takes over. If you do not want to lose messages, your design must ensure the standby journal is consistent with the active journal at the point of failure. You can adapt your design from one of the two configurations that are described with examples in subsequent topics that do maintain consistency.

1. Mirror the journal from the system that is running the active queue manager instance to the systems that are running standby instances.

2. Place the journal in an Independent Auxiliary Storage Pool (IASP) that is transferable from the system running the active instance to a standby instance.

The first solution requires no additional hardware or software as it uses basic ASPs. The second solution requires switchable IASPs which need IBM i clustering support that is available as a separately priced IBM i License Product 5761-SS1 Option 41.

IBM i *Reliability and availability on IBM i*

Multi-instance queue managers aim to improve the availability of applications. Technological and physical constraints mean you need different solutions to meet the demands of disaster recovery, backing up queue managers and continuous operation.

In configuring for reliability and availability you trade off a large number of factors, resulting in four distinct design points:

Disaster recovery
Optimized for recovery after a major disaster that destroys all your local assets.

Disaster recovery on IBM i is often based on geographic mirroring of IASP.

Backup
Optimized for recovery after a localized failure, commonly a human error or some unforeseen technical problem.

IBM MQ provides backup queue managers to back up queue managers periodically. You could also use asynchronous replication of queue manager journals to improve the currency of the backup.

Availability
Optimized for restoring operations quickly giving the appearance of a nearly uninterrupted service following foreseeable technical failures such as a server or disk failure.

Recovery is typically measured in minutes, with detection sometimes taking longer than the recovery process. A multi-instance queue manager assists you in configuring for *availability*.

Continuous operation
Optimized for providing an uninterrupted service.

Continuous operation solutions have to solve the detection problem, and nearly always involve submitting the same work through more than one system and either using the first result, or if correctness is a major consideration, comparing at least two outcomes.

A multi-instance queue manager assists you in configuring for *availability*. One instance of the queue manager is active at a time. Switching over to a standby instance takes from a little more than ten seconds to a fifteen minutes or more, depending on how the system is configured, loaded and tuned.

A multi-instance queue manager can give the appearance of a nearly uninterrupted service if used with reconnectable IBM MQ MQI clients, which are able to continue processing without the application program necessarily being aware of a queue manager outage; see the topic Automated client reconnection.

IBM i *Components of a high availability solution on IBM i*

Construct a high availability solution using multi-instance queue managers by providing robust networked storage for queue manager data, journal replication or robust IASP storage for queue manager journals, and using reconnectable clients, of applications configured as restartable queue manager services.

A multi-instance queue manager reacts to the detection of queue manager failure by resuming the startup of another queue manager instance on another server. To complete its startup, the instance needs access to the shared queue manager data in networked storage, and to its copy of the local queue manager journal.

To create a high availability solution, you need to manage the availability of the queue manager data, the currency of the local queue manager journal, and either build reconnectable client applications, or deploy your applications as queue manager services to restart automatically when the queue manager resumes. Automatic client reconnect is not supported by IBM MQ classes for Java.

Queue manager data

Place queue manager data onto networked storage that is shared, highly available and reliable, possibly by using RAID level 1 disks or greater. The file system needs to meet the requirements for a shared file system for multi-instance queue managers; for more information about the requirements for shared file systems, see Requirements for shared file systems. Network File System 4 (NFS4) is a protocol that meets these requirements.

Queue manager journals

You also need to configure the IBM i journals used by the queue manager instances so that the standby instance is able to restore its queue manager data to a consistent state. For uninterrupted service, this means you must restore the journals to their state when the active instance failed. Unlike backup or disaster recovery solutions, restoring journals to an earlier checkpoint is not sufficient.

You cannot physically share journals between multiple IBM i systems on networked storage. To restore queue manager journals to the consistent state at the point of failure, you either need to transfer the physical journal that was local to the active queue manager instance at the time of failure to the new instance that has been activated, or a maintain mirrors of the journal on running standby instances. The mirrored journal is a remote journal replica that has been kept exactly in sync with the local journal belonging to the failed instance.

Three configurations are starting points for designing how you manage the journals for a multi-instance queue manager,

1. Using synchronized journal replication (journal mirroring) from the active instance ASP, to the standby instances ASPs.
2. Transferring an IASP you have configured to hold the queue manager journal from the active instance to the standby instance that is taking over as the active instance.
3. Using synchronized secondary IASP mirrors.

See ASP options, for more information on putting queue manager data onto an iASP, in the IBM MQ IBM i CRTMQM command.

Also see High availability in the IBM i information in IBM Documentation.

Applications

To build a client to automatically reconnect to the queue manager when the standby queue manager resumes, connect your application to the queue manager using `MQCONNX` and specify `MQCNO_RECONNECT_Q_MGR` in the **`MQCNO`** `Options` field. See, High availability sample programs for three sample programs using reconnectable clients, and Application recovery for information about designing client applications for recovery.

IBM i *Creating a network share for queue manager data using NetServer on IBM i*

Create a network share on an IBM i server for storing queue manager data. Set up connections from two servers, which are going to host queue manager instances, to access the network share.

Before you begin

- You require three IBM i servers for this task. The network share is defined on one of the servers, GAMMA. The other two servers, ALPHA and BETA, are to connect to GAMMA.
- Install IBM MQ on all three servers.
- Install the System i Navigator; see System i Navigator.

About this task

- Create the queue manager directory on GAMMA and set the correct ownership and permissions for the user profiles QMQM and QMQMADM. The directory and permission are easily created by installing IBM MQ on GAMMA.
- Use System i Navigator to create a share to the queue manager data directory on GAMMA.
- Create directories on ALPHA and BETA that point to the share.

Procedure

1. On GAMMA, create the directory to host the queue manager data with the QMQM user profile as the owner, and QMQMADM as the primary group.

   Tip:

   A quick and reliable way to create the directory with the correct permissions is to install IBM MQ on GAMMA.

   Later, if you do not want to run IBM MQ on GAMMA, uninstall IBM MQ. After uninstallation, the directory /QIBM/UserData/mqm/qmgrs remains on GAMMA with the owner QMQM user profile, and QMQMADM the primary group.

   The task uses the /QIBM/UserData/mqm/qmgrs directory on GAMMA for the share.
2. Start the System i Navigator **Add connection** wizard and connect to the GAMMA system.
   a) Double-click the **System i Navigator** icon on your Windows desktop.
   b) Click **Yes** to create a connection.
   c) Follow the instructions in the **Add Connection** wizard and create a connection from the IBM i system to GAMMA.

      The connection to GAMMA is added to **My Connections**.
3. Add a new file share on GAMMA.
   a) In the **System i Navigator** window, click the File Shares folder in My Connections/GAMMA/File Systems.
   b) In the **My Tasks** window, click **Manage IBM i NetServer shares**.

      A new window, **IBM i NetServer - GAMMA**, opens on your desktop and shows shared objects.
   c) Right-click the Shared Objects folder> **File** > **New** > **File**.

      A new window, **IBM i NetServer File Share - GAMMA**, opens.
   d) Give the share a name, WMQ for example.
   e) Set the access control to Read/Write.
   f) Select the **Path name** by browsing to the /QIBM/UserData/mqm/qmgrs directory you created earlier, and click **OK**.

      The **IBM i NetServer File Share - GAMMA** window closes, and WMQ is listed in the shared objects window.
4. Right click **WMQ** in the shared objects window. Click **File** > **Permissions**.

   A window opens, **Qmgrs Permissions - GAMMA**, for the object /QIBM/UserData/mqm/qmgrs.
   a) Check the following permissions for QMQM, if they are not already set:

      ```
      Read
      Write
      Execute
      Management
      Existence
      Alter
      ```

Reference

b) Check the following permissions for QMQMADM, if they are not already set:

Read
Write
Execute
Reference

c) Add other user profiles that you want to give permissions to /QIBM/UserData/mqm/qmgrs.

For example, you might give the default user profile (Public) Read and Execute permissions to /QIBM/UserData/mqm/qmgrs.

5. Check that all the user profiles that are granted access to /QIBM/UserData/mqm/qmgrs on GAMMA have the same password as they do on the servers that access GAMMA.

In particular, ensure that the QMQM user profiles on other servers, which are going to access the share, have the same password as the QMQM user profile on GAMMA.

Tip: Click the My Connections/GAMMA/Users and Groups folder in the System i Navigator to set the passwords. Alternatively, use the **CHFUSRPRF** and **CHGPWD** commands.

Results

Check you can access GAMMA from other servers using the share. If you are doing the other tasks, check you can access GAMMA from ALPHA and BETA using the path /QNTC/GAMMA/WMQ. If the /QNTC/GAMMA directory does not exist on ALPHA or BETA then you must create the directory. Depending on the NetServer domain, you might have to IPL ALPHA or BETA before creating the directory.

```
CRTDIR DIR('/QNTC/GAMMA')
```

When you have checked that you have access to /QNTC/GAMMA/WMQ from ALPHA or BETA, issuing the command, CRTMQM MQMNAME('QM1') MQMDIRP('/QNTC/GAMMA/WMQ') creates /QIBM/UserData/mqm/qmgrs/QM1 on GAMMA.

What to do next

Create a multi-instance queue manager by following the steps in either of the tasks, "Creating a multi-instance queue manager using journal mirroring and NetServer on IBM i" on page 422 or "Converting a single instance queue manager to a multi-instance queue manager using NetServer and journal mirroring on IBM i" on page 426.

IBM i *Failover performance on IBM i*

The time it takes to detect a queue manager instance has failed, and then to resume processing on a standby can vary between tens of seconds to fifteen minutes or more depending on the configuration. Performance needs to be a major consideration in designing and testing a high availability solution.

There are advantages and disadvantages to weigh up in deciding whether to configure a multi-instance queue manager to use journal replication, or to use an IASP. Mirroring requires the queue manager to write synchronously to a remote journal. From a hardware point of view, this need not affect performance, but from a software perspective there is a greater pathlength involved in writing to a remote journal than just to a local journal, and this might be expected to reduce the performance of a running queue manager to some extent. However, when the standby queue manager takes over, the delay in synchronizing its local journal from the remote journal maintained by the active instance before it failed, is typically small in comparison to the time it takes for IBM i to detect and transfer the IASP to the server running the standby instance of the queue manager. IASP transfer times can be as much as ten to fifteen minutes rather than being completed in seconds. The IASP transfer time depends on the number of objects that need to be *varied-on* when the IASP is transferred to the standby system and the size of the access paths, or indexes, that need to be merged.

When the standby queue manager takes over, the delay in synchronizing its local journal from the remote journal maintained by the active instance before it failed, is typically small in comparison to the time it takes for IBM i to detect and transfer the independent ASP to the server running the standby instance of the queue manager. Independent ASP transfer times can be as much as ten to fifteen minutes rather than being completed in seconds. The independent ASP transfer time depends on the number of objects that need to be *varied-on* when the independent ASP is transferred to the standby system and the size of the access paths, or indices, that need to be merged.

However, transferring the journal is not the only factor influencing the time it takes for the standby instance to fully resume. You also need to consider the time it takes for the network file system to release the lock on queue manager data that signals to the standby instance to try to continue with its start-up, and also the time it takes to recover queues from the journal so that the instance is able to start processing messages again. These other sources of delay all add to the time it takes to start a standby instance. The total time to switch over consists of the following components,

Failure detection time
The time it takes for NFS to release the lock on the queue manager data, and the standby instance to continue its startup process.

Transfer time
In the case of an HA cluster, the time it takes IBM i to transfer the IASP from the system hosting the active instance to the standby instance, and in the case of journal replication, the time it takes to update the local journal at the standby with the data from the remote replica.

Restart time
The time it takes for the newly active queue manager instance to rebuild its queues from the latest checkpoint in its restored journal and to resume processing messages.

Note:

If the standby instance that has taken over is configured to synchronously replicate to the previously active instance, the startup could be delayed. The new activated instance might be unable to replicate to its remote journal, if the remote journal is on the server that hosted the previously active instance, and the server has failed.

The default time to wait for a synchronous response is one minute. You can configure the maximum delay before the replication times out. Alternatively, you can configure standby instances to start using asynchronous replication to the failed active instance. Later you switch the to synchronous replication, when the failed instance is running on standby again. The same consideration applies to using synchronous independent ASP mirrors.

You can make separate baseline measurements for these components to help you assess the overall time to failover, and to factor into your decision which configuration approach to use. In making the best configuration decision you also need to consider how other applications on the same server will failover, and whether there are backup or disaster recovery processes that already use IASP.

IASP transfer times can be shortened by tuning your cluster configuration:

1. User profiles across systems in the cluster should have the same GID and UID to eliminate the need for the vary-on process to change UIDs and GIDs.
2. Minimize the number of database objects in the system and basic user disk pools, as these need to be merged to create the cross-reference table for the disk-pool group.
3. Further performance tips can be found in the IBM Redbook, *Implementing PowerHA® for IBM i, SG24-7405*.

A configuration using basic ASPs, journal mirroring, and a small configuration should switch over in the order of tens of seconds.

IBM i *Overview of combining IBM i clustering capabilities with IBM MQ clustering*

Running IBM MQ on IBM i, and exploiting the IBM i clustering capabilities can provide a more comprehensive High Availability solution, than using only IBM MQ clustering.

To have this capability, you need to set up:

1. Clusters on your IBM i machine; see “IBM i clusters” on page 414
2. An independent auxiliary storage pool (IASP), into which you move the queue manager; see “Independent auxiliary storage pools (IASPs)” on page 414
3. A cluster resource group (CRG); see “Device cluster resource groups” on page 414, in which you define the:
   - Recovery domain
   - IASP
   - Exit program; see “Device CRG exit program” on page 415

IBM i clusters

An IBM i cluster is a collection of instances, that is IBM i computers or partitions, that are logically linked together.

The purpose of this grouping is to allow for each instance to be backed up, eliminating a single point of failure and increasing application and data resiliency. With a cluster created, the various cluster resource group (CRG) types can be configured to manage applications, data, and devices in the cluster.

See Creating a cluster and the Create Cluster (CRTCLU) command for further information.

Independent auxiliary storage pools (IASPs)

An IASP is a type of user ASP that serves as an extension of single-level storage. It is a piece of storage that, due to its independence from the system storage, can be easily manipulated without having to IPL the system.

An IASP can be easily switched to another operating system instance or replicated to a target IASP on another operating system instance. Two methods can be used to switch an IASP between instances:

- The first method requires all the computers in the cluster, and the switchable disk tower containing the IASP, to be connected using a High Speed Link (HSL) loop.
- The second method requires the operating system instances to be partitions on the same IBM i computer where input/output processors (IOPs) can be switched between partitions. No special hardware is needed to be able to replicate an IASP. The replication is performed using TCP/IP over the network.

See the Configure Device ASP (CFGDEVASP) command for more information.

Device cluster resource groups

There are several types of cluster resource groups (CRGs). For more information about the different types of CRGs available, see Cluster resource group.

This topic concentrates on a device CRG. A device CRG:

- Describes and manages device resources such as independent auxiliary storage pools (IASPs).
- Defines the recovery domain of the cluster nodes
- Assigns a device, and
- Assigns the exit program that will handle cluster events.

The recovery domain denotes which cluster node will be considered as the primary node. The rest of the nodes are considered to be backups. The backup nodes are also ordered in the recovery domain, specifying which node is the first backup, the second backup, and so on, depending on how many nodes there are in the recovery domain.

In the event of a primary node failure, the exit program is run on all nodes in the recovery domain. The exit program running on the first backup can then make the necessary initializations to make this node the new primary node.

See Creating device CRGs and the Create Cluster Resource Group (CRTCRG) command for more information.

Device CRG exit program

The operating system cluster resource service calls a device CRG exit program when an event occurs in one of the nodes the recovery domain defines; for example, a failover or switchover event.

A failover event occurs when the primary node of the cluster fails and the CRGs are switched with all the resources they manage, and a switchover event occurs when a specific CRG is manually switched from the primary node to the backup node.

Either way, the exit program is in charge of initializing and starting all the programs that were running on the previous primary node, which converts the first backup node into the new primary node.

For example, with IBM MQ, the exit program should be in charge of starting the IBM MQ subsystem (QMQM), and queue managers. Queue managers should be configured to automatically start listeners and services, such as trigger monitors.

A sample exit program, AMQSCRG4, is available on IBM i.

Switchable IASP configuration

IBM MQ can be set up to take advantage of the clustering capabilities of IBM i. To do this:

1. Create an IBM i cluster between the data center systems
2. Move the queue manager to an IASP.

   "Moving, or removing, a queue manager to, or from, an independent auxiliary storage pool" on page 416 contains some sample code to help you carry out this operation.
3. You need to create a CRG defining the recovery domain, the IASP, and the exit program.

   "Configuring a device cluster resource group" on page 415 contains some sample code to help you carry out this operation.

Related concepts

"Independent ASPs and high availability" on page 434
Independent ASPs enable applications and data to be moved between servers. The flexibility of independent ASPs means they are the basis for some IBM i high availability solutions. In considering whether to use an ASP or independent ASP for the queue manager journal, you should consider other high availability configuration based on independent ASPs.

IBM i *Configuring a device cluster resource group*

An example program to set up a device Cluster resource group (CRG).

About this task

In the following example, note that:

- [PRIMARY SITE NAME] and [BACKUP SITE NAME] could be any two distinct strings of eight characters or fewer.
- [PRIMARY IP] and [BACKUP IP] are the IPs to be used for mirroring.

Procedure

1. Identify the name of the cluster.
2. Identify the CRG exit program name and library.
3. Determine the name of the primary node and backup nodes to be defined by this CRG.
4. Identify the IASP to be managed by this CRG, and make sure it has been created under the primary node.

5. Create a device description in the backup nodes by using the command:

```
CRTDEVASP DEVD([IASP NAME]) RSRCNAME([IASP NAME])
```

6. Add the takeover IP address to all the nodes by using the command:

```
ADDTCPIFC INTNETADR(' [TAKEOVER IP]') LIND([LINE DESC])
SUBNETMASK('[SUBNET MASK]') AUTOSTART(*NO)
```

7. Start the takeover IP address only in the primary node by using the command:

```
STRTCPIFC INTNETADR('[TAKEOVER IP')
```

8. Optional: If your IASP is switchable, call this command:

```
CRTCRG CLUSTER([CLUSTER NAME]) CRG( [CRG NAME]) CRGTYPE(*DEV) EXITPGM([EXIT LIB]/[EXIT
NAME])
USRPRF([EXIT PROFILE]) RCYDMN(( [PRIMARY NODE] *PRIMARY) ([BACKUP NAME] *BACKUP))
EXITPGMFMT(EXTP0200) CFGOBJ(([IAPS NAME] *DEVD *ONLINE '[TAKEOVER IP]')
```

9. Optional: If your IASP is to be mirrored, call this command:

```
CRTCRG CLUSTER([CLUSTER NAME]) CRG([CRG NAME]) CRGTYPE(*DEV) EXITPGM([EXIT LIB]/[EXIT NAME])
USRPRF([EXIT PROFILE]) RCYDMN(([PRIMARY NODE] *PRIMARY *LAST [PRIMARY SITE NAME] ('[PRIMARY
IP]'))
[BACKUP NAME] *BACKUP *LAST [BACKUP SITE NAME] ('[BACKUP IP]'))) EXITPGMFMT(EXTP0200)
CFGOBJ(([IAPS NAME] *DEVD *ONLINE '[TAKEOVER IP]'))
```

IBM i *Moving, or removing, a queue manager to, or from, an independent auxiliary storage pool*

An example program to move a queue manager to an independent auxiliary storage pool (IASP) and commands to remove a queue manager from an IASP.

About this task

In the following example, note that:

- [MANAGER NAME] is the name of your queue manager.
- [IASP NAME] is the name of your IASP.
- [MANAGER LIBRARY] is the name of your queue manager library.
- [MANAGER DIRECTORY] is the name of your queue manager directory.

Procedure

1. Identify your primary node and your backup nodes.
2. Carry out the following procedure on your primary node:
   a) Make sure your queue manager has ended.
   b) Make sure your IASP is `vary on` by using the command

```
VRYCFG CFGOBJ([IASP NAME]) CFGTYPE(*DEV) STATUS(*ON)
```

   c) Create the queue managers directory under the IASP.
   There will be a directory under root with the name of your IASP, which is:

```
QSH CMD('mkdir -p /[IASP_NAME]/QIBM/UserData/mqm/qmgrs/')
```

   d) Move the IFS objects of your manager to the queue managers directory you have just created under the IASP using the following command:

```
QSH CMD('mv /QIBM/UserData/mqm/qmgrs/[MANAGER NAME]
/[IASP NAME]/QIBM/UserData/mqm/qmgrs')
```

   e) Create a temporary save file named `MGRLIB` by using the command:

```
CRTSAVF QGPL/MGRLIB
```

f) Save your queue manager library to the MGRLIB save file, by using the following command:

```
SAVLIB LIB([MANGER LIBRARY]) DEV(*SAVF) SAVF(QGPL/MGRLIB)
```

g) Delete the queue manager library by using the following command, and ignore all the inquiry messages:

```
DLTLIB [MANAGER LIBRARY]
```

h) Restore your queue manager library to the IASP by using the following command:

```
RSTLIB SAVLIB([MANAGER LIBRARY]) DEV(*SAVF) SAVF(QGPL/MGRLIB)
RSTASPDEV([IASP NAME])
```

i) Delete the temporary save file by using the following command:

```
 DLTF FILE(QGPL/MGRLIB)
```

j) Create a symbolic link to the queue manager IFS objects under the IASP, by using the following command:

```
ADDLNK OBJ('/[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
NEWLNK('/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
```

k) Attach to the IASP by using the following command:

```
SETASPGRP [IASP NAME]
```

l) Start your queue manager by using the command:

```
STRMQM [MANAGER NAME]
```

3. Carry out the following procedure on your backup node, or nodes:

a) Create a temporary queue manager directory by using the following command:

```
QSH CMD('mkdir -p /[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
```

b) Create a symbolic link to the queue manager temporary directory by using the following command:

```
ADDLNK OBJ('/[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
NEWLNK('/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
```

c) Delete the temporary directory by using the following command:

```
QSH CMD('rm -r /[IASP NAME]')
```

d) Add the following at the end of the file /QIBM/UserData/mqm/mqs.ini:

```
QueueManager:
Name=[MANAGER NAME]
Prefix=/QIBM/UserData/mqm
Library=[MANAGER LIBRARY]
Directory=[MANAGER DIRECTORY]
```

4. To remove a queue manager from an IASP, issue the following commands:

a) VRYCFG CFGOBJ([IASP NAME]) CFGTYPE(*DEV) STATUS(*ON)

b) SETASPGRP [IASP NAME]

c) ENDMQM [MANAGER NAME]

d) DLTMQM [MANAGER NAME]

IBM i *Mirrored journal configuration for ASP on IBM i*

Configure a robust multi-instance queue manager using synchronous replication between mirrored journals.

A mirrored queue manager configuration uses journals that are created in basic or independent auxiliary storage pools (ASP).

On IBM i, queue manager data is written to journals and to a file system. Journals contain the master copy of queue manager data. Journals are shared between systems using either synchronous or asynchronous journal replication. A mix of local and remote journals are required to restart a queue manager instance. Queue manager restart reads journal records from the mix of local and remote journals on the server, and the queue manager data on the shared network file system. The data in the file system speeds up restarting the queue manager. Checkpoints are stored in the file system, marking points of synchronization between the file system and the journals. Journal records stored before the checkpoint are not required for typical queue manager restarts. However, the data in the file system might not be up to date, and journal records after the checkpoint are used to complete the queue manager restart. The data in the journals attached to the instance are kept up to date so that the restart can complete successfully.

But even the journal records might not be up to date, if the remote journal on the standby server was being asynchronously replicated, and the failure occurred before it was synchronized. In the event that you decide to restart a queue manager using a remote journal that is not synchronized, the standby queue manager instance might either reprocess messages that were deleted before the active instance failed, or not process messages that were received before the active instance failed.

Another, rare possibility, is that the file system contains the most recent checkpoint record, and an unsynchronized remote journal on the standby does not. In this case the queue manager does not restart automatically. You have a choice of waiting until the remote journal is synchronized, or cold starting the standby queue manager from the file system. Even though, in this case, the file system contains a more recent checkpoint of the queue manager data than the remote journal, it might not contain all the messages that were processed before the active instance failed. Some messages might be reprocessed, and some not processed, after a cold restart that is out of synchronization with the journals.

With a multi-instance queue manager, the file system is also used to control which instance of a queue manager is active, and which is the standby. The active instance acquires a lock to the queue manager data. The standby waits to acquire the lock, and when it does, it becomes the active instance. The lock is released by the active instance, if it ends normally. The lock is released by the file system if the file system detects the active instance has failed, or cannot access the file system. The file system must meet the requirements for detecting failure; see Requirements for shared file systems.

The architecture of multi-instance queue managers on IBM i provides automatic restart following server or queue manager failure. It also supports restoration of queue manager data following failure of the file system where the queue manager data is stored.

In Figure 24 on page 419, if ALPHA fails, you can manually restart `QM1` on BETA, using the mirrored journal. By adding the multi-instance queue manager capability to `QM1`, the standby instance of `QM1` resumes automatically on BETA if the active instance on ALPHA fails. `QM1` can also resume automatically if it is the server ALPHA that fails, not just the active instance of `QM1`. Once BETA becomes the host of the active queue manager instance, the standby instance can be started on ALPHA.

Figure 24 on page 419 shows a configuration that mirrors journals between two instances of a queue manager using NetServer to store queue manager data. You might expand the pattern to include more journals, and hence more instances. Follow the journal naming rules explained in the topic, “Queue manager journals on IBM i” on page 399. Currently the number of running instances of a queue manager is limited to two, one is active and one is in standby.

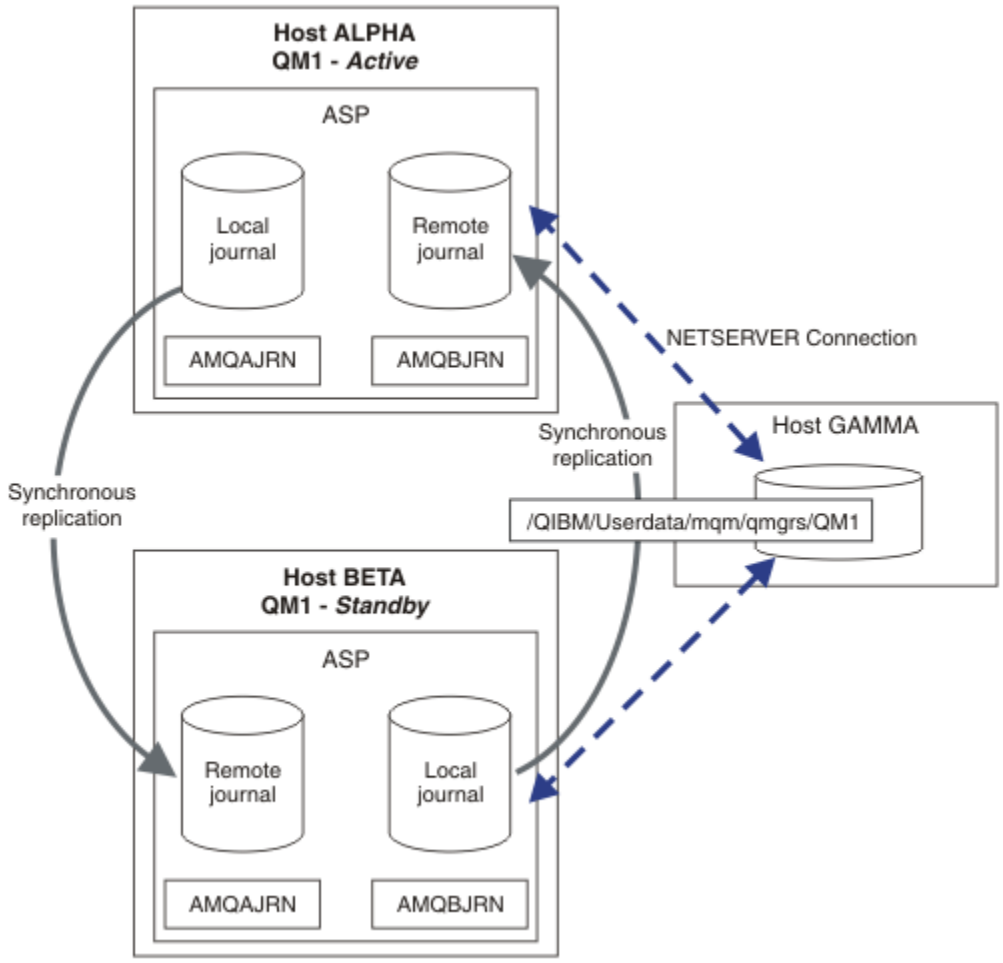


Figure 24. Mirror a queue manager journal

The local journal for QM1 on host ALPHA is called AMQAJRN (or more fully, QMQM1/AMQAJRN) and on BETA the journal is QMQM1/AMQBJRN. Each local journal replicates to remote journals on all other instances of the queue manager. If the queue manager is configured with two instances, a local journal is replicated to one remote journal.

*SYNC or *ASYNC remote journal replication

IBM i journals are mirrored using either synchronous (*SYNC) or asynchronous (*ASYNC) journaling; see Remote journal management.

The replication mode in Figure 24 on page 419 is *SYNC, not *ASYNC. *ASYNC is faster, but if a failure occurs when the remote journal state is *ASYNCPEND, the local and remote journal are not consistent. The remote journal must catch up with the local journal. If you choose *SYNC, then the local system waits for the remote journal before returning from a call that requires a completed write. The local and remote journals generally remain consistent with one another. Only if the *SYNC operation takes longer than a designated time [1] , and remote journaling is deactivated, do the journals get out of synchronization. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.

[1] The designated time is 60 seconds on IBM i 5 and in the range 1 - 3600 seconds on IBM i 6.1 onwards.

A problem with the *SYNC / *SYNC configuration illustrated in Figure 24 on page 419 is how the standby queue manager instance on BETA takes control. As soon as the queue manager instance on BETA writes its first persistent message, it attempts to update the remote journal on ALPHA. If the cause of control passing from ALPHA to BETA was the failure of ALPHA, and ALPHA is still down, remote journaling to ALPHA fails. BETA waits for ALPHA to respond, and then deactivates remote journaling and resumes processing messages with only local journaling. BETA has to wait a while to detect that ALPHA is down, causing a period of inactivity.

The choice between setting remote journaling to *SYNC or *ASYNC is a trade-off. Table 26 on page 420 summarizes the trade-offs between using *SYNC and *ASYNC journaling between a pair of queue managers:

Table 26. Remote journaling options

Active	Standby	*SYNC	*ASYNC
***SYNC**		1. Consistent switchover and failover 2. The standby instance does not resume immediately after failover. 3. Remote journaling must be available all the time 4. Queue manager performance depends on remote journaling	1. Consistent switchover and failover 2. Remote journaling must be switched to *SYNC when standby server available 3. Remote journaling must remain available after it has been restarted 4. Queue manager performance depends on remote journaling
***ASYNC**		1. Not a sensible combination	1. Some messages might be lost or duplicated after a failover or switchover 2. Standby instance need not be available all the time for the active instance to continue without delay. 3. Performance does not depend on remote journaling

***SYNC / *SYNC**

The active queue manager instance uses *SYNC journaling, and when the standby queue manager instance starts, it immediately tries to use *SYNC journaling.

1. The remote journal is transactionally consistent with the local journal of the active queue manager. If the queue manager is switched over to the standby instance, it can resume immediately. The standby instance normally resumes without any loss or duplication of messages. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
2. If the queue manager fails over to the standby instance, it might not be able to start immediately. The standby queue manager instance is activated with *SYNC journaling. The cause of the failover might prevent remote journaling to the server hosting the standby instance. The queue manager waits until the problem is detected before processing any persistent messages. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.
3. The server to which the remote journal is replicated must always be available to maintain the remote journal. The remote journal is typically replicated to the same server that hosts the standby queue manager. The server might become unavailable. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue

manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.

4. Remote journaling is slower than local journaling, and substantially slower if the servers are separated by a large distance. The queue manager must wait for remote journaling, which reduces queue manager performance.

The *SYNC / *SYNC configuration between a pair of servers has the disadvantage of a delay in resuming the standby instance after failover. The *SYNC / *ASYNC configuration does not have this problem.

*SYNC / *SYNC does guarantee no message loss after switchover or failover, as long as a remote journal is available. If you want to reduce the risk of message loss after failover or switchover you have two choices. Either stop the active instance if the remote journal becomes inactive, or create remote journals on more than one server.

***SYNC / *ASYNC**

The active queue manager instance uses *SYNC journaling, and when the standby queue manager instance starts, it uses *ASYNC journaling. Shortly after the server hosting the new standby instance becomes available, the system operator must switch the remote journal on the active instance to *SYNC. When the operator switches remote journaling from *ASYNC to *SYNC the active instance pauses if the status of the remote journal is *ASYNCPEND. The active queue manager instance waits until remaining journal entries are transferred to the remote journal. When the remote journal has synchronized with the local journal, the new standby is transactionally consistent again with the new active instance. From the perspective of the management of multi-instance queue managers, in an *SYNC / *ASYNC configuration the IBM i system operator has an additional task. The operator must switch remote journaling to *SYNC in addition to restarting the failed queue manager instance.

1. The remote journal is transactionally consistent with the local journal of the active queue manager. If the active queue manager instance is switched over, or fails over to the standby instance, the standby instance can then resume immediately. The standby instance normally resumes without any loss or duplication of messages. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
2. The system operator must switch remote journal from *ASYNC to *SYNC shortly after the system hosting the active instance becomes available again. The operator might wait for the remote journal to catch up before switching the remote journal to *SYNC. Alternatively the operator might switch the remote instance to *SYNC immediately, and force the active instance to wait until the standby instance journal has caught up. When remote journaling is set to *SYNC, the standby instance is generally transactionally consistent with the active instance. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
3. When the configuration has been restored from a switchover or failover, the server on which the remote journal is hosted must be available all the time.

Choose *SYNC / *ASYNC when you want the standby queue manager to resume quickly after a failover. You must restore the remote journal setting to *SYNC on the new active instance manually. The *SYNC / *ASYNC configuration matches the normal pattern of administering a pair of multi-instance queue managers. After one instance has failed, there is a time before the standby instance is restarted, during which the active instance cannot fail over.

***ASYNC / *ASYNC**

Both the servers hosting the active and standby queue managers are configured to use *ASYNC remote journaling.

1. When switchover or failover take place, the queue manager continues with the journal on the new server. The journal might not be synchronized when the switchover or failover takes place. Consequently messages might be lost or duplicated.
2. The active instance runs, even if the server hosting the standby queue manager is not be available. The local journal is replicated asynchronously with the standby server when it is available.

3. The performance of the local queue manager is unaffected by remote journaling.

Choose *ASYNC / *ASYNC if performance is your principal requirement, and you are prepared to loose or duplicate some messages after failover or switchover.

***ASYNC / *SYNC**
There is no reason to use this combination of options.

Queue manager activation from a remote journal

Journals are either replicated synchronously or asynchronously. The remote journal might not be active, or it might be catching up with the local journal. The remote journal might be catching up, even if it is synchronously replicated, because it might have been recently activated. The rules that the queue manager applies to the state of the remote journal it uses during start-up are as follows.

1. Standby startup fails if it must replay from the remote journal on the standby and the journal status is *FAILED or *INACTPEND.
2. When activation of the standby begins, the remote journal status on the standby must be either *ACTIVE or *INACTIVE. If the state is *INACTIVE, it is possible for activation to fail, if not all the journal data has been replicated.

   The failure occurs if the queue manager data on the network file system has a more recent checkpoint record than present in the remote journal. The failure is unlikely to happen, as long as the remote journal is activated well within the default 30 minute maximum interval between checkpoints. If the standby queue manager does read a more recent checkpoint record from the file system, it does not start.

   You have a choice: Wait until the local journal on the active server can be restored, or cold start the standby queue manager. If you choose to cold start, the queue manager starts with no journal data, and relies on the consistency and completeness of the queue manager data in the file system.

   Note: If you cold start a queue manager, you run the risk of losing or duplicating messages after the last checkpoint. The message transactions were written to the journal, but some of the transactions might not have been written to the queue manager data in the file system. When you cold start a queue manager, a fresh journal is started, and transactions not written to the queue manager data in the file system are lost.
3. The standby queue manager activation waits for the remote journal status on the standby to change from *ASYNCPEND or *SYNCPEND to *ASYNC or *SYNC. Messages are written to the job log of the execution controller periodically.

   Note: In this case activation is waiting on the remote journal local to the standby queue manager that is being activated. The queue manager also waits for a time before continuing without a remote journal. It waits when it tries to write synchronously to its remote journal (or journals) and the journal is not available.
4. Activation stops if the journal status changes to *FAILED or *INACTPEND.

The names and states of the local and remote journals to be used in the activation are written to the queue manager error log.

IBM i *Creating a multi-instance queue manager using journal mirroring and NetServer on IBM i*

Create a multi-instance queue manager to run on two IBM i servers. The queue manager data is stored on a third IBM i server using NetServer. The queue manager journal is mirrored between the two servers using remote journaling. The **ADDMQMJRN** command is used to simplify creating the remote journals.

Before you begin

1. The task requires three IBM i servers. Install IBM MQ on two of them, ALPHA and BETA in the example. the product must be at least at IBM WebSphere MQ 7.0.1 Fix Pack 1.
2. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.

3. Make sure that the QMQM user profile has the same password on all three servers.
4. Install IBM i NetServer; see i5/OS NetServer.

About this task

Perform the following steps to create the configuration shown in Figure 25 on page 425. The queue manager data is connected using IBM i NetServer.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Add Relational Database Entries (RDBE) to the IBM i systems that are going to run queue manager instances. The RDBE entries are used to connect to the IBM i systems used for remote journaling.
- Create the queue manager QM1 on the IBM i server, ALPHA.
- Add the queue manager control information for QM1 on the other IBM i server, BETA.
- Create remote journals on both the IBM i servers for both queue manager instances. Each queue manager writes to the local journal. The local journal is replicated to the remote journal. The command, **ADDMQMJRN** simplifies adding the journals and the connections.
- Start the queue manager, permitting a standby instance.

Procedure

1. Do the task, “Creating a network share for queue manager data using NetServer on IBM i” on page 410.

   As a result, ALPHA and BETA have a share, /QNTC/GAMMA/WMQ, that points to /QIBM/UserData/mqm/qmgrs on GAMMA. The user profiles QMQM and QMQMADM have the necessary permissions, and QMQM has matching passwords on all three systems.
2. Add Relational Database Entries (RDBE) to the IBM i systems that are going to host queue manager instances.
   a) On ALPHA create the connection to BETA.

   ```
   ADDRDBDIRE RDB(BETA) RMTLOCNAME(BETA *IP) RMTAUTMTH(*USRIDPWD)
   ```

   b) On BETA create the connections to ALPHA.

   ```
   ADDRDBDIRE RDB(ALPHA) RMTLOCNAME(ALPHA *IP) RMTAUTMTH(*USRIDPWD)
   ```
3. Create the queue manager QM1 on ALPHA, saving the queue manager data on GAMMA.

   ```
   CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   MQMDIRP(' /QNTC/GAMMA/WMQ ')
   ```

   The path, /QNTC/GAMMA/WMQ , uses NetServer to create the queue manager data in /QIBM/UserData/mqm/qmgrs.
4. Run **ADDMQMJRN** on ALPHA. The command adds a remote journal on BETA for QM1.

   ```
   ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(BETA)
   ```

   QM1 creates journal entries in its local journal on ALPHA when the active instance of QM1 is on ALPHA. The local journal on ALPHA is replicated to the remote journal on BETA.
5. Use the command, **DSPF**, to inspect the IBM MQ configuration data created by **CRTMQM** for QM1 on ALPHA.

   The information is needed in the next step.

In this example, the following configuration is created in `/QIBM/UserData/mqm/mqs.ini` on ALPHA for `QM1`:

```
Name=QM1
Prefix=/QIBM/UserData/mqm
Library=QMQM1
Directory=QM1
DataPath= /QNTC/GAMMA/WMQ /QM1
```

6. Create a queue manager instance of QM1 on BETA using the **ADDMQMINF** command. Run the following command on BETA to modify the queue manager control information in `/QIBM/UserData/mqm/mqs.ini` on BETA.

```
ADDMQMINF MQMNAME(QM1)
PREFIX('/QIBM/UserData/mqm')
MQMDIR(QM1)
MQMLIB(QMQM1)
DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
```

Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.

7. Run **ADDMQMJRN** on BETA. The command adds a local journal on BETA and a remote journal on ALPHA for `QM1`.

```
ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(ALPHA)
```

`QM1` creates journal entries in its local journal on BETA when the active instance of `QM1` is on BETA. The local journal on BETA is replicated to the remote journal on ALPHA.

Note: As an alternative, you might want to set up remote journaling from BETA to ALPHA using asynchronous journaling.

Use this command to set up asynchronous journaling from BETA to ALPHA, instead of the command in step “7” on page 424.

```
ADDMQMJRN MQMNAME (QM1) RMTJRNRDB (ALPHA) RMTJRNDLV (*ASYNC)
```

If the server or journaling on ALPHA is the source of the failure, BETA starts without waiting for new journal entries to be replicated to ALPHA.

Switch the replication mode to *SYNC, using the **CHGMQMJRN** command, when ALPHA is online again.

Use the information in “Mirrored journal configuration for ASP on IBM i” on page 418 to decide whether to mirror the journals synchronously, asynchronously, or a mixture of both. The default is to replicate synchronously, with a 60 second wait period for a response from the remote journal.

8. Verify that the journals on ALPHA and BETA are enabled and the status of remote journal replication is *ACTIVE.
   a) On ALPHA:

```
WRKMQMJRN MQMNAME(QM1)
```

   b) On BETA:

```
WRKMQMJRN MQMNAME(QM1)
```

9. Start the queue manager instances on ALPHA and BETA.
   a) Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.

```
STRMQM MQMNAME(QM1) STANDBY(*YES)
```

   b) Start the second instance on BETA, making it the standby instance.

```
STRMQM MQMNAME(QM1) STANDBY(*YES)
```

Results

Use **WRKMQM** to check queue manager status:

1. The status of the queue manager instance on ALPHA should be *ACTIVE.
2. The status of the queue manager instance on BETA should be *STANDBY.

Example

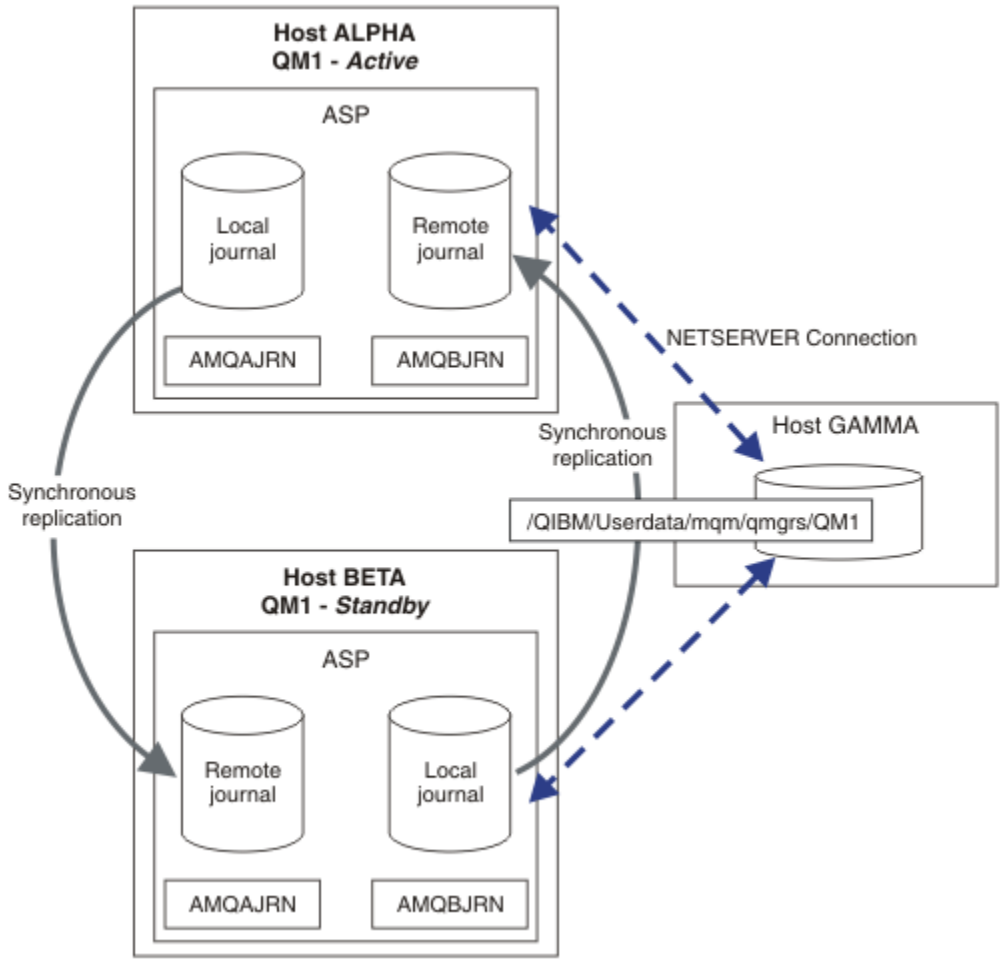


Figure 25. Mirrored journal configuration

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.
  1. Start the high availability sample programs.
  2. On ALPHA, end the queue manager requesting switch over:

     ```
     ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
     ```

  3. Check that the instance of QM1 on BETA is active.

4. Restart QM1 on ALPHA

```
STRMQM MQMNAME(QM1) STANDBY(*YES)
```

- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on a Windows server.
  2. Instead of using remote journaling to mirror the queue manager journal, store the journal on an independent ASP. Use IBM i clustering to transfer the independent ASP from ALPHA to BETA.

IBM i *Converting a single instance queue manager to a multi-instance queue manager using NetServer and journal mirroring on IBM i*

Convert a single instance queue manager to a multi-instance queue manager. Move the queue manager data to a network share connected by NetServer. Mirror the queue manager journal to a second IBM i server using remote journaling.

Before you begin

1. The task requires three IBM i servers. The existing IBM MQ installation, on the server ALPHA in the example, must be at least at IBM WebSphere MQ 7.0.1 Fix Pack 1. ALPHA is running a queue manager called QM1 in the example.
2. Install IBM MQ on the second IBM i server, BETA in the example.
3. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.
4. Make sure that the QMQM user profile has the same password on all three servers.
5. Install IBM i NetServer; see i5/OS NetServer.

About this task

Perform the following steps to convert a single instance queue manager to the multi-instance queue manager shown in Figure 26 on page 429. The single instance queue manager is deleted in the task, and then re-created, storing the queue manager data on the network share connected by NetServer. This procedure is more reliable than moving the queue manager directories and files to the network share using the **CPY** command.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Add Relational Database Entries (RDBE) to the IBM i systems that are going to run queue manager instances. The RDBE entries are used to connect to the IBM i systems used for remote journaling.
- Save the queue manager logs and definitions, stop the queue manager, and delete it.
- Re-create the queue manager, storing the queue manager data on the network share on GAMMA.
- Add the second instance of the queue manager to the other server.
- Create remote journals on both the IBM i servers for both queue manager instances. Each queue manager writes to the local journal. The local journal is replicated to the remote journal. The command, **ADDMQMJRN** simplifies adding the journals and the connections.
- Start the queue manager, permitting a standby instance.

Note:

In step “4” on page 427 of the task, you delete the single instance queue manager, QM1. Deleting the queue manager deletes all the persistent messages on queues. For this reason, complete processing all the messages stored by the queue manager, before converting the queue manager. If processing all the messages is not possible, back up the queue manager library before step “4” on page 427. Restore the queue manager library after step “5” on page 427.

Note:

In step "5" on page 427 of the task, you re-create QM1. Although the queue manager has the same name, it has a different queue manager identifier. Queue manager clustering uses the queue manager identifier. To delete and re-create a queue manager in a cluster, you must first remove the queue manager from the cluster; see Removing a queue manager from a cluster: Alternative method or Removing a queue manager from a cluster. When you have re-created the queue manager, add it to the cluster. Although it has the same name as before, it appears to be a new queue manager to the other queue managers in the cluster.

Procedure

1. Do the task, "Creating a network share for queue manager data using NetServer on IBM i" on page 410.

   As a result, ALPHA and BETA have a share, /QNTC/GAMMA/WMQ, that points to /QIBM/UserData/mqm/qmgrs on GAMMA. The user profiles QMQM and QMQMADM have the necessary permissions, and QMQM has matching passwords on all three systems.
2. Add Relational Database Entries (RDBE) to the IBM i systems that are going to host queue manager instances.
   a) On ALPHA create the connection to BETA.

   ```
   ADDRDBDIRE RDB(BETA) RMTLOCNAME(BETA *IP) RMTAUTMTH(*USRIDPWD)
   ```

   b) On BETA create the connections to ALPHA.

   ```
   ADDRDBDIRE RDB(ALPHA) RMTLOCNAME(ALPHA *IP) RMTAUTMTH(*USRIDPWD)
   ```
3. Create the scripts that re-create the queue manager objects.

   ```
   QSAVEQMGR LCLQMGRNAM(QM1) FILENAME('*CURLIB/QMQSC(QM1)')
   OUTPUT(*REPLACE) MAKEAUTH(*YES) AUTHFN('*CURLIB/QMAUT(QM1)')
   ```
4. Stop the queue manager and delete it.

   ```
   ENDMQM MQMNAME(QM1) OPTION(*IMMED) ENDCCTJOB(*YES) RCDMQMIMG(*YES) TIMEOUT(15)
   DLTMQM MQMNAME(QM1)
   ```
5. Create the queue manager QM1 on ALPHA, saving the queue manager data on GAMMA.

   ```
   CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   MQMDIRP(' /QNTC/GAMMA/WMQ ')
   ```

   The path, /QNTC/GAMMA/WMQ , uses NetServer to create the queue manager data in /QIBM/UserData/mqm/qmgrs.
6. Re-create the queue manager objects for QM1 from the saved definitions.

   ```
   STRMQMMQSC SRCMBR(QM1) SRCFILE(*CURLIB/QMQSC) MQMNAME(QM1)
   ```
7. Apply the authorizations from the saved information.
   a) Compile the saved authorization program.

   ```
   CRTCLPGM PGM(*CURLIB/QM1) SRCFILE(*CURLIB/QMAUT)
   SRCMBR(QM1) REPLACE(*YES)
   ```

   b) Run the program to apply the authorizations.

   ```
   CALL PGM(*CURLIB/QM1)
   ```

   c) Refresh the security information for QM1.

   ```
   RFRMQMAUT MQMNAME(QM1)
   ```
8. Run **ADDMQMJRN** on ALPHA. The command adds a remote journal on BETA for QM1.

```
ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(BETA)
```

QM1 creates journal entries in its local journal on ALPHA when the active instance of QM1 is on ALPHA. The local journal on ALPHA is replicated to the remote journal on BETA.

9. Use the command, **DSPF**, to inspect the IBM MQ configuration data created by **CRTMQM** for QM1 on ALPHA.

   The information is needed in the next step.

   In this example, the following configuration is created in `/QIBM/UserData/mqm/mqs.ini` on ALPHA for QM1:

   ```
   Name=QM1
   Prefix=/QIBM/UserData/mqm
   Library=QMQM1
   Directory=QM1
   DataPath= /QNTC/GAMMA/WMQ /QM1
   ```

10. Create a queue manager instance of QM1 on BETA using the **ADDMQMINF** command. Run the following command on BETA to modify the queue manager control information in `/QIBM/UserData/mqm/mqs.ini` on BETA.

    ```
    ADDMQMINF MQMNAME(QM1)
    PREFIX('/QIBM/UserData/mqm')
    MQMDIR(QM1)
    MQMLIB(QMQM1)
    DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
    ```

    Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.

11. Run **ADDMQMJRN** on BETA. The command adds a local journal on BETA and a remote journal on ALPHA for QM1.

    ```
    ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(ALPHA)
    ```

    QM1 creates journal entries in its local journal on BETA when the active instance of QM1 is on BETA. The local journal on BETA is replicated to the remote journal on ALPHA.

    Note: As an alternative, you might want to set up remote journaling from BETA to ALPHA using asynchronous journaling.

    Use this command to set up asynchronous journaling from BETA to ALPHA, instead of the command in step “7” on page 424.

    ```
    ADDMQMJRN MQMNAME (QM1) RMTJRNRDB (ALPHA) RMTJRNDLV (*ASYNC)
    ```

    If the server or journaling on ALPHA is the source of the failure, BETA starts without waiting for new journal entries to be replicated to ALPHA.

    Switch the replication mode to *SYNC, using the **CHGMQMJRN** command, when ALPHA is online again.

    Use the information in “Mirrored journal configuration for ASP on IBM i” on page 418 to decide whether to mirror the journals synchronously, asynchronously, or a mixture of both. The default is to replicate synchronously, with a 60 second wait period for a response from the remote journal.

12. Verify that the journals on ALPHA and BETA are enabled and the status of remote journal replication is *ACTIVE.

    a) On ALPHA:

    ```
    WRKMQMJRN MQMNAME(QM1)
    ```

    b) On BETA:

    ```
    WRKMQMJRN MQMNAME(QM1)
    ```

13. Start the queue manager instances on ALPHA and BETA.

a) Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.

```
STRMQM MQMNAME(QM1) STANDBY(*YES)
```

b) Start the second instance on BETA, making it the standby instance.

```
STRMQM MQMNAME(QM1) STANDBY(*YES)
```

Results

Use **WRKMQM** to check queue manager status:

1. The status of the queue manager instance on ALPHA should be *ACTIVE.
2. The status of the queue manager instance on BETA should be *STANDBY.

Example

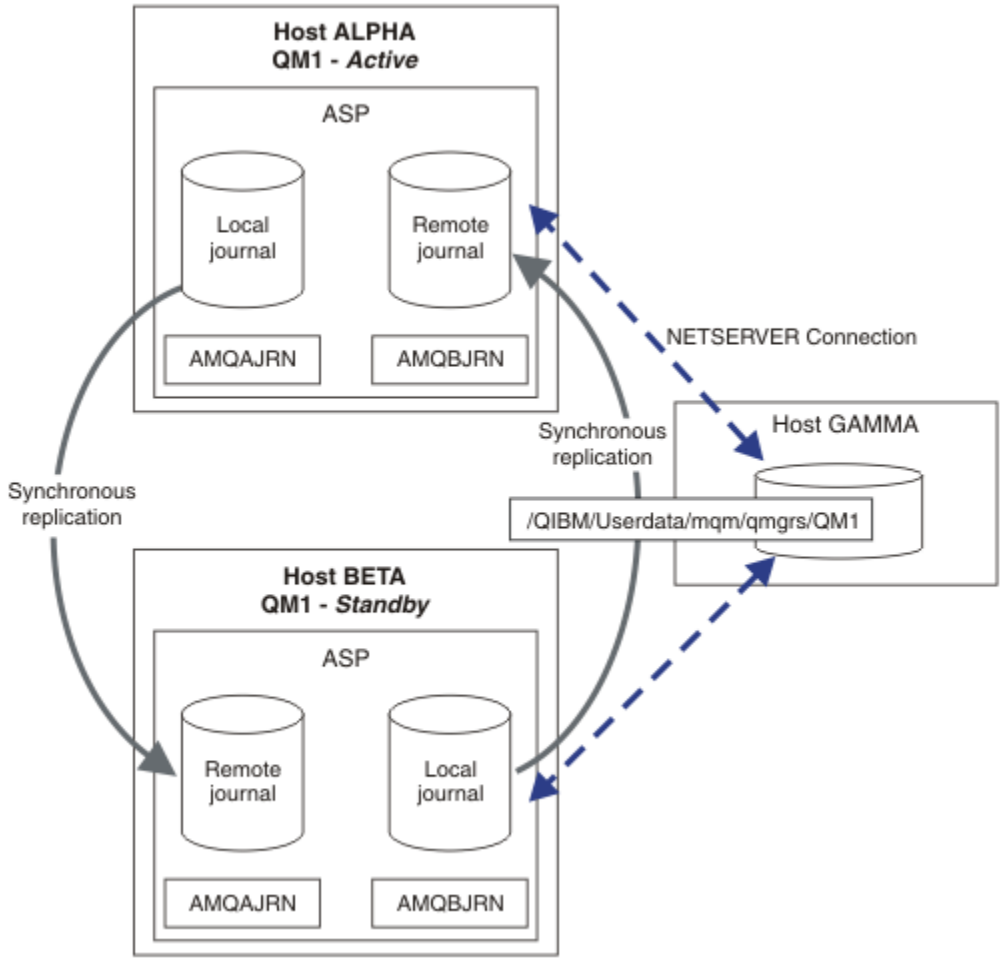


Figure 26. Mirrored journal configuration

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.

1. Start the high availability sample programs.
2. On ALPHA, end the queue manager requesting switch over:

   ```
   ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
   ```

3. Check that the instance of QM1 on BETA is active.
4. Restart QM1 on ALPHA

   ```
   STRMQM MQMNAME(QM1) STANDBY(*YES)
   ```

- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on a Windows server.
  2. Instead of using remote journaling to mirror the queue manager journal, store the journal on an independent ASP. Use IBM i clustering to transfer the independent ASP from ALPHA to BETA.

IBM i *Switched independent ASP journal configuration on IBM i*

You do not need to replicate an independent ASP journal to create a multi-instance queue manager configuration. You do need to automate a means to transfer the independent ASP from the active queue manager to the standby queue manager. There are alternative high availability solutions possible using an independent ASP, not all of which require using a multi-instance queue manager.

When using an independent ASP you do not need to mirror the queue manager journal. If you have installed cluster management, and the servers hosting the queue manager instances are in the same cluster resource group, then the queue manager journal can be transferred automatically to another server within a short distance of the active server, if the host running the active instance fails. You can also transfer the journal manually, as part of a planned switch, or you can write a command procedure to transfer the independent ASP programmatically.

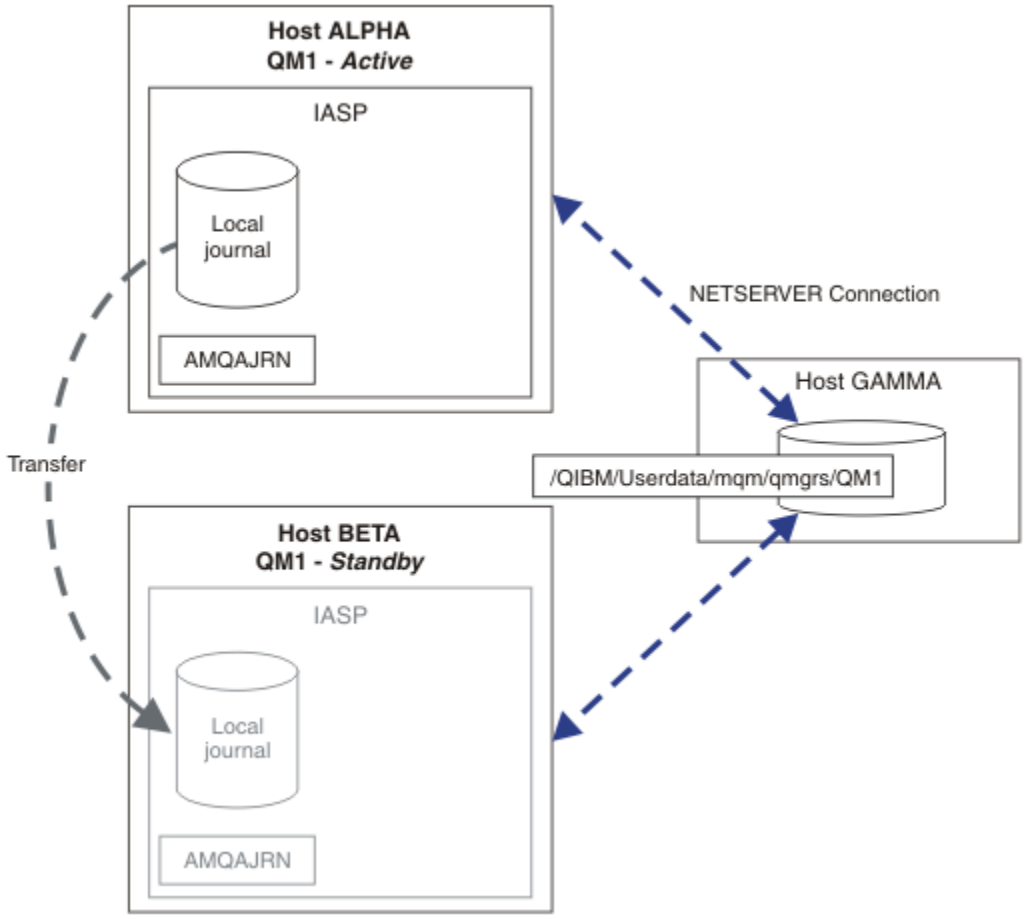


Figure 27. Transfer a queue manager journal using an independent ASP

For multi-instance queue manager operation, queue manager data must be stored on an shared file system The file system can be hosted on a variety of different platforms. You cannot store multi-instance queue manager data on an ASP or independent ASP.

The shared file system performs two roles in the configuration: The same queue manager data is shared betweem all instances of the queue manager. The file system must have a robust locking protocol that ensures only one instance of the queue manager has access to queue manager data once it has started. If the queue manager fails, or the communications to the file server breaks, then the file system must release the lock to the queue manager data held by the active instance that is no longer communicating with the file syste. The standby queue manager instance can then gain read/write access to the queue manager data. The file system protocol must conform to a set of rules to work correctly with multi-instance queue managers; see “Components of a high availability solution on IBM i” on page 409.

The locking mechanism serializes the start queue manager command and controls which instance of the queue manager is active. Once a queue manager becomes active, it rebuilds its queues from the local journal that you, or the HA cluster, has transferred to the standby server. Reconnectable clients that are waiting for reconnection to the same queue manager get reconnected, and any inflight transactions are backed out. Applications that are configured to start as queue manager services are started.

You need to ensure that the local journal from the failed active queue manager instance on the independent ASP is transferred to the server that hosts the newly activated standby queue manager instance, either by configuring the cluster resource manager, or transferring the independent ASP manually. Using independent ASPs does not preclude configuring remote journals and mirroring, if you decide to use independent ASP for backup and disaster recovery, and use remote journal mirroring for multi-instance queue manager configuration.

If you have chosen to use an independent ASP, there are alternative highly available configurations you might consider. The background to these solutions are described in "Independent ASPs and high availability" on page 434.

1. Rather than use multi-instance queue managers, install and configure a single instance queue manager entirely on an independent ASP, and use IBM i high availability services to fail the queue manager over. You would probably need to augment the solution with a queue manager monitor to detect whether the queue manager has failed independently of the server. This is the basis of the solution provided in, *Supportpac MC41: Configuring IBM MQ for iSeries for High Availability*.
2. Use independent ASPs and cross site mirroring (XSM) to mirror the independent ASP rather than switching the independent ASP on the local bus. This extends the geographic range of the independent ASP solution to as far as the time taken to write log records over a long distance allows.

IBM i *Creating a multi-instance queue manager using an independent ASP and NetServer on IBM i*

Create a multi-instance queue manager to run on two IBM i servers. The queue manager data is stored an IBM i server using NetServer. The queue manager journal is stored on an independent ASP. Use IBM i clustering or a manual procedure to transfer the independent ASP containing the queue manager journal to the other IBM i server.

Before you begin

1. The task requires three IBM i servers. Install IBM MQ on two of them, ALPHA and BETA in the example. the product must be at least at IBM WebSphere MQ 7.0.1 Fix Pack 1.
2. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.
3. Make sure that the `QMQM` user profile has the same password on all three servers.
4. Install IBM i NetServer; see i5/OS NetServer.
5. Create procedures to transfer the independent ASP from the failed queue manager to the standby that is taking over. You might find some of the techniques in *SupportPac MC41: Configuring IBM MQ for iSeries for High Availability* helpful in designing your independent ASP transfer procedures.

About this task

Perform the following steps to create the configuration shown in Figure 28 on page 434. The queue manager data is connected using IBM i NetServer.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Create the queue manager `QM1` on the IBM i server, ALPHA.
- Add the queue manager control information for `QM1` on the other IBM i server, BETA.
- Start the queue manager, permitting a standby instance.

Procedure

1. Do the task, "Creating a network share for queue manager data using NetServer on IBM i" on page 410.

   As a result, ALPHA and BETA have a share, `/QNTC/GAMMA/WMQ`, that points to `/QIBM/UserData/mqm/qmgrs` on GAMMA. The user profiles `QMQM` and `QMQMADM` have the necessary permissions, and `QMQM` has matching passwords on all three systems.
2. Create the queue manager `QM1` on ALPHA, saving the queue manager data on GAMMA.

```
CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
MQMDIRP(' /QNTC/GAMMA/WMQ ')
```

The path, `/QNTC/GAMMA/WMQ` , uses NetServer to create the queue manager data in `/QIBM/UserData/mqm/qmgrs`.

3. Use the command, **DSPF**, to inspect the IBM MQ configuration data created by **CRTMQM** for QM1 on ALPHA.

   The information is needed in the next step.

   In this example, the following configuration is created in `/QIBM/UserData/mqm/mqs.ini` on ALPHA for QM1:

   ```
   Name=QM1
   Prefix=/QIBM/UserData/mqm
   Library=QMQM1
   Directory=QM1
   DataPath= /QNTC/GAMMA/WMQ /QM1
   ```

4. Create a queue manager instance of QM1 on BETA using the **ADDMQMINF** command. Run the following command on BETA to modify the queue manager control information in `/QIBM/UserData/mqm/mqs.ini` on BETA.

   ```
   ADDMQMINF MQMNAME(QM1)
   PREFIX('/QIBM/UserData/mqm')
   MQMDIR(QM1)
   MQMLIB(QMQM1)
   DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
   ```

   Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.

5. Start the queue manager instances on ALPHA and BETA.

   a) Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.

   ```
   STRMQM MQMNAME(QM1) STANDBY(*YES)
   ```

   b) Start the second instance on BETA, making it the standby instance.

   ```
   STRMQM MQMNAME(QM1) STANDBY(*YES)
   ```

Results

Use **WRKMQM** to check queue manager status:

1. The status of the queue manager instance on ALPHA should be *ACTIVE.
2. The status of the queue manager instance on BETA should be *STANDBY.

Example

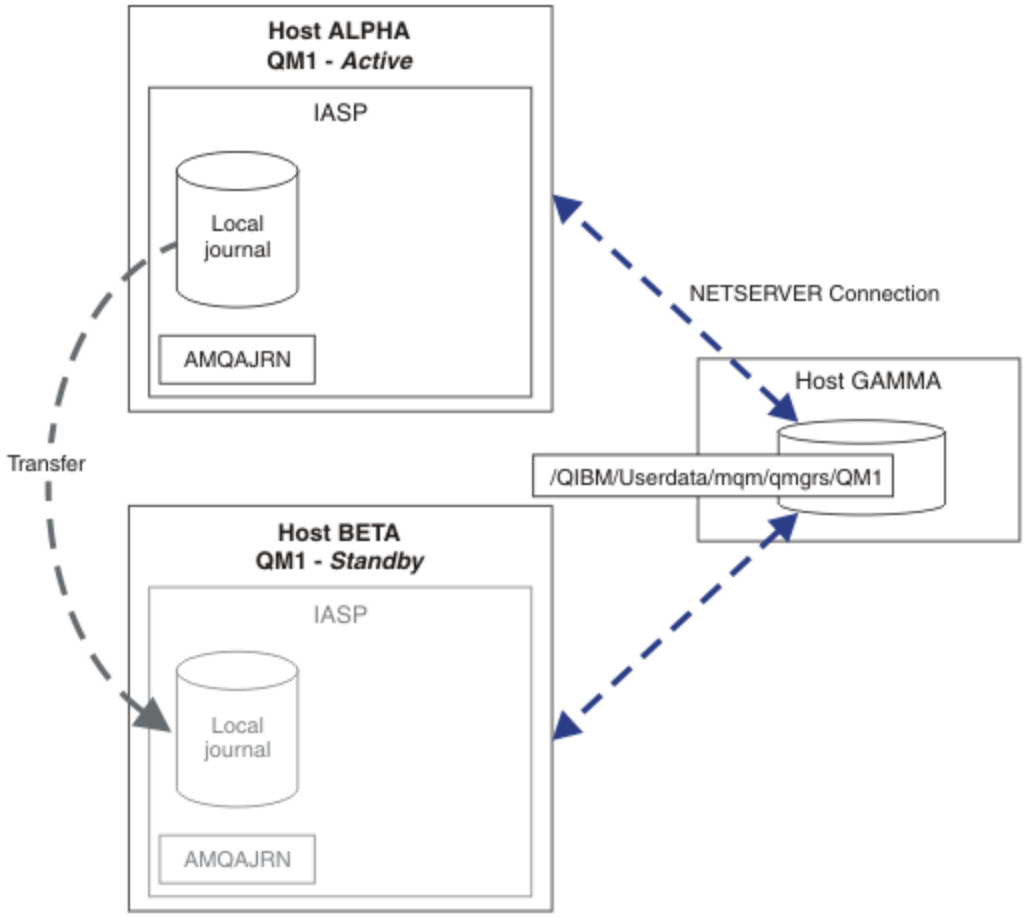


Figure 28. Transfer a queue manager journal using an independent ASP

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.
  1. Start the high availability sample programs.
  2. On ALPHA, end the queue manager requesting switch over:

     ```
     ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
     ```

  3. Check that the instance of QM1 on BETA is active.
  4. Restart QM1 on ALPHA

     ```
     STRMQM MQMNAME(QM1) STANDBY(*YES)
     ```

- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on an IBM i server.
  2. Instead of using an independent ASP to transfer the queue manager journal to the standby server, use remote journaling to mirror the journal onto the standby server.

IBM i *Independent ASPs and high availability*

Independent ASPs enable applications and data to be moved between servers. The flexibility of independent ASPs means they are the basis for some IBM i high availability solutions. In considering

whether to use an ASP or independent ASP for the queue manager journal, you should consider other high availability configuration based on independent ASPs.

Auxiliary storage pools (ASPs) are a building block of IBM i architecture. Disk units are grouped together to form a single ASP. By placing objects in different ASPs you can protect data in one ASP from being affected by disk failures in another ASP.

Every IBM i server has at least one *basic* ASP, known as the system ASP. It is designated as ASP1, and sometimes known as *SYSBAS. You can configure up to 31 additional basic *user* ASPs that are indistinguishable from the system ASP from the application's point of view, because they share the same namespace. By using multiple basic ASPs to distribute applications over many disks you can improve performance and reduce recovery time. Using multiple basic ASPs can also provide some degree of isolation against disk failure, but it does not improve reliability overall.

Independent ASPs are a special type of ASP. They are often called independent disk pools. Independent disk pools are key component of IBM i high availability. You can store data and applications that regard themselves as independent from the current system to which they are connected on independent disk storage units. You can configure switchable or non-switchable independent ASPs. From an availability perspective you are generally only concerned with switchable independent ASPs, which can be transferred automatically from server to server. As a result you can move the applications and data on the independent ASP from server to server.

Unlike basic user ASPs, independent ASPs do not share the same namespace as the system ASP. Applications that work with user ASPs require changes to work with an independent ASP. You need to verify your software, and third-party software you use, works in an independent ASP environment.

When the independent ASP is attached to a different server the namespace of the independent ASP has to be combined with the namespace of the system ASP. This process is called *varying-on* the independent ASP. You can vary-on an independent ASP without IPLing the server. Clustering support is required to transfer independent ASPs automatically from one server to another.

Building reliable solutions with independent ASPs

Journaling to an independent ASP, rather than journaling to an ASP and using journal replication, provides an alternative means to provide the standby queue manager with a copy of the local journal from the failed queue manager instance. To automatically transfer the independent ASP to another server you need to have installed and configured clustering support. There are a number of high-availability solutions for independent ASPs based on the cluster support, and low level disk mirroring, that you can combine with, or substitute for, using multi-instance queue managers.

The following list describes the components that are needed to build a reliable solution based on independent ASPs.

Journaling
Queue managers, and other applications, use local journals to write persistent data safely to disk to protect against loss of data in memory due to server failure. This is sometimes termed point-in-time consistency. It does not guarantee the consistency of multiple updates that take place over a period of time.

Commitment control
By using global transactions, you can coordinate updates to messages and databases so that the data written to the journal is consistent. It gives consistency over a period of time by using a two-phase commit protocol.

Switched disk

Switched disks are managed by the device cluster resource group (CRG) in an HA cluster. CRG switches independent ASPs automatically to a new server in the case of an unplanned outage. CRGs are geographically limited to the extent of the local IO bus.

By configuring your local journal on a switchable independent ASP, you can transfer the journal to a different server, and resume processing messages. No changes to persistent messages made without syncpoint control, or committed with syncpoint control, are lost, unless the independent ASP fails.

If you use both journaling and commitment control on switchable independent ASPs, you can transfer database journals and queue manager journals to a different server and resume processing transactions with no loss of consistency or committed transactions.

Cross-site mirroring (XSM)
XSM mirrors the primary independent ASP to a geographically remote secondary independent ASP across a TCP/IP network, and transfers control automatically in case of a failure. You have a choice of configuring a synchronous or asynchronous mirror. Synchronous mirroring reduces the performance of the queue manager because data is mirrored before the write operations on the production system complete, but it does guarantee the secondary independent ASP is up to date. Whereas if you use asynchronous mirroring you cannot guarantee that the secondary independent ASP is up to date. Asynchronous mirroring does maintain the consistency of the secondary independent ASP.

There are three XSM technologies.

Geographic mirroring
Geographic mirroring is an extension of clustering, enabling you to switch independent ASPs across a wide area. It has both synchronous and asynchronous modes. You can guarantee high availability only in synchronous mode, but the separation of independent ASPs might impact performance too much. You can combine geographic mirroring with switched disk to provide high availability locally and disaster recovery remotely.

Metro mirroring
Metro mirroring is a device level service that provides fast local synchronous mirroring over longer distances than the local bus. You can combine it with a multi-instance queue manager to give you high availability of the queue manager, and by having two copies of the independent ASP, high availability of the queue manager journal.

Global mirroring
Global mirroring is device level service that provides asynchronous mirroring, and is suitable for backing up and disaster recovery over longer distances, but is not an normal choice for high availability, because it only maintains point in time consistency rather than currency.

The key decision points you should consider are,

ASP or independent ASP?
You do not need to run a IBM i HA cluster to use multi-instance queue managers. You might choose independent ASPs, if you are already using independent ASPs, or you have availability requirements for other applications that require independent ASPs. It might be worth combining independent ASPs with multi-instance queue managers to replace queue manager monitoring as a means of detecting queue manager failure.

Availability?
What is the recovery time objective (RTO)? If you require the appearance of near uninterrupted behavior, then which solution has the quickest recovery time?

Journal availability?
How do you eliminate the journal as a single point of failure. You might adopt a hardware solution, using RAID 1 devices or better, or your might combine or use a software solution using replica journals or disk mirroring.

Distance?
How far apart are the active and standby queue manager instances. Can your users tolerate the performance degradation of replicating synchronously over distances greater than about 250 meters?

Skills?
There is work to be done to automate the administrative tasks involved in maintaining and exercising the solution regularly. The skills required to do the automation are different for the solutions based on ASPs and independent ASPs.

IBM i *Deleting a multi-instance queue manager on IBM i*

Before you delete a multi-instance queue manager, stop remote journaling, and remove queue manager instances.

Before you begin

1. In this example, two instances of the QM1 queue manager are defined on the servers ALPHA and BETA. ALPHA is the active instance and BETA is the standby. The queue manager data associated with the queue manager QM1 is stored on the IBM i server GAMMA, using NetServer. See "Creating a multi-instance queue manager using journal mirroring and NetServer on IBM i" on page 422.
2. ALPHA and BETA must be connected so that any remote journals that are defined can be deleted by IBM MQ.
3. Verify that the /QNTC directory and server directory file share can be accessed, using the system commands **EDTF** or **WRKLNK**

About this task

Before you delete a multi-instance queue manager from a server using the **DLTMQM** command, remove any queue manager instances on other servers using the **RMVMQMINF** command.

When you remove a queue manager instance using the **RMVMQMINF** command, local and remote journals prefixed with AMQ, and associated with the instance, are deleted. Configuration information about the queue manager instance, local to the server, is also deleted.

Do not run the **RMVMQMINF** command on the server holding the remaining instance of the queue manager. Doing so prevents **DLTMQM** from working correctly.

Delete the queue manager using the **DLTMQM** command. Queue manager data is removed from the network share. Local and remote journals prefixed with AMQ and associated with the instance are deleted. **DLTMQM** also deletes configuration information about the queue manager instance, local to the server.

In the example, there are only two queue manager instances. IBM MQ supports a running multi-instance configuration that has one active queue manager instance and one standby instance. If you have created additional queue manager instances to use in running configurations, remove them, using the **RMVMQMINF** command, before deleting the remaining instance.

Procedure

1. Run the **CHGMQMJRN RMTJRNSTS** (*INACTIVE) command on each server to make remote journaling between the queue manager instances inactive.
   a) On ALPHA:

   ```
   CHGMQMJRN MQMNAME('QM1')
   RMTJRNRDB('BETA') RMTJRNSTS(*INACTIVE)
   ```

   b) On BETA:

   ```
   CHGMQMJRN MQMNAME('QM1')
   RMTJRNRDB('ALPHA') RMTJRNSTS(*INACTIVE)
   ```

2. Run the **ENDMQM** command on ALPHA, the active queue manager instance, to stop both instances of QM1.

   ```
   ENDMQM MQMNAME(QM1) OPTION(*IMMED) INSTANCE(*ALL) ENDCCTJOB(*YES)
   ```

3. Run the **RMVMQMINF** command on ALPHA to remove the queue manager resources for the instance from ALPHA and BETA.

   ```
   RMVMQMINF MQMNAME(QM1)
   ```

RMVMQMINF removes the queue manager configuration information for QM1 from ALPHA. If the journal name is prefixed by AMQ, it deletes the local journal associated with QM1 from ALPHA. If the journal name is prefixed by AMQ and a remote journal has been created, it also removes the remote journal from BETA.

4. Run the **DLTMQM** command on BETA to delete QM1.

```
DLTMQM MQMNAME(QM1)
```

DLTMQM deletes the queue manager data from the network share on GAMMA. It removes the queue manager configuration information for QM1 from BETA. If the journal name is prefixed by AMQ, it deletes the local journal associated with QM1 from BETA. If the journal name is prefixed by AMQ and a remote journal has been created, it also removes the remote journal from ALPHA.

Results

DLTMQM and **RMVMQMINF** delete the local and remote journals created by **CRTMQM** and **ADDMQJRN**. The commands also delete the journal receivers. The journals and journal receivers must follow the naming convention of having names starting with AMQ. **DLTMQM** and **RMVMQMINF** remove the queue manager objects, queue manager data, and the queue manager configuration information from `mqs.ini`.

What to do next

An alternative approach is to issue the following commands after deactivating journaling in step "1" on page 437 and before ending the queue manager instances. Or, if you have not followed the naming convention, you must delete the journals and journal receivers by name.

1. On ALPHA:

```
RMVMQMJRN MQMNAME('QM1') RMTJRNRDB('BETA')
```

2. On BETA:

```
RMVMQMJRN MQMNAME('QM1') RMTJRNRDB('ALPHA')
```

After deleting the journals, continue with the rest of the steps.

IBM i *Backing up a multi-instance queue manager on IBM i*

The procedure shows you how to back up queue manager objects on the local server and the queue manager data on the network file server. Adapt the example to back up data for other queue managers.

Before you begin

In this example, the queue manager data associated with the queue manager QM1 is stored on the IBM i server called GAMMA, using NetServer. See "Creating a multi-instance queue manager using journal mirroring and NetServer on IBM i" on page 422. IBM MQ is installed on the servers, ALPHA and BETA. The queue manager, QM1, is configured on ALPHA and BETA.

About this task

IBM i does not support saving data from a remote directory. Save the queue manager data on a remote file system using the backup procedures local to the file system server. In this task, the network file system is on an IBM i server, GAMMA. The queue manager data is backed up in a save file on GAMMA.

If the network file system was on Windows or Linux, you might store the queue manager data in a compressed file, and then save it. If you have a back-up system, such as Tivoli Storage Manager, use it to back up the queue manager data.

Procedure

1. Create a save file on ALPHA for the queue manager library associated with QM1.

Use the queue manager library name to name the save file.

```
CRTSAVF FILE(QGPL/QMQM1)
```

2. Save the queue manager library in the save file on ALPHA.

```
SAVLIB LIB(QMQM1) DEV(*SAVF) SAVF(QGPL/QMQM1)
```

3. Create a save file for the queue manager data directory on GAMMA.

   Use the queue manager name to name the save file.

```
CRTSAVF FILE(QGPL/QMDQM1)
```

4. Save the copy of the queue manager data from the local directory on GAMMA.

```
SAV DEV('/QSYS.LIB/QGPL.LIB/QMDQM1.FILE') OBJ('/QIBM/Userdata/mqm/qmgrs/QM1')
```

IBM i *Commands to set up multi-instance queue managers*

IBM MQ has commands to simplify configuring journal replication, adding new queue manager instances, and configuring queue managers to use independent ASP.

The journal commands to create and manage local and remote journals are,

ADDMQMJRN
With this command you can create named local and remote journals for a queue manager instance, and configure whether replication is synchronous or asynchronous, what the synchronous timeout is, and if the remote journal is to be activated immediately.

CHGMQMJRN
The command modifies the timeout, status and delivery parameters affecting replica journals.

RMVMQMJRN
Removes named *remote* journals from a queue manager instance.

WRKMQMJRN
Lists the status of local and remote journals for a local queue manager instance.

Add and manage additional queue manager instances using the following commands, which modify the `mqs.ini` file.

ADDMQMINF
The command uses information you extracted from the `mqs.ini` file with DSPMQMINF command to add a new queue manager instance on a different IBM i server.

RMVMQMINF
Remove a queue manager instance. Use this command either to remove an instance of an existing queue manager, or to remove the configuration information for a queue manager that has been deleted from a different server.

The **CRTMQM** command has three parameters to assist configuring a multi-instance queue manager,

MQMDIRP (*DFT| *directory-prefix*)
Use this parameter to select a mount point that is mapped to queue manager data on networked storage.

ASP (*SYSTEM|*ASPDEV| *auxiliary-storage-pool-number*)
Specify *SYSTEM, or an *auxiliary-storage-pool-number* to place the queue manager journal on the system or a basic user ASP. Select the *ASPDEV option, and also set a device name using the **ASPDEV** parameter, to place the queue manager journal on an independent ASP.

ASPDEV (*ASP|*device-name*)
Specify a *device-name* of a primary or secondary independent ASP device. Selecting *ASP has the same result as specifying **ASP** (*SYSTEM).

IBM i Performance and disk failover considerations on IBM i

Use different auxiliary storage pools to improve performance and reliability.

If you use a large number of persistent messages or large messages in your applications, the time spent writing these message to disk becomes a significant factor in the performance of the system.

Ensure that you have sufficient disk activation to cope with this possibility, or consider a separate Auxiliary Storage Pool (ASP) in which to hold your queue manager journal receivers.

You can specify which ASP your queue manager library and journals are stored on when you create your queue manager using the ASP parameter of **CRTMQM**. By default, the queue manager library and journals and IFS data are stored in the system ASP.

ASPs allow isolation of objects on one or more specific disk units. This can also reduce the loss of data because of a disk media failure. In most cases, only the data that is stored on disk units in the affected ASP is lost.

You are recommended to store the queue manager library and journal data in separate user ASPs to that of the root IFS file system to provide failover and reduce disk contention.

For more information, see Backup and recovery in the IBM i documentation.

IBM i Using SAVLIB to save IBM MQ libraries on IBM i

You cannot use `SAVLIB LIB(*ALLUSR)` to save the IBM MQ libraries, because these libraries have names beginning with Q.

You can use `SAVLIB LIB(QM*)` to save all the queue manager libraries, but only if you are using a save device other than *SAVF. For `DEV(*SAVF)`, you must use a SAVLIB command for each and every queue manager library on your system.

IBM i Quiescing IBM MQ for IBM i

This section explains how to quiesce (end gracefully) IBM MQ for IBM i.

To quiesce IBM MQ for IBM i:

1. Sign on to a new interactive IBM MQ for IBM i session, ensuring that you are not accessing any objects.
2. Ensure that you have:
   - *ALLOBJ authority , or object management authority for the QMQM library
   - Sufficient authority to use the ENDSBS command
3. Advise all users that you are going to stop IBM MQ for IBM i.
4. How you then proceed depends on whether you want to shut down (quiesce) a single queue manager (where others might exist) (see “Shutting down a single queue manager for IBM MQ for IBM i” on page 441) or all the queue managers (see “Shutting down all queue managers for IBM MQ for IBM i” on page 442).
5. Shutdown the mqweb server by entering the following command in qshell:

   ```
   /QIBM/ProdData/mqm/bin/endmqweb
   ```

ENDMQM parameter ENDCCTJOB(*YES)

The ENDMQM parameter ENDCCTJOB(*YES) works differently in IBM MQ for IBM i V6.0 and later compared to previous versions.

On previous versions, when you specify ENDCCTJOB(*YES), MQ forcibly terminates your applications for you.

On IBM MQ for IBM i V6.0 or later, when you specify ENDCCTJOB(*YES), your applications are not terminated but are instead disconnected from the queue manager.

If you specify ENDCCTJOB(*YES) and you have applications that are not written to detect that a queue manager is ending, the next time a new MQI call is issued, the call will return with a MQRC_CONNECTION_BROKEN (2009) error.

As an alternative to using ENDCCTJOB(*YES), use the parameter ENDCCTJOB(*NO) and use WRKMQM option 22 (Work with jobs) to manually end any application jobs that will prevent a queue manager restart.

IBM i Shutting down a single queue manager for IBM MQ for IBM i

Use this information to understand the three types of shutdown.

In the procedures that follow, we use a sample queue manager name of QMgr1 and a sample subsystem name of SUBX. Replace these names with your own values if necessary.

Planned shutdown

Planned shutdown of a queue manager on IBM i

1. Before shutdown, execute:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```

2. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```

   If QMgr1 does not end, the channel or applications are probably busy.
3. If you must shut down QMgr1 immediately, execute the following:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Unplanned shutdown

1. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   If QMgr1 does not end, the channel or applications are probably busy.
2. If you need to shut down QMgr1 immediately, execute the following:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Shut down under abnormal conditions

1. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   If QMgr1 does not end, continue with step 3 providing that:

   - QMgr1 is in its own subsystem, or
   - You can end all queue managers that share the same subsystem as QMgr1. Use the unplanned shutdown procedure for all such queue managers.

2. When you have taken all the steps in the procedure for all the queue managers sharing the subsystem (SUBX in our examples), execute:

   ```
   ENDSBS SUBX *IMMED
   ```

   If this command fails to complete, shut down all queue managers, using the unplanned shutdown procedure, and perform an IPL on your machine.

   Warning: Do not use ENDJOBABN for IBM MQ jobs that fail to end as result of ENDJOB or ENDSBS, unless you are prepared to perform an IPL on your machine immediately after.
3. Start the subsystem by executing:

   ```
   STRSBS SUBX
   ```
4. Shut down the queue manager immediately, by executing:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(10)
   ```
5. Restart the queue manager by executing:

   ```
   STRMQM MQMNAME(QMgr1)
   ```

   If this fails, and you:

   - Have restarted your machine by performing an IPL, or
   - Have only a single queue manager

   Tidy up IBM MQ shared memory by executing:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

   before repeating step 5.

   If the queue manager restart takes more than a few seconds, IBM MQ adds status messages intermittently to the job log detailing the startup progress.

   If you still have problems restarting your queue manager, contact IBM Support. Any further action you might take could damage the queue manager, leaving IBM MQ unable to recover.

IBM i Shutting down all queue managers for IBM MQ for IBM i

Use this information to understand the three types of shutdown.

The procedures are almost the same as for a single queue manager, but using *ALL instead of the queue manager name where possible, and otherwise using a command repeatedly using each queue manager name in turn. Throughout the procedures, we use a sample queue manager name of QMgr1 and a sample subsystem name of SUBX. Replace these with your own.

Planned shutdown

1. One hour before shutdown, execute:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```

   Repeat this for every queue manager that you want to shut down.
2. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

If any queue manager does not end within a reasonable time (for example 10 minutes), proceed to step 3.

3. To shut down all queue managers immediately, execute the following:

```
ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
ENDCCTJOB(*YES) TIMEOUT(15)
```

Unplanned shutdown

1. To shut down a queue manager, execute:

```
ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
```

Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

If queue managers do not end, the channel or applications are probably busy.

2. If you need to shut down the queue managers immediately, execute the following:

```
ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
ENDCCTJOB(*YES) TIMEOUT(15)
```

Shut down under abnormal conditions

1. To shut down the queue managers, execute:

```
ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
```

Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

2. End the subsystems (SUBX in our examples), by executing:

```
ENDSBS SUBX *IMMED
```

Repeat this for every subsystem that you want to shut down; separate commands can run in parallel.

If this command fails to complete, perform an IPL on your system.

Warning: Do not use ENDJOBABN for jobs that fail to end as result of ENDJOB or ENDSBS, unless you are prepared to perform an IPL on your system immediately after.

3. Start the subsystems by executing:

```
STRSBS SUBX
```

Repeat this for every subsystem that you want to start.

4. Shut the queue managers down immediately, by executing:

```
ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
ENDCCTJOB(*YES) TIMEOUT(15)
```

5. Restart the queue managers by executing:

```
STRMQM MQMNAME(QMgr1)
```

Repeat this for every queue manager that you want to start.

If any queue manager restart takes more than a few seconds IBM MQ will show status messages intermittently detailing the startup progress.

If you still have problems restarting any queue manager, contact IBM Support. Any further action you might take could damage the queue managers, leaving MQSeries® or IBM MQ unable to recover.

z/OS Administering IBM MQ for z/OS

IBM MQ for z/OS can be controlled and managed by MQSC and PCF commands, by a set of utilities and programs provided with the product, and by authorized applications.

For details of how to administer IBM MQ for z/OS and the different administrative tasks you might have to undertake, see the following links.

You can also administer IBM MQ for z/OS using the IBM MQ Explorer running in a Linux shell. For more information, see “Administration using the IBM MQ Explorer” on page 125.

Related concepts

IBM MQ for z/OS concepts

Related tasks

“Administering IBM MQ” on page 7
To administer your IBM MQ queue managers and associated resources, choose your preferred method from a set of tasks you can use to activate and manage those resources.

Planning your IBM MQ environment on z/OS

Configuring queue managers on z/OS

z/OS Issuing queue manager commands on z/OS

You can control most of the operational environment of IBM MQ by using control commands. You can issue MQSC and PCF commands from the IBM MQ for z/OS console, the initialization input data sets, the batch utility CSQUTIL, or authorized applications.

About this task

You use MQSC commands, in batch or interactive mode, to administer queue managers directly. You use PCF commands to help you create applications that administer queue managers. MQSC commands are in human-readable text form, whereas PCF commands let applications create requests and read the replies without having to parse text strings. Like MQSC commands, applications issue PCF commands by sending them as messages to the command input queue.

The following topics describe how you issue queue manager commands from the IBM MQ for z/OS console, the initialization input data sets, the batch utility CSQUTIL, or from authorized applications.

Not all commands can be issued from all sources. See “Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS” on page 444.

Related tasks

Preparing sample applications for the TSO environment on z/OS

Related information

Administering IBM MQ using MQSC commands

z/OS Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS

You can issue MQSC and PCF commands from the IBM MQ for z/OS console, the initialization input data sets, the batch utility CSQUTIL, or from authorized applications. Not all commands can be issued from all these sources.

Which MQSC and PCF commands can control each IBM MQ object

Table 1 in "Command summary for IBM MQ for z/OS" maps which MQSC and PCF commands can be used on IBM MQ for z/OS to alter, define, delete and display each IBM MQ object. See also MQSC commands reference and “Using IBM MQ Programmable Command Formats” on page 26.

List of sources from which commands can be issued

If you are a suitably authorized user, you can issue IBM MQ commands from the following sources:

- The z/OS console or equivalent (such as SDSF/TSO).

  See also “Using the operations and control panels on z/OS” on page 458.

  Note: When using the z/OS console, you need to add `/cpf` to the start of a command, where `cpf` is the command prefix for the queue manager subsystem.
- The initialization input data sets CSQINP1, CSQINP2, CSQINPT and CSQINPX.

  See “Initialization commands for IBM MQ for z/OS” on page 455.
- The z/OS master get command routine, MGCRE (SVC 34).
- The IBM MQ batch utility programs such as CSQUTIL, which processes a list of commands in a sequential data set.

  See “Using the IBM MQ for z/OS utilities” on page 467.
- Suitably authorized applications, sending commands as messages to the SYSTEM.COMMAND.INPUT queue.

  The application can be any of the following:

  – A batch region program
  – A CICS application
  – An IMS application
  – A TSO application
  – An application program or utility on another IBM MQ system

  See “Writing programs to administer IBM MQ for z/OS” on page 475 and Preparing sample applications for the TSO environment on z/OS.

Not all commands can be issued from all sources

Commands are classified according to where they can be issued from:

1
CSQINP1

2
CSQINP2

C
The z/OS console

R
The command server and command queue, by means of, for example, CSQUTIL, CSQINPT, CSQINPX, or authorized applications.

Within the command descriptions in MQSC commands reference, these sources are identified by the use of the characters 1, 2, C, and R in each command description. Table 2 in "Command summary for IBM MQ for z/OS" summarizes the MQSC commands and the sources from which they can be issued.

Related tasks

Preparing sample applications for the TSO environment on z/OS

Related information

Administering IBM MQ using MQSC commands

z/OS Command summary for IBM MQ for z/OS

A summary of the main MQSC and PCF commands, and of the sources from which you can run MQSC commands on IBM MQ for z/OS.

Table 27 on page 446 maps which MQSC and PCF commands can be used on IBM MQ for z/OS to alter, define, delete and display each IBM MQ object.

Table 27. Summary of the main MQSC and PCF commands by object type

MQSC command	**ALTER**	**DEFINE**	**DISPLAY**	**DELETE**
PCF command	**Change**	**Create/Copy**	**Inquire**	**Delete**
AUTHINFO	X	X	X	X
CFSTATUS			X	
CFSTRUCT	X	X	X	X
CHANNEL	X	X	X	X
CHSTATUS			X	
NAMELIST	X	X	X	X
PROCESS	X	X	X	X
QALIAS	M	M	M	M
QCLUSTER			M	
QLOCAL	M	M	M	M
QMGR	X		X	
QMODEL	M	M	M	M
QREMOTE	M	M	M	M
QUEUE	P	P	X	P
QSTATUS			X	
STGCLASS	X	X	X	X

Key to table symbols:

- M = MQSC only
- P = PCF only
- X = both

There are many other MQSC and PCF commands which allow you to manage other IBM MQ resources, and carry out other actions in addition to those summarized in Table 27 on page 446.

Table 28 on page 447 shows every MQSC command, and where each command can be issued from.

- CSQINP1 initialization input data set
- CSQINP2 initialization input data set
- z/OS console (or equivalent)
- SYSTEM.COMMAND.INPUT queue and command server (from applications, CSQUTIL, or the CSQINPX initialization input data set)

Table 28. Sources from which to run MQSC commands				
Command	**CSQINP1**	**CSQINP2**	**z/OS console**	**Command input queue and server**
ALTER AUTHINFO		X	X	X
ALTER BUFFPOOL		X	X	X
ALTER CFSTRUCT		X	X	X
ALTER CHANNEL		X	X	X
ALTER NAMELIST		X	X	X
ALTER PROCESS		X	X	X
ALTER PSID			X	X
ALTER QALIAS		X	X	X
ALTER QLOCAL		X	X	X
ALTER QMGR		X	X	X
ALTER QMODEL		X	X	X
ALTER QREMOTE		X	X	X
ALTER SECURITY	X	X	X	X
ALTER SMDS		X	X	X
ALTER STGCLASS		X	X	X
ALTER SUB			X	X
ALTER TOPIC		X	X	X
ALTER TRACE	X	X	X	X
ARCHIVE LOG	X	X	X	X
BACKUP CFSTRUCT			X	X
CLEAR QLOCAL		X	X	X
CLEAR TOPICSTR			X	X
DEFINE AUTHINFO		X	X	X
DEFINE BUFFPOOL	X			
DEFINE CFSTRUCT		X	X	X
DEFINE CHANNEL		X	X	X
DEFINE LOG			X	X
DEFINE MAXSMSGS		X	X	X
DEFINE NAMELIST		X	X	X
DEFINE PROCESS		X	X	X
DEFINE PSID	X		X	X
DEFINE QALIAS		X	X	X
DEFINE QLOCAL		X	X	X

Table 28. Sources from which to run MQSC commands (continued)

Command	CSQINP1	CSQINP2	z/OS console	Command input queue and server
DEFINE QMODEL		X	X	X
DEFINE QREMOTE		X	X	X
DEFINE STGCLASS		X	X	X
DEFINE SUB			X	X
DEFINE TOPIC		X	X	X
DELETE AUTHINFO		X	X	X
DELETE BUFFPOOL		X	X	X
DELETE CFSTRUCT		X	X	X
DELETE CHANNEL			X	X
DELETE NAMELIST		X	X	X
DELETE PROCESS		X	X	X
DELETE PSID			X	X
DELETE QALIAS		X	X	X
DELETE QLOCAL		X	X	X
DELETE QMODEL		X	X	X
DELETE QREMOTE		X	X	X
DELETE STGCLASS		X	X	X
DELETE SUB			X	X
DELETE TOPIC		X	X	X
DISPLAY ARCHIVE	X	X	X	X
DISPLAY AUTHINFO		X	X	X
DISPLAY CFSTATUS			X	X
DISPLAY CFSTRUCT		X	X	X
DISPLAY CHANNEL		X	X	X
DISPLAY CHINIT			X	X
DISPLAY CHLAUTH		X	X	X
DISPLAY CHSTATUS			X	X
DISPLAY CLUSQMGR			X	X
DISPLAY CMDSERV	X	X	X	X
DISPLAY CONN		X	X	X
DISPLAY GROUP		X	X	X
DISPLAY LOG	X	X	X	X
DISPLAY MAXSMSGS		X	X	X

Table 28. Sources from which to run MQSC commands (continued)

Command	CSQINP1	CSQINP2	z/OS console	Command input queue and server
DISPLAY NAMELIST		X	X	X
DISPLAY PROCESS		X	X	X
DISPLAY PUBSUB		X	X	X
DISPLAY QALIAS		X	X	X
DISPLAY QCLUSTER		X	X	X
DISPLAY QLOCAL		X	X	X
DISPLAY QMGR		X	X	X
DISPLAY QMODEL		X	X	X
DISPLAY QREMOTE		X	X	X
DISPLAY QSTATUS		X	X	X
DISPLAY QUEUE		X	X	X
DISPLAY SBSTATUS			X	X
DISPLAY SECURITY			X	X
DISPLAY SMDS		X	X	X
DISPLAY SMDSCONN		X	X	X
DISPLAY STGCLASS		X	X	X
DISPLAY SUB			X	X
DISPLAY SYSTEM	X	X	X	X
DISPLAY TCLUSTER		X	X	X
DISPLAY THREAD		X	X	X
DISPLAY TOPIC		X	X	X
DISPLAY TPSTATUS		X	X	X
DISPLAY TRACE	X	X	X	X
DISPLAY USAGE		X	X	X
MOVE QLOCAL		X	X	X
PING CHANNEL			X	X
RECOVER BSDS			X	X
RECOVER CFSTRUCT			X	X
REFRESH CLUSTER			X	X
REFRESH QMGR		X	X	X
REFRESH SECURITY			X	X
RESET CFSTRUCT			X	X
RESET CHANNEL			X	X

Table 28. Sources from which to run MQSC commands (continued)				
Command	**CSQINP1**	**CSQINP2**	**z/OS console**	**Command input queue and server**
RESET CLUSTER			X	X
RESET QMGR		X	X	X
RESET QSTATS		X	X	X
RESET SMDS			X	X
RESET TPIPE			X	X
RESOLVE CHANNEL			X	X
RESOLVE INDOUBT		X	X	X
RESUME QMGR			X	X
RVERIFY SECURITY		X	X	X
SET ARCHIVE	X	X	X	X
SET CHLAUTH		X	X	X
SET LOG	X	X	X	X
SET SYSTEM	X	X	X	X
START CHANNEL			X	X
START CHINIT		X	X	X
START CMDSERV	X	X	X	
START LISTENER			X	X
START QMGR			X	
START SMDSCONN		X	X	X
START TRACE	X	X	X	X
STOP CHANNEL			X	X
STOP CHINIT			X	X
STOP CMDSERV	X	X	X	
STOP LISTENER			X	X
STOP QMGR			X	X
STOP SMDSCONN		X	X	X
STOP TRACE	X	X	X	X
SUSPEND QMGR			X	X

In MQSC commands, each command description identifies the sources from which that command can be run.

z/OS Using MQSC to start and stop a queue manager on z/OS

An introduction to using control commands on IBM MQ for z/OS: After you have installed IBM MQ, use MQSC commands to start and stop a queue manager.

Before you begin

After you have installed IBM MQ, it is defined as a formal z/OS subsystem. This message appears during any initial program load (IPL) of z/OS:

```
CSQ3110I +CSQ1 CSQ3UR00 - SUBSYSTEM ssnm INITIALIZATION COMPLETE
```

where *ssnm* is the IBM MQ subsystem name.

From now on, you can start the queue manager for that subsystem *from any z/OS console that has been authorized to issue system control commands* ; that is, a z/OS SYS command group. You must issue the START command from the authorized console, you cannot issue it through JES or TSO.

If you are using queue sharing groups, you must start RRS first, and then Db2®, before you start the queue manager.

About this task

When a queue manager stops under normal conditions, its last action is to take a termination checkpoint. This checkpoint, and the logs, give the queue manager the information it needs to restart.

The following steps contain information about the START and STOP commands, and contain a brief overview of start-up after an abnormal termination has occurred.

Procedure

1. Start a queue manager

   You start a queue manager by issuing a START QMGR command. However, you cannot successfully use the START command unless you have appropriate authority. See the Setting up security on z/OS for information about IBM MQ security. The following code shows examples of the START command. Note that you must prefix an MQSC command with a command prefix string (CPF).

   ```
   +CSQ1  START QMGR
   +CSQ1  START QMGR PARM(NEWLOG)
   ```

   See START QMGR for information about the syntax of the START QMGR command.

   You cannot run the queue manager as a batch job or start it using a z/OS command START. These methods are likely to start an address space for IBM MQ that then ends abnormally. Nor can you start a queue manager from the CSQUTIL utility program or a similar user application.

   You can, however, start a queue manager from an APF-authorized program by passing a START QMGR command to the z/OS MGCRE (SVC 34) service.

   If you are using queue sharing groups, the associated Db2 systems and RRS must be active when you start the queue manager.

   Start options

   When you start a queue manager, a system parameter module is loaded. You can specify the name of the system parameter module in one of two ways:

   - With the PARM parameter of the /cpf START QMGR command, for example

     ```
     /cpf START QMGR PARM(CSQ1ZPRM)
     ```

   - With a parameter in the startup procedure, for example, code the JCL EXEC statement as

```
//MQM  EXEC PGM=CSQYASCP,PARM='ZPARM(CSQ1ZPRM)'
```

A system parameter module provides information specified when the queue manager was customized.

You can use the **QMGRPROD** option to specify the product against which the queue manager usage is to be recorded, and the **AMSPROD** option to specify the equivalent for AMS if that is used. See the MQSC START QMGR command for details of the permitted values.

An example JCL EXEC statement follows:

```
//MQM EXEC  PGM=CSQYASCP,PARM='QMGRPROD(MQ)'
```

See z/OS MVS Product Management for more information on measured usage and product registration.

You can also use the ENVPARM option to substitute one or more parameters in the JCL procedure for the queue manager.

For example, you can update your queue manager startup procedure, so that the **DDname** CSQINP2 is a variable. This means that you can change the CSQINP2 **DDname** without changing the startup procedure. This is useful for implementing changes, providing back-outs for operators, and queue manager operations.

Suppose your start-up procedure for queue manager CSQ1 looked like this:

```
//CSQ1MSTR PROC INP2=NORM
//MQMESA  EXEC PGM=CSQYASCP
//STEPLIB DD  DISP=SHR,DSN=thlqual.SCSQANLE
//     DD  DISP=SHR,DSN=thlqual.SCSQAUTH
//     DD  DISP=SHR,DSN=db2qual.SDSNLOAD
//BSDS1  DD  DISP=SHR,DSN=myqual.BSDS01
//BSDS2  DD  DISP=SHR,DSN=myqual.BSDS02
//CSQP0000 DD  DISP=SHR,DSN=myqual.PSID00
//CSQP0001 DD  DISP=SHR,DSN=myqual.PSID01
//CSQP0002 DD  DISP=SHR,DSN=myqual.PSID02
//CSQP0003 DD  DISP=SHR,DSN=myqual.PSID03
//CSQINP1 DD  DISP=SHR,DSN=myqual.CSQINP(CSQ1INP1)
//CSQINP2 DD  DISP=SHR,DSN=myqual.CSQINP(CSQ1&INP2.)
//CSQOUT1 DD  SYSOUT=*
//CSQOUT2 DD  SYSOUT=*
```

If you then start your queue manager with the following command:

```
+CSQ1  START QMGR
```

then the CSQINP2 used is a member called CSQ1NORM.

However, suppose you are putting a new suite of programs into production so that the next time you start queue manager CSQ1, the CSQINP2 definitions are to be taken from member CSQ1NEW. To do this, you would start the queue manager with this command:

```
+CSQ1  START QMGR ENVPARM('INP2=NEW')
```

and CSQ1NEW would be used instead of CSQ1NORM. Note: z/OS limits the KEYWORD=value specifications for symbolic parameters (as in INP2=NEW) to 255 characters.

Starting after an abnormal termination

IBM MQ automatically detects whether restart follows a normal shutdown or an abnormal termination.

Starting a queue manager after it ends abnormally is different from starting it after the STOP QMGR command has been issued. After STOP QMGR, the system finishes its work in an orderly way and takes a termination checkpoint before stopping. When you restart the queue manager, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

However, if the queue manager ends abnormally, it terminates without being able to finish its work or take a termination checkpoint. When you restart a queue manager after an abend, it refreshes its knowledge of its status at termination using information in the log, and notifies you of the status of various tasks. Normally, the restart process resolves all inconsistent states. But, in some cases, you must take specific steps to resolve inconsistencies.

User messages on start-up

When you start a queue manager successfully, the queue manager produces a set of startup messages.

2. Stop a queue manager.

Before stopping a queue manager, all IBM MQ-related write-to-operator-with-reply (WTOR) messages must receive replies, for example, getting log requests. Each of the following commands terminates a running queue manager.

```
+CSQ1  STOP QMGR
+CSQ1  STOP QMGR MODE(QUIESCE)
+CSQ1  STOP QMGR MODE(FORCE)

+CSQ1  STOP QMGR MODE(RESTART)
```

The command STOP QMGR defaults to STOP QMGR MODE(QUIESCE).

In QUIESCE mode, IBM MQ does not allow any new connection threads to be created, but allows existing threads to continue; it terminates only when all threads have ended. Applications can request to be notified in the event of the queue manager quiescing. Therefore, use the QUIESCE mode where possible so that applications that have requested notification have the opportunity to disconnect. See What happens during termination for details.

If the queue manager does not terminate in a reasonable time in response to a STOP QMGR MODE(QUIESCE) command, use the DISPLAY CONN command to determine whether any connection threads exist, and take the necessary steps to terminate the associated applications. If there are no threads, issue a STOP QMGR MODE(FORCE) command.

The STOP QMGR MODE(QUIESCE) and STOP QMGR MODE(FORCE) commands deregister IBM MQ from the MVS Automatic Restart Manager (ARM), preventing ARM from restarting the queue manager automatically. The STOP QMGR MODE(RESTART) command works in the same way as the STOP QMGR MODE(FORCE) command, except that it does not deregister IBM MQ from ARM. This means that the queue manager is eligible for immediate automatic restart.

If the IBM MQ subsystem is not registered with ARM, the STOP QMGR MODE(RESTART) command is rejected and the following message is sent to the z/OS console:

```
CSQY205I ARM element arm-element is not registered
```

If this message is not issued, the queue manager is restarted automatically. For more information about ARM, see “Using the z/OS Automatic Restart Manager (ARM)” on page 534.

Only cancel the queue manager address space if STOP QMGR MODE(FORCE) does not terminate the queue manager.

If a queue manager is stopped by either canceling the address space or by using the command STOP QMGR MODE(FORCE), consistency is maintained with connected CICS or IMS systems. Resynchronization of resources is started when a queue manager restarts and is completed when the connection to the CICS or IMS system is established.

Note: When you stop your queue manager, you might find message IEF352I is issued. z/OS issues this message if it detects that failing to mark the address space as unusable would lead to an integrity exposure. You can ignore this message.

Stop messages

After issuing a STOP QMGR command, you get the messages CSQY009I and CSQY002I, for example:

```
CSQY009I +CSQ1 ' STOP QMGR' COMMAND ACCEPTED FROM
USER(userid), STOP MODE(FORCE)
CSQY002I +CSQ1 QUEUE MANAGER STOPPING
```

Where `userid` is the user ID that issued the STOP QMGR command, and the MODE parameter depends on that specified in the command.

When the STOP command has completed successfully, the following messages are displayed on the z/OS console:

```
CSQ9022I +CSQ1 CSQYASCP ' STOP QMGR' NORMAL COMPLETION
CSQ3104I +CSQ1 CSQ3EC0X - TERMINATION COMPLETE
```

If you are using ARM, and you did not specify MODE(RESTART), the following message is also displayed:

```
CSQY204I +CSQ1 ARM DEREGISTER for element arm-element type
arm-element-type successful
```

You cannot restart the queue manager until the following message has been displayed:

```
CSQ3100I +CSQ1 CSQ3EC0X - SUBSYSTEM ssnm READY FOR START COMMAND
```

z/OS Issuing commands from a z/OS console or equivalent

You can issue IBM MQ MQSC and PCF commands from a z/OS console or its equivalent. You can also issue IBM MQ commands from anywhere where you can issue z/OS commands, such as SDSF or by a program using the MGCRE macro. When using the z/OS console, you add `/cpf` to the start of a command.

Before you begin

Not all commands can be issued by the z/OS console. Within the command description topics (the children of MQSC commands reference), each command that can be issued by the console is identified by the character 'C'. Table 2 in "Command summary for IBM MQ for z/OS" summarizes the MQSC commands and the sources from which they can be issued.

You cannot issue IBM MQ commands using the IMS/SSR command format from an IMS terminal. This function is not supported by the IMS adapter.

The input field provided by SDSF might not be long enough for some commands, particularly those commands for channels.

The maximum amount of data that can be displayed as a result of a command typed in at the console is 32 KB.

About this task

If you are a suitably authorized user, you can issue IBM MQ commands from the z/OS console or equivalent (such as SDSF/TSO).

When using the z/OS console, you need to add `/cpf` to the start of a command, where `cpf` is the command prefix for the queue manager subsystem.

The following steps refer to commands and attributes using their MQSC command names rather than their PCF names.

Procedure

- Use command prefix strings

  Each IBM MQ command must be prefixed with a command prefix string (CPF).

  Because more than one IBM MQ subsystem can run under z/OS, the CPF is used to indicate which IBM MQ subsystem processes the command.

  For example, to start the queue manager for a subsystem called CSQ1, where CPF is ' +CSQ1 ', you issue the following command from the operator console:

  ```
  +CSQ1 START QMGR
  ```

  This CPF must be defined in the subsystem name table (for the subsystem CSQ1), as described in Defining command prefix strings (CPFs). In the examples, the string +CSQ1 is used as the command prefix.
- Use the z/OS console to issue commands

  You can type simple commands from the z/OS console, for example:

  ```
  +CSQ1 DISPLAY QUEUE(TRANSMIT.QUEUE.PROD) TYPE(QLOCAL)
  ```

  However, for complex commands or for sets of commands that you issue frequently, the other methods of issuing commands are better.
- Receive command responses

  Direct responses to commands are sent to the console that issued the command. IBM MQ supports the *Extended Console Support* (EMCS) function available in z/OS, and therefore consoles with 4 byte IDs can be used. Additionally, all commands except START QMGR and STOP QMGR support the use of Command and Response Tokens (CARTs) when the command is issued by a program using the MGCRE macro.

Related tasks

“Using the operations and control panels on z/OS” on page 458
You use these panels for defining, displaying, altering, or deleting IBM MQ objects. Use the panels for day-to-day administration and for making small changes to objects.

Preparing sample applications for the TSO environment on z/OS

z/OS Initialization commands for IBM MQ for z/OS

Initialization commands can be used to control the queue manager startup.

Commands in the initialization input data sets are processed when IBM MQ is initialized on queue manager startup. Three types of command can be issued from the initialization input data sets:

- Commands to define IBM MQ entities that cannot be defined elsewhere, for example DEFINE BUFFPOOL.

  These commands must reside in the data set identified by the DD name CSQINP1. They are processed before the restart phase of initialization. They cannot be issued through the console, operations and control panels, or an application program. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT1 statement of the started task procedure.
- Commands to define IBM MQ objects that are recoverable after restart. These definitions must be specified in the data set identified by the DD name CSQINP2. They are stored in page set zero. CSQINP2 is processed after the restart phase of initialization. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT2 statement of the started task procedure.
- Commands to manipulate IBM MQ objects. These commands must also be specified in the data set identified by the DD name CSQINP2. For example, the IBM MQ-supplied sample contains an ALTER

QMGR command to specify a dead-letter queue for the subsystem. The response to these commands is written to the CSQOUT2 output data set.

Note: If IBM MQ objects are defined in CSQINP2, IBM MQ attempts to redefine them each time the queue manager is started. If the objects already exist, the attempt to define them fails. If you need to define your objects in CSQINP2, you can avoid this problem by using the REPLACE parameter of the DEFINE commands, however, this overrides any changes that were made during the previous run of the queue manager.

Sample initialization data set members are supplied with IBM MQ for z/OS. They are described in Sample definitions supplied with IBM MQ.

Initialization commands for distributed queuing

You can also use the CSQINP2 initialization data set for the START CHINIT command. If you need a series of other commands to define your distributed queuing environment (for example, starting listeners), IBM MQ provides a third initialization input data set, called CSQINPX, that is processed as part of the channel initiator started task procedure.

The MQSC commands contained in the data set are executed at the end of channel initiator initialization, and output is written to the data set specified by the CSQOUTX DD statement. You might use the CSQINPX initialization data set to start listeners for example.

A sample channel initiator initialization data set member is supplied with IBM MQ for z/OS. It is described in Sample definitions supplied with IBM MQ.

Initialization commands for publish/Subscribe

If you need a series of commands to define your publish/subscribe environment (for example, when defining subscriptions), IBM MQ provides a fourth initialization input data set, called CSQINPT.

The MQSC commands contained in the data set are executed at the end of publish/subscribe initialization, and output is written to the data set specified by the CSQOUTT DD statement. You might use the CSQINPT initialization data set to define subscriptions for example.

A sample publish/subscribe initialization data set member is supplied with IBM MQ for z/OS. It is described in Sample definitions supplied with IBM MQ.

z/OS Private and global definitions on IBM MQ for z/OS

When you define an object on IBM MQ for z/OS, you can choose whether you want to share that definition with other queue managers (a *global* definition), or whether the object definition is to be used by one queue manager only (a *private* definition). This is called the object *disposition*.

Global definition

If your queue manager belongs to a queue sharing group, you can choose to share any object definitions you make with the other members of the group. This means that you have to define an object once only, reducing the total number of definitions required for the whole system.

Global object definitions are held in a *shared repository* (a Db2 shared database), and are available to all the queue managers in the queue sharing group. These objects have a disposition of GROUP.

Private definition

If you want to create an object definition that is required by one queue manager only, or if your queue manager is not a member of a queue sharing group, you can create object definitions that are not shared with other members of a queue sharing group.

Private object definitions are held on page set zero of the defining queue manager. These objects have a disposition of QMGR.

You can create private definitions for all types of IBM MQ objects except CF structures (that is, channels, namelists, process definitions, queues, queue managers, storage class definitions, and authentication information objects), and global definitions for all types of objects except queue managers.

IBM MQ automatically copies the definition of a group object to page set zero of each queue manager that uses it. You can alter the copy of the definition temporarily if you want, and IBM MQ allows you to refresh the page set copies from the repository copy if required.

IBM MQ always tries to refresh the page set copies from the repository copy at startup (for channel commands, this is done when the channel initiator restarts), or if the group object is changed.

Note: The copy of the definition is refreshed from the definition of the group, only if the definition of the group has changed after you created the copy of the definition.

This ensures that the page set copies reflect the version on the repository, including any changes that were made when the queue manager was inactive. The copies are refreshed by generating DEFINE REPLACE commands, therefore there are circumstances under which the refresh is not performed, for example:

- If a copy of the queue is open, a refresh that changes the usage of the queue fails.
- If a copy of a queue has messages on it, a refresh that deletes that queue fails.
- If a copy of a queue would require ALTER with FORCE to change it.

In these circumstances, the refresh is not performed on that copy, but is performed on the copies on all other queue managers.

If the queue manager is shut down and then restarted stand-alone, any local copies of objects are deleted, unless for example, the queue has associated messages.

There is a third object disposition that applies to local queues only. This allows you to create shared queues. The definition for a shared queue is held on the shared repository and is available to all the queue managers in the queue sharing group. In addition, the messages on a shared queue are also available to all the queue managers in the queue sharing group. This is described in Shared queues and queue sharing groups. Shared queues have an object disposition of SHARED.

The following table summarizes the effect of the object disposition options for queue managers started stand-alone, and as a member of a queue sharing group.

Disposition	Stand-alone queue manager	Member of a queue sharing group
QMGR	Object definition held on page set zero.	Object definition held on page set zero.
GROUP	Not allowed.	Object definition held in the shared repository. Local copy held on page set zero of each queue manager in the group.
SHARED	Not allowed.	Queue definition held in the shared repository. Messages available to any queue manager in the group.

Manipulating global definitions

If you want to change the definition of an object that is held in the shared repository, you need to specify whether you want to change the version on the repository, or the local copy on page set zero. Use the object disposition as part of the command to do this.

z/OS Directing commands to different queue managers on z/OS

You can use the *command scope* to control on which queue manager the command runs.

You can choose to execute a command on the queue manager where it is entered, or on a different queue manager in the queue sharing group. You can also choose to issue a particular command in parallel on all the queue managers in a queue sharing group. This is possible for both MQSC commands and PCF commands.

This is determined by the *command scope*. The command scope is used with the object disposition to determine which version of an object you want to work with.

For example, you might want to alter some of the attributes of an object, the definition of which is held in the shared repository.

- You might want to change the version on one queue manager only, and not make any changes to the version on the repository or those in use by other queue managers.
- You might want to change the version in the shared repository for future users, but leave existing copies unchanged.
- You might want to change the version in the shared repository, but also want your changes to be reflected immediately on all the queue managers in the queue sharing group that hold a copy of the object on their page set zero.

Use the command scope to specify whether the command is executed on this queue manager, another queue manager, or all queue managers. Use the object disposition to specify whether the object you are manipulating is in the shared repository (a group object), or is a local copy on page set zero (a queue manager object).

You do not have to specify the command scope and object disposition to work with a shared queue because every queue manager in the queue sharing group handles the shared queue as a single queue.

z/OS Using the operations and control panels on z/OS

You use these panels for defining, displaying, altering, or deleting IBM MQ objects. Use the panels for day-to-day administration and for making small changes to objects.

Before you begin

The IBM MQ for z/OS operations and controls panels (CSQOREXX) might not support all new function and parameters added at IBM WebSphere MQ for z/OS 7.0 or later. For example, there are no panels for the direct manipulation of topic objects or subscriptions. Use one of the following supported mechanisms to administer publish/subscribe definitions and other system controls that are not directly available from other panels:

1. IBM MQ Explorer
2. z/OS console
3. Programmable Command Format (PCF) messages
4. COMMAND function of CSQUTIL
5. IBM MQ Console

Note that the generic **Command** action in the CSQOREXX panels allows you to issue any valid MQSC command, including SMDS related commands. You can use all the commands that the COMMAND function of CSQUTIL issues.

You cannot issue the IBM MQ commands directly from the command line in the panels.

To use the operations and control panels, you must have the correct security authorization; this is described in the User IDs for command security and command resource security.

You cannot provide a user ID and password using CSQUTIL, or the CSQOREXX panels. Instead, if you user ID has UPDATE authority to the BATCH profile in MQCONN, you can bypass the **CHCKLOCL**(*REQUIRED* setting. See Using **CHCKLOCL** on locally bound applications for more information.

If you are setting up or changing many objects, use the COMMAND function of the CSQUTIL utility program. See “Using the CSQUTIL utility for IBM MQ for z/OS” on page 468.

About this task

The operations and control panels support the controls for the channel initiator (for example, to start a channel or a TCP/IP listener), for clustering, and for security. They also enable you to display information about threads and page set usage.

The panels work by sending MQSC type IBM MQ commands to a queue manager, through the system command input queue.

> V 10.0.0 From IBM MQ for z/OS 9.4.3, the MQSC, PCF commands, the IBM MQ Console or the IBM MQ Explorer can be used to define a TCP channel with a **CONNAME** value of length greater than 48 characters. It is recommended that channels defined with such a **CONNAME** value are not administered using the operations and controls panels since they only support a **CONNAME** value of up to 48 characters and the value will be truncated.

Example

This is the panel that displays when you start a panel session:

```
                  IBM MQ for z/OS - Main Menu

Complete fields. Then press Enter.

Action  . . . . . . . . . . 1     0. List with filter   4. Manage
                                  1. List or Display    5. Perform
                                  2. Define like        6. Start
                                  3. Alter              7. Stop
                                  8. Command
Object type . . . . . . . . CHANNEL       +
Name  . . . . . . . . . . . *
Disposition . . . . . . . . A  Q=Qmgr, C=Copy, P=Private, G=Group,
                               S=Shared, A=All

Connect name  . . . . . . . MQ1C  - local queue manager or group
Target queue manager  . . . MQ1C
         - connected or remote queue manager for command input
Action queue manager  . . . MQ1C  - command scope in group
Response wait time  . . . . 30    5 - 999 seconds

(C) Copyright IBM Corporation 1993, 2026. All rights reserved.

Command ===>
 F1=Help      F2=Split     F3=Exit      F4=Prompt    F9=SwapNext F10=Messages
F12=Cancel
```

From this panel you can perform actions such as these:

- Choose the local queue manager you want and whether you want the commands issued on that queue manager, on a remote queue manager, or on another queue manager in the same queue sharing group as the local queue manager. Over type the queue manager name if you need to change it.
- Select the action you want to perform by typing in the appropriate number in the **Action** field.
- Specify the object type that you want to work with. Press function key F1 for help about the object types if you are not sure what they are.
- Specify the disposition of the object type that you want to work with.
- Display a list of objects of the type specified. Type in an asterisk (*) in the **Name** field and press **Enter** to display a list of objects (of the type specified) that have already been defined on the action queue manager. You can then select one or more objects to work with in sequence. All the actions are available from the list.

Note: You are recommended to make choices that result in a list of objects being displayed, and then work from that list. Use the **Display** action, because that is allowed for all object types.

z/OS Invocation and rules for the operations and control panels

You can control IBM MQ and issue control commands through the ISPF panels.

How to access the IBM MQ operations and control panels

If the ISPF/PDF primary options menu has been updated for IBM MQ, you can access the IBM MQ operations and control panels from that menu. For details about updating the menu, see the Task 20: Set up the operations and control panels.

You can access the IBM MQ operations and control panels from the TSO command processor panel (typically option 6 on the ISPF/PDF primary options menu). The name of the exec that you run to do this is CSQOREXX. It has two parameters; `thlqual` is the high-level qualifier for the IBM MQ libraries to be used, and `langletter` is the letter identifying the national language libraries to be used (for example, E for U.S. English). The parameters can be omitted if the IBM MQ libraries are permanently installed in your ISPF setup. Alternatively, you can issue CSQOREXX from the TSO command line.

These panels are designed to be used by operators and administrators with a minimum of formal training. Read these instructions with the panels running and try out the different tasks suggested.

Note: While using the panels, temporary dynamic queues with names of the form SYSTEM.CSQOREXX.* are created.

Rules for the operations and control panels

See Rules for naming IBM MQ objects about the general rules for IBM MQ character strings and names. However, there are some rules that apply only to the operations and control panels:

- Do not enclose strings, for example descriptions, in single or double quotation marks.
- If you include an apostrophe or quotation mark in a text field, you do not have to repeat it or add an escape character. The characters are saved exactly as you type them; for example:

  ```
  This is Maria's queue
  ```

  The panel processor doubles them for you to pass them to IBM MQ. However, if it has to truncate your data to do this, it does so.
- You can use uppercase or lowercase characters in most fields, and they are folded to uppercase characters when you press Enter. The exceptions are:
  - Storage class names and coupling facility structure names, which must start with uppercase A through Z and be followed by uppercase A through Z or numeric characters.
  - Certain fields that are not translated. These include:
    - Application ID
    - Description
    - Environment data
    - Object names (but if you use a lowercase object name, you might not be able to enter it at a z/OS console)
    - Remote system name
    - Trigger data
    - User data
- In names, leading blanks and leading underscores are ignored. Therefore, you cannot have object names beginning with blanks or underscores.
- Underscores are used to show the extent of blank fields. When you press Enter, trailing underscores are replaced by blanks.

- Many description and text fields are presented in multiple parts, each part being handled by IBM MQ independently. This means that trailing blanks are retained and the text is not contiguous.

Blank fields

When you specify the **Define** action for an IBM MQ object, each field on the define panel contains a value. See the general help (extended help) for the display panels for information about where IBM MQ gets the values. If you type over a field with blanks, and blanks are not allowed, IBM MQ puts the installation default value in the field or prompts you to enter the required value.

When you specify the **Alter** action for an IBM MQ object, each field on the alter panel contains the current value for that field. If you type over a field with blanks, and blanks are not allowed, the value of that field is unchanged.

z/OS Objects and actions on z/OS

The operations and control panels offer you many different types of object and a number of actions that you can perform on them.

The actions are listed on the initial panel and enable you to manipulate the objects and display information about them. These objects include all the IBM MQ objects, together with some extra ones. The objects fall into the following categories.

- Queues, processes, authentication information objects, namelists, storage classes and CF structures
- Channels
- Cluster objects
- Queue manager and security
- Connections
- System

Refer to Actions for a cross-reference table of the actions which can be taken with the IBM MQ objects.

Queues, processes, authentication information objects, namelists, storage classes and CF structures

These are the basic IBM MQ objects. There can be many of each type. They can be listed, listed with filter, defined, and deleted, and have attributes that can be displayed and altered, using the LIST or DISPLAY, LIST with FILTER, DEFINE LIKE, MANAGE, and ALTER actions. (Objects are deleted using the MANAGE action.)

This category consists of the following objects:

QLOCAL	Local queue
QREMOTE	Remote queue
QALIAS	Alias queue for indirect reference to a queue
QMODEL	Model queue for defining queues dynamically
QUEUE	Any type of queue
QSTATUS	Status of a local queue
PROCESS	Information about an application to be started when a trigger event occurs
AUTHINFO	Authentication information: definitions required to perform Certificate Revocation List (CRL) checking using LDAP servers
NAMELIST	List of names, such as queues or clusters
STGCLASS	Storage class
CFSTRUCT	coupling facility (CF) structure

CFSTATUS	Status of a CF structure

Channels

Channels are used for distributed queuing. There can be many of each type, and they can be listed, listed with filter, defined, deleted, displayed, and altered. They also have other functions available using the START, STOP and PERFORM actions. PERFORM provides reset, ping, and resolve channel functions.

This category consists of the following objects:

CHANNEL	Any type of channel
SENDER	Sender channel
SERVER	Server channel
RECEIVER	Receiver channel
REQUESTER	Requester channel
CLUSRCVR	Cluster-receiver channel
CLUSSDR	Cluster-sender channel
SVRCONN	Server-connection channel
CLNTCONN	Client-connection channel
CHSTATUS	Status of a channel connection

Cluster objects

Cluster objects are created automatically for queues and channels that belong to a cluster. The base queue and channel definitions can be on another queue manager. There can be many of each type, and names can be duplicated. They can be listed, listed with filter, and displayed. PERFORM, START, and STOP are also available through the LIST actions.

This category consists of the following objects:

CLUSQ	Cluster queue, created for a queue that belongs to a cluster
CLUSCHL	Cluster channel, created for a channel that belongs to a cluster
CLUSQMGR	Cluster queue manager, the same as a cluster channel but identified by its queue manager name

Cluster channels and cluster queue managers do have the PERFORM, START and STOP actions, but only indirectly through the DISPLAY action.

Queue manager and security

Queue manager and security objects have a single instance. They can be listed, and have attributes that can be displayed and altered (using the LIST or DISPLAY, and ALTER actions), and have other functions available using the PERFORM action.

This category consists of the following objects:

MANAGER	Queue manager: the PERFORM action provides suspend and resume cluster functions
SECURITY	Security functions: the PERFORM action provides refresh and reverify functions

Connection

Connections can be listed, listed with filter and displayed.

This category consists only of the connection object, CONNECT.

System

A collection of other functions. This category consists of the following objects:

SYSTEM	System functions
CONTROL	Synonym for SYSTEM

The functions available are:

LIST or DISPLAY	Display queue sharing group, distributed queuing, page set, or data set usage information.
PERFORM	Refresh or reset clustering
START	Start the channel initiator or listeners
STOP	Stop the channel initiator or listeners

Actions

The actions that you can perform for each type of object are shown in the following table:

Table 29. Valid operations and control panel actions for IBM MQ objects

Object	Alter	Define like	Manage (1)	List or Display	List with Filter	Perform	Start	Stop
AUTHINFO	X	X	X	X	X			
CFSTATUS				X				
CFSTRUCT	X	X	X	X	X			
CHANNEL	X	X	X	X	X	X	X	X
CHSTATUS				X	X			
CLNTCONN	X	X	X	X	X			
CLUSCHL				X	X	X(2)	X(2)	X(2)
CLUSQ				X	X			
CLUSQMGR				X	X	X(2)	X(2)	X(2)
CLUSRCVR	X	X	X	X	X	X	X	X
CLUSSDR	X	X	X	X	X	X	X	X
CONNECT				X	X			
CONTROL				X		X	X	X
MANAGER	X			X		X		
NAMELIST	X	X	X	X	X			
PROCESS	X	X	X	X	X			
QALIAS	X	X	X	X	X			
QLOCAL	X	X	X	X	X			
QMODEL	X	X	X	X	X			
QREMOTE	X	X	X	X	X			
QSTATUS				X	X			

Table 29. Valid operations and control panel actions for IBM MQ objects (continued)								
Object	**Alter**	**Define like**	**Manage (1)**	**List or Display**	**List with Filter**	**Perform**	**Start**	**Stop**
QUEUE	X	X	X	X	X			
RECEIVER	X	X	X	X	X	X	X	X
REQUESTER	X	X	X	X	X	X	X	X
SECURITY	X			X		X		
SENDER	X	X	X	X	X	X	X	X
SERVER	X	X	X	X	X	X	X	X
SVRCONN	X	X	X	X	X		X	X
STGCLASS	X	X	X	X	X			
SYSTEM				X		X	X	X

Note:

1. Provides `Delete` and other functions
2. Using the `List or Display` action

z/OS Object dispositions on z/OS

You can specify the *disposition* of the object with which you need to work. The disposition signifies where the object **definition** is kept, and how the object behaves.

The disposition is significant only if you are working with any of the following object types:

- queues
- channels
- processes
- namelists
- storage classes
- authentication information objects

If you are working with other object types, the disposition is disregarded.

Permitted values are:

Q
QMGR. The object definitions are on the page set of the queue manager and are accessible only by the queue manager.

C
COPY. The object definitions are on the page set of the queue manager and are accessible only by the queue manager. They are local copies of objects defined as having a disposition of GROUP.

P
PRIVATE. The object definitions are on the page set of the queue manager and are accessible only by the queue manager. The objects have been defined as having a disposition of QMGR or COPY.

G
GROUP. The object definitions are in the shared repository, and are accessible by all queue managers in the queue sharing group.

S
SHARED. This disposition applies only to local queues. The queue definitions are in the shared repository, and are accessible by all queue managers in the queue sharing group.

A

ALL. If the action queue manager is either the target queue manager, or *, objects of **all** dispositions are included; otherwise, objects of QMGR and COPY dispositions only are included. This is the default.

z/OS Selecting a queue manager, defaults, and levels using the ISPF control panel on z/OS

You can use the CSQOREXX exec in ISPF to control your queue managers.

While you are viewing the initial panel, you are not connected to any queue manager. However, as soon as you press Enter, you are connected to the queue manager, or a queue manager in the queue sharing group named in the **Connect name** field. You can leave this field blank; this means that you are using the default queue manager for batch applications. This is defined in CSQBDEFV (see Task 19: Set up Batch, TSO, and RRS adapters for information about this).

Use the **Target queue manager** field to specify the queue manager where the actions you request are to be performed. If you leave this field blank, it defaults to the queue manager specified in the **Connect name** field. You can specify a target queue manager that is not the one you connect to. In this case, you would normally specify the name of a remote queue manager object that provides a queue manager alias definition (the name is used as the *ObjectQMgrName* when opening the command input queue). To do this, you must have suitable queues and channels set up to access the remote queue manager.

The **Action queue manager** field allows you to specify a queue manager that is in the same queue sharing group as the queue manager specified in the **Target queue manager** field to be the queue manager where the actions you request are to be performed. If you specify * in this field, the actions you request are performed on all queue managers in the queue sharing group. If you leave this field blank, it defaults to the value specified in the **Target queue manager** field. The **Action queue manager** field corresponds to using the CMDSCOPE command modifier described in The MQSC commands.

Queue manager defaults

If you leave any queue manager fields blank, or choose to connect to a queue sharing group, a secondary window opens when you press **Enter**. This window confirms the names of the queue managers you will be using. Press **Enter** to continue. When you return to the initial panel after having made some requests, you find fields completed with the actual names.

Queue manager levels

If the action queue manager is not at IBM MQ 8.0.0 or later, some fields are not displayed, and some values cannot be entered. A few objects and actions are disallowed. In such cases, a secondary window opens asking for you to confirm that you want to proceed.

z/OS Using the function keys and command line with the ISPF control panels on z/OS

To use the panels, you must use the function keys or enter the equivalent commands in the ISPF control panel command area.

- Function keys
  - Processing your actions
  - “Displaying IBM MQ user messages” on page 466
  - Canceling your actions
  - Getting help
- Using the command line

Function keys

The function keys have special settings for IBM MQ. (This means that you cannot use the ISPF default values for the function keys; if you have previously used the KEYLIST OFF ISPF command anywhere, you must type KEYLIST ON in the command area of any operations and control panel and then press Enter to enable the IBM MQ settings.)

These function key settings can be displayed on the panels, as shown in "Using the operations and control panels on z/OS" on page 458. If the settings are not shown, type PFSHOW in the command area of any operations and control panel and then press **Enter**. To remove the display of the settings, use the command PFSHOW OFF.

The function key settings in the operations and control panels conform to CUA standards. Although you can change the key setting through normal ISPF procedures (such as the **KEYLIST** utility), you are not recommended to do so.

Note: Using the **PFSHOW** and **KEYLIST** commands affects any other logical ISPF screens that you have, and their settings remain when you leave the operations and control panels.

Processing your actions

Press **Enter** to carry out the action requested on a panel. The information from the panel is sent to the queue manager for processing.

Each time you press **Enter** in the panels, IBM MQ generates one or more operator messages. If the operation was successful, you get confirmation message CSQ9022I, otherwise you get some error messages.

Displaying IBM MQ user messages

Press function key F10 in any panel to see the IBM MQ user messages.

Canceling your actions

On the initial panel, both F3 and F12 exit the operations and control panels and return you to ISPF. No information is sent to the queue manager.

On any other panel, press function keys F3 or F12 to leave the current panel **ignoring any data you have typed since last pressing Enter**. Again, no information is sent to the queue manager.

- F3 takes you straight back to the initial panel.
- F12 takes you back to the previous panel.

Getting help

Each panel has help panels associated with it. The help panels use the ISPF protocols:

- Press function key F1 on any panel to see general help (extended help) about the task.
- Press function key F1 with the cursor on any field to see specific help about that field.
- Press function key F5 from any field help panel to get the general help.
- Press function key F3 to return to the base panel, that is, the panel from which you pressed function key F1.
- Press function key F6 from any help panel to get help about the function keys.

If the help information carries on into a second or subsequent pages, a **More** indicator is displayed in the upper-right of the panel. Use these function keys to navigate through the help pages:

- F11 to get to the next help page (if there is one).
- F10 to get back to the previous help page (if there is one).

Using the command line

You never need to use the command line to issue the commands used by the operations and control panels because they are available from function keys. The command line is provided to allow you to enter normal ISPF commands (like **PFSHOW**).

The ISPF command `PANELID ON` displays the name of the current CSQOREXX panel.

The command line is initially displayed in the default position at the bottom of the panels, regardless of what ISPF settings you have. You can use the `SETTINGS ISPF` command from any of the operations and control panels to change the position of the command line. The settings are remembered for subsequent sessions with the operations and control panels.

z/OS Using the IBM MQ for z/OS utilities

IBM MQ for z/OS provides a set of utility programs that you can use to help with system administration.

IBM MQ for z/OS supplies a set of utility programs to help you perform various administrative tasks, including the following:

- Manage message security policies.
- Perform backup, restoration, and reorganization tasks.
- Issue commands and process object definitions.
- Generate data-conversion exits.
- Modify the bootstrap data set.
- List information about the logs.
- Print the logs.
- Set up Db2 tables and other Db2 utilities.
- Process messages on the dead-letter queue.

The message security policy utility

The message security policy utility (CSQ0UTIL) runs as a stand-alone utility to manage message security policies. See The message security policy utility (CSQ0UTIL) for more information.

The CSQUTIL utility

This is a utility program provided to help you with backup, restore and reorganize tasks. See “Using the CSQUTIL utility for IBM MQ for z/OS” on page 468.

The data conversion exit utility

The IBM MQ for z/OS data conversion exit utility (**CSQUCVX**) runs as a stand-alone utility to create data conversion exit routines.

The change log inventory utility

The IBM MQ for z/OS change log inventory utility program (**CSQJU003**) runs as a stand-alone utility to change the bootstrap data set (BSDS). You can use the utility to perform the following functions:

- Add or delete active or archive log data sets.
- Supply passwords for archive logs.

The print log map utility

The IBM MQ for z/OS print log map utility program (**CSQJU004**) runs as a stand-alone utility to list the following information:

- Log data set name and log RBA association for both copies of all active and archive log data sets. If dual logging is not active, there is only one copy of the data sets.
- Active log data sets available for new log data.
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS).
- Contents of the archive log command history record.
- System and utility time stamps.

The log print utility

The log print utility program (**CSQ1LOGP**) is run as a stand-alone utility. You can run the utility specifying:

- A bootstrap data set (BSDS)
- Active logs (with no BSDS)
- Archive logs (with no BSDS)

The queue sharing group utility

The queue sharing group utility program (**CSQ5PQSG**) runs as a stand-alone utility to set up Db2 tables and perform other Db2 tasks required for queue sharing groups.

The active log preformat utility

The active log preformat utility (**CSQJUFMT**) formats active log data sets before they are used by a queue manager. If the active log data sets are preformatted by the utility, log write performance is improved on the queue manager's first pass through the active logs.

The dead-letter queue handler utility

The dead-letter queue handler utility program (**CSQUDLQH**) runs as a stand-alone utility. It checks messages that are on the dead-letter queue and processes them according to a set of rules that you supply to the utility.

The queue load and unload utility

The queue load and unload utility copies or moves the contents of a queue, or its messages, to a file. The utility was originally shipped as the **QLOAD** utility in IBM MQ Supportpac MO03. From IBM MQ 8.0 it is integrated into the product as executable module **CSQUDMSG** in the SCSQLOAD library, with an alias of QLOAD for compatibility. Sample JCL is provided as member CSQ4QLOD in SCSQPROC.

The equivalent utility for Multiplatforms is called **dmpmqmsg**. For details of the available options, including the differences for z/OS, see dmpmqmsg (queue load and unload).

You can also reload messages as described in Restoring messages from a data set to a queue (LOAD) on z/OS and Restoring messages from a data set to a queue (SLOAD) on z/OS.

z/OS Using the CSQUTIL utility for IBM MQ for z/OS

The CSQUTIL utility program is provided with IBM MQ for z/OS to help you perform backup, restoration, and reorganization tasks, and to issue commands and process object definitions.

About this task

Use this utility program to invoke the following functions. For example, you can issue commands from a sequential data set using the COMMAND function of the CSQUTIL utility. This function transfers the

commands, as messages, to the *system-command input queue* and waits for the response, which is printed together with the original commands in SYSPRINT.

For more information about the CSQUTIL utility program, see IBM MQ utility program (CSQUTIL).

Procedure

- COMMAND

  Use this function to issue MQSC commands, to record object definitions, and to make client-channel definition files.
- COPY

  Use this function to read the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set, and put them into a sequential file and retain the original queue.
- COPYPAGE

  Use this function to copy whole page sets to larger page sets.
- EMPTY

  Use this function to delete the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set, retaining the definitions of the queues.
- FORMAT

  Use this function to format IBM MQ for z/OS page sets.
- Restoring messages from a data set to a queue (LOAD) on z/OS

  Use this function to restore the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set from a sequential file created by the COPY function.
- PAGEINFO

  Use this function to extract page set information from one or more page sets.
- RESETPAGE

  Use this function to copy whole page sets to other page set data sets and reset the log information in the copy.
- SCOPY

  Use this function to copy the contents of a queue to a data set while the queue manager is offline.
- SDEFS

  Use this function to produce a set of define commands for objects while the queue manager is offline.
- SLOAD

  Use this function to restore messages from the destination data set of an earlier COPY or SCOPY operation. SLOAD processes a single queue.
- SWITCH

  Use this function to switch or query the transmission queue associated with cluster-sender channels.

z/OS Using the Command Facility on z/OS

Use the editor to enter or amend MQSC commands to be passed to the queue manager.

From the primary panel, CSQOPRIA, select option **8 Command**, to start the Command Facility.

You are presented with an edit session of a sequential file, *prefix*.CSQUTIL.COMMANDS, used as input to the CSQUTIL COMMAND function; see Issuing commands to IBM MQ.

You do not need to prefix commands with the command prefix string (CPF).

You can continue MQSC commands on subsequent lines by terminating the current line with the continuation characters **+** or **-**. Alternatively, use line edit mode to provide long MQSC commands or the values of long attribute values within the command.

line edit

To use line edit, move the cursor to the appropriate line in the edit panel and use **F4** to display a single line in a scrollable panel. A single line can be up to 32 760 bytes of data.

To leave line edit:

- **F3 exit** saves changes made to the line and exits
- **F12 cancel** returns to the edit panel discarding changes made to the line.

To discard changes made in the edit session, use **F12 cancel** to terminate the edit session leaving the contents of the file unchanged. Commands are not executed.

Executing commands

When you have finished entering MQSC commands, terminate the edit session with **F3 exit** to save the contents of the file and invoke CSQUTIL to pass the commands to the queue manager. The output from command processing is held in file *prefix*.CSQUTIL.OUTPUT. An edit session opens automatically on this file so that you can view the responses. Press **F3 exit** to exit this session and return to the main menu.

z/OS Working with IBM MQ objects on z/OS

Many of the tasks described in this documentation involve manipulating IBM MQ objects. The object types are queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

- Defining simple queue objects
- Defining other types of objects
- Working with object definitions
- Working with namelists

Defining simple queue objects

To define a new object, use an existing definition as the basis for it. You can do this in one of three ways:

- By selecting an object that is a member of a list displayed as a result of options selected on the initial panel. You then enter action type 2 (**Define like**) in the action field next to the selected object. Your new object has the attributes of the selected object, except the disposition. You can then change any attributes in your new object as you require.
- On the initial panel, select the **Define like** action type, enter the type of object that you are defining in the **Object type** field, and enter the name of a specific existing object in the **Name** field. Your new object has the same attributes as the object you named in the **Name** field, except the disposition. You can then change any attributes in your new object definition as you require.
- By selecting the **Define like** action type, specifying an object type and then leaving the **Name** field blank. You can then define your new object and it has the default attributes defined for your installation. You can then change any attributes in your new object definition as you require.

Note: You do not enter the name of the object you are defining on the initial panel, but on the **Define** panel you are presented with.

The following example demonstrates how to define a local queue using an existing queue as a template.

Defining a local queue

To define a local queue object from the operations and control panels, use an existing queue definition as the basis for your new definition. There are several panels to complete. When you have completed

all the panels and you are satisfied that the attributes are correct, press Enter to send your definition to the queue manager, which then creates the actual queue.

Use the **Define like** action either on the initial panel or against an object entry in a list displayed as a result of options selected on the initial panel.

For example, starting from the initial panel, complete these fields:

Action	2 (Define like)
Object type	QLOCAL
Name	QUEUE.YOU.LIKE. This is the name of the queue that provides the attributes for your new queue.

Press Enter to display the **Define a Local Queue** panel. The queue name field is blank so that you can supply the name for the new queue. The description is that of the queue upon which you are basing this new definition. Over type this field with your own description for the new queue.

The values in the other fields are those of the queue upon which you are basing this new queue, except the disposition. You can over type these fields as you require. For example, type Y in the **Put enabled** field (if it is not already Y) if suitably authorized applications can put messages on this queue.

You get field help by moving the cursor into a field and pressing function key F1. Field help provides information about the values that can be used for each attribute.

When you have completed the first panel, press function key F8 to display the second panel.

Hints:

1. Do not press Enter at this stage, otherwise the queue will be created before you have a chance to complete the remaining fields. (If you do press Enter prematurely, do not worry; you can always alter your definition later on.)
2. Do not press function keys F3 or F12, or the data you typed will be lost.

Press function key F8 repeatedly to see and complete the remaining panels, including the trigger definition, event control, and backout reporting panels.

When your local queue definition is complete

When your definition is complete, press Enter to send the information to the queue manager for processing. The queue manager creates the queue according to the definition you have supplied. If you do not want the queue to be created, press function key F3 to exit and cancel the definition.

Defining other types of objects

To define other types of object, use an existing definition as the base for your new definition as explained in Defining a local queue.

Use the **Define like** action either on the initial panel or against an object entry in a list displayed as a result of options selected on the initial panel.

For example, starting from the initial panel, complete these fields:

Action	2 (Define like)
Object type	QALIAS, NAMELIST, PROCESS, CHANNEL, and other resource objects.
Name	Leave blank or enter the name of an existing object of the same type.

Press Enter to display the corresponding DEFINE panels. Complete the fields as required and then press Enter again to send the information to the queue manager.

Like defining a local queue, defining another type of object generally requires several panels to be completed. Defining a namelist requires some additional work, as described in “Working with namelists” on page 472.

Working with object definitions

When an object has been defined, you can specify an action in the **Action** field, to alter, display, or manage it.

In each case, you can either:

- Select the object you want to work with from a list displayed as a result of options selected on the initial panel. For example, having entered 1 in the **Action** field to display objects, Queue in the **Object type** field, and * in the **Name** field, you are presented with a list of all queues defined in the system. You can then select from this list the queue with which you need to work.
- Start from the initial panel, where you specify the object you are working with by completing the **Object type** and **Name** fields.

Altering an object definition
To alter an object definition, specify action 3 and press Enter to see the ALTER panels. These panels are very similar to the DEFINE panels. You can alter the values you want. When your changes are complete, press Enter to send the information to the queue manager.

Displaying an object definition
If you want to see the details of an object without being able to change them, specify action 1 and press Enter to see the DISPLAY panels. Again, these panels are similar to the DEFINE panels except that you cannot change any of the fields. Change the object name to display details of another object.

Deleting an object
To delete an object, specify action 4 (Manage) and the **Delete** action is one of the actions presented on the resulting menu. Select the **Delete** action.

You are asked to confirm your request. If you press function key F3 or F12, the request is canceled. If you press Enter, the request is confirmed and passed to the queue manager. The object you specified is then deleted.

Note: You cannot delete most types of channel object unless the channel initiator is started.

Working with namelists

When working with namelists, proceed as you would for other objects.

For the actions DEFINE LIKE or ALTER, press function key F11 to add names to the list or to change the names in the list. This involves working with the ISPF editor and all the normal ISPF edit commands are available. Enter each name in the namelist on a separate line.

When you use the ISPF editor in this way, the function key settings are the normal ISPF settings, and **not** those used by the other operations and control panels.

If you need to specify lowercase names in the list, specify CAPS(OFF) on the editor panel command line. When you do this, all the namelists that you edit in the future are in lowercase until you specify CAPS(ON).

When you have finished editing the namelist, press function key F3 to end the ISPF edit session. Then press Enter to send the changes to the queue manager.

Attention: If you do not press Enter at this stage but press function key F3 instead, you lose any updates that you have typed in.

z/OS Implementing the system using multiple cluster transmission queues

It makes no difference if the channel is used in a single cluster, or an overlapping cluster. When the channel is selected and started, the channel selects the transmission queue depending on the definitions.

Procedure

- If you are using the DEFCLXQ option, see “Using the automatic definition of queues and switching” on page 473.
- If you are using a staged approach, see “Changing your cluster-sender channels using a phased approach” on page 473.

z/OS Using the automatic definition of queues and switching

Use this option if you are planning on using the DEFCLXQ option. There will be a queue created for every channel, and every new channel.

Procedure

1. Review the definition of the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE and change the attributes if required.

   This queue is defined in member SCSQPROC(csq4insx).
2. Create the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE model queue.
3. Apply security policies for this model queue, and the SYSTEM.CLUSTER.TRANSMIT.** queues.

   For z/OS the channel initiator started task user ID needs:

   - Control access to CLASS(MQADMIN) for

     ```
     ssid.CONTEXT.SYSTEM.CLUSTER.TRANSMIT.channelname
     ```

   - Update access to CLASS(MQQUEUE) for

     ```
     ssid.SYSTEM.CLUSTER.TRANSMIT.channelname
     ```

z/OS Changing your cluster-sender channels using a phased approach

Use this option if you are planning on using a staged approach. This process allows you to move to the new cluster-sender channels at various times to suit the needs of your enterprise.

Before you begin

- Identify your business applications, and which channels are used.
- For the queues you use, display the clusters they are in.
- Display the channels to show the connection names, the names of the remote queue managers, and which clusters the channel supports.

About this task

- Create a transmission queue. On z/OS you might want to consider which page set you use for the queue.
- Set up security policy for the queue.
- Change any queue monitoring to include this queue name.
- Decide which channels are to use this transmission queue. The channels should have a similar name, so generic characters ' * ' in the CLCHNAME identify the channel.

- When you are ready to use the new function, alter the transmission queue to specify the name of the channels to use this transmission queue. For example CLUSTER1.TOPARIS, or CLUSTER1.* or *.TOPARIS
- Start the channels

Procedure

1. Use the `DIS CLUSQMGR(xxxx) XMITQ` command to display the cluster sender channels defined in the cluster, where *xxxx* is the name of the remote queue manager.
2. Set up the security profile for the transmission queue and give the queue access to the channel initiator.
3. Define the transmission queue to be used, and specify USAGE(XMITQ) INDXTYPE(CORRELID) SHARE and CLCHNAME(*value*)

   The channel initiator started task user ID needs the following access:

   ```
   alter class(MQADMIN) ssid.CONTEXT.SYSTEM.CLUSTER.TRANSMIT.channel
   update class(MQQUEUE ssid.SYSTEM.CLUSTER.TRANSMIT.channel
   ```

   and the user ID using the SWITCH command needs the following access:

   ```
   alter cl(MQADMIN) ssid.QUEUE.queuename
   ```

4. Stop and restart the channels.

   The channel change occurs when the channel starts using an MQSC command, or you use CSQUTIL. You can identify which channels need to be restarted using the `SWITCH CHANNEL(*)` STATUS of CSQUTIL

   If you have problems when the channel is started, stop the channel, resolve the problems, and restart the channel.

   Note that you can change the CLCHNAME attribute as often as you need to.

   The value of CLCHNAME used is the one when the channel is started, so you can change the CLCHNAME definition while the channel continues to use the definitions from the time that it started. The channel uses the new definition when it is restarted.

z/OS Undoing a change to a transmission queue on z/OS

You need to have a process to backout a change if it the results are not as you expect.

What can go wrong?

If the new transmission queue is not what you expect:

1. Check the CLCHNAME is as you expect
2. Review the job log to check if the switch process has finished. If not, wait and check the new transmission queue of the channel later.

If you are using multiple cluster transmission queues, it is important that you design the transmission queues definitions explicitly and avoid complicated overlapping configuration. In this way, you can make sure that if there are problems, you can go back to the original queues and configuration.

If you encounter problems during the move to using a different transmission queue, you must resolve any problems before you can proceed with the change.

An existing change request must complete before a new change request can be made. For example, you:

1. Define a new transmission queue with a maximum depth of one and there are 10 messages waiting to be sent.
2. Change the transmission queue to specify the channel name in the CLCHNAME parameter.
3. Stop and restart the channel. The attempt to move the messages fails and reports the problems.

4. Change the CLCHNAME parameter on the transmission queue to be blank.
5. Stop and restart the channel. The channel continues to try and complete the original request, so the channel continues to use the new transmission queue.
6. Need to resolve the problems and restart the channel so the moving of messages completes successfully.

Next time the channel is restarted it picks up any changes, so if you had set CLCHNAME to blanks, the channel will not use the specified transmission queue.

In this example, changing the CLCHNAME on the transmission queue to blanks does not necessarily mean that the channel uses the SYSTEM.CLUSTER.TRANSMIT queue, as there might be other transmission queues whose CLCHNAME parameter match the channel name. For example, a generic name, or the queue manager attribute DEFCLXQ might be set to channel, so the channel uses a dynamic queue instead of the SYSTEM.CLUSTER.TRANSMIT queue.

z/OS Writing programs to administer IBM MQ for z/OS

You can write your own application programs to administer a queue manager. Use this topic to understand the requirements for writing your own administration programs.

Start of General-use programming interface information

This set of topics contains hints and guidance to enable you to issue IBM MQ commands from an IBM MQ application program.

Note: In this topic, the MQI calls are described using C-language notation. For typical invocations of the calls in the COBOL, PL/I, and assembler languages, see Function calls.

Understanding how it all works

In outline, the procedure for issuing commands from an application program is as follows:

1. Build an IBM MQ command into a type of IBM MQ message called a *request message*. The command can be in MQSC or PCF format.
2. Send (use MQPUT) this message to a special queue called the system-command input queue. The IBM MQ command processor runs the command.
3. Retrieve (use MQGET) the results of the command as *reply messages* on the reply-to queue. These messages contain the user messages that you need to determine whether your command was successful and, if it was, what the results were.

Then it is up to your application program to process the results.

This set of topics contains:

z/OS Preparing queues for administration programs

Administration programs require a number of predefined queues for system command input and receiving responses.

This section applies to commands in the MQSC format. For the equivalent in PCF, see “Using IBM MQ Programmable Command Formats” on page 26.

Before you can issue any MQPUT or MQGET calls, you must first define, and then open, the queues you are going to use.

Defining the system-command input queue

The system-command input queue is a local queue called SYSTEM.COMMAND.INPUT. The supplied CSQINP2 initialization data set, thlqual.SCSQPROC(CSQ4INSG), contains a default definition for the system-command input queue. For compatibility with IBM MQ on other platforms, an alias of this

queue, called SYSTEM.ADMIN.COMMAND.QUEUE is also supplied. See Sample definitions supplied with IBM MQ for more information.

Defining a reply-to queue

You must define a reply-to queue to receive reply messages from the IBM MQ command processor. It can be any queue with attributes that allow reply messages to be put on it. However, for normal operation, specify these attributes:

- USAGE(NORMAL)
- NOTRIGGER (unless your application uses triggering)

Avoid using persistent messages for commands, but if you choose to do so, the reply-to queue must not be a temporary dynamic queue.

The supplied CSQINP2 initialization data set, thlqual.SCSQPROC(CSQ4INSG), contains a definition for a model queue called SYSTEM.COMMAND.REPLY.MODEL. You can use this model to create a dynamic reply-to queue.

Note: Replies generated by the command processor can be up to 15 000 bytes in length.

If you use a permanent dynamic queue as a reply-to queue, your application should allow time for all PUT and GET operations to complete before attempting to delete the queue, otherwise MQRC2055 (MQRC_Q_NOT_EMPTY) can be returned. If this occurs, try the queue deletion again after a few seconds.

Opening the system-command input queue

Before you can open the system-command input queue, your application program must be connected to your queue manager. Use the MQI call MQCONN or MQCONNX to do this.

Then use the MQI call MQOPEN to open the system-command input queue. To use this call:

1. Set the **Options** parameter to MQOO_OUTPUT
2. Set the MQOD object descriptor fields as follows:

   ObjectType
   MQOT_Q (the object is a queue)

   ObjectName
   SYSTEM.COMMAND.INPUT

   ObjectQMgrName
   If you want to send your request messages to your local queue manager, leave this field blank. This means that your commands are processed locally.

   If you want your IBM MQ commands to be processed on a remote queue manager, put its name here. You must also have the correct queues and links set up, as described in Distributed queuing and clusters.

Opening a reply-to queue

To retrieve the replies from an IBM MQ command, you must open a reply-to queue. One way of doing this is to specify the model queue, SYSTEM.COMMAND.REPLY.MODEL in an MQOPEN call, to create a permanent dynamic queue as the reply-to queue. To use this call:

1. Set the **Options** parameter to MQOO_INPUT_SHARED
2. Set the MQOD object descriptor fields as follows:

   ObjectType
   MQOT_Q (the object is a queue)

ObjectName
The name of the reply-to queue. If the queue name you specify is the name of a model queue object, the queue manager creates a dynamic queue.

ObjectQMgrName
To receive replies on your local queue manager, leave this field blank.

DynamicQName
Specify the name of the dynamic queue to be created.

z/OS Using the command server

The command server is an IBM MQ component that works with the command processor component. You can send formatted messages to the command server which interprets the messages, runs the administration requests, and sends responses back to your administration application.

The command server reads request messages from the system-command input queue, verifies them, and passes the valid ones as commands to the command processor. The command processor processes the commands and puts any replies as reply messages on to the reply-to queue that you specify. The first reply message contains the user message CSQN205I. See "Interpreting the reply messages from the command server" on page 481 for more information. The command server also processes channel initiator and queue sharing group commands, wherever they are issued from.

Identifying the queue manager that processes your commands

The queue manager that processes the commands you issue from an administration program is the queue manager that owns the system-command input queue that the message is put onto.

Starting the command server

Normally, the command server is started automatically when the queue manager is started. It becomes available as soon as the message CSQ9022I 'START QMGR' NORMAL COMPLETION is returned from the START QMGR command. The command server is stopped when all the connected tasks have been disconnected during the system termination phase.

You can control the command server yourself using the START CMDSERV and STOP CMDSERV commands. To prevent the command server starting automatically when IBM MQ is restarted, you can add a STOP CMDSERV command to your CSQINP1 or CSQINP2 initialization data sets. However, this is not recommended as it prevents any channel initiator or queue sharing group commands being processed.

The STOP CMDSERV command stops the command server as soon as it has finished processing the current message, or immediately if no messages are being processed.

If the command server has been stopped by a STOP CMDSERV command in the program, no other commands from the program can be processed. To restart the command server, you must issue a START CMDSERV command from the z/OS console.

If you stop and restart the command server while the queue manager is running, all the messages that are on the system-command input queue when the command server stops are processed when the command server is restarted. However, if you stop and restart the queue manager after the command server is stopped, only the persistent messages on the system-command input queue are processed when the command server is restarted. All nonpersistent messages on the system-command input queue are lost.

Sending commands to the command server

For each command, you build a message containing the command, then put it onto the system-command input queue.

Building a message that includes IBM MQ commands

You can incorporate IBM MQ commands in an application program by building request messages that include the required commands. For each such command you:

1. Create a buffer containing a character string representing the command.
2. Issue an MQPUT call specifying the buffer name in the **buffer** parameter of the call.

The simplest way to do this in C is to define a buffer using 'char'. For example:

```
char message_buffer[ ] = "ALTER QLOCAL(SALES) PUT(ENABLED)";
```

When you build a command, use a null-terminated character string. Do not specify a command prefix string (CPF) at the start of a command defined in this way. This means that you do not have to alter your command scripts if you want to run them on another queue manager. However, you must take into account that a CPF is included in any response messages that are put onto the reply-to queue.

The command server folds all lowercase characters to uppercase unless they are inside quotation marks.

Commands can be any length up to a maximum 32 762 characters.

Putting messages on the system-command input queue

Use the MQPUT call to put request messages containing commands on the system-command input queue. In this call you specify the name of the reply-to queue that you have already opened.

To use the MQPUT call:

1. Set these MQPUT parameters:

   Hconn
   The connection handle returned by the MQCONN or MQCONNX call.

   Hobj
   The object handle returned by the MQOPEN call for the system-command input queue.

   BufferLength
   The length of the formatted command.

   Buffer
   The name of the buffer containing the command.

2. Set these MQMD fields:

   MsgType
   MQMT_REQUEST

   Format
   MQFMT_STRING or MQFMT_NONE

   If you are not using the same code page as the queue manager, set *CodedCharSetId* as appropriate and set MQFMT_STRING, so that the command server can convert the message. Do not set MQFMT_ADMIN, as that causes your command to be interpreted as PCF.

   ReplyToQ
   Name of your reply-to queue.

   ReplyToQMgr
   If you want replies sent to your local queue manager, leave this field blank. If you want your IBM MQ commands to be sent to a remote queue manager, put its name here. You must also have the correct queues and links set up, as described in Distributed queuing and clusters.

3. Set any other MQMD fields, as required. You should normally use nonpersistent messages for commands.
4. Set any *PutMsgOpts* options, as required.

If you specify MQPMO_SYNCPOINT (the default), you must follow the MQPUT call with a syncpoint call.

Using MQPUT1 and the system-command input queue

If you want to put just one message on the system-command input queue, you can use the **MQPUT1** call. This call combines the functions of an **MQOPEN**, followed by an **MQPUT** of one message, followed by an **MQCLOSE**, all in one call. If you use this call, modify the parameters accordingly. See Putting one message on a queue using the MQPUT1 call for details.

z/OS Retrieving replies to your commands

The command server sends a response to a reply queue for each request message it receives. Any administration application must receive, and handle the reply messages.

When the command processor processes your commands, any reply messages are put onto the reply-to queue specified in the MQPUT call. The command server sends the reply messages with the same persistence as the command message it received.

Waiting for a reply

Use the MQGET call to retrieve a reply from your request message. One request message can produce several reply messages. For details, see "Interpreting the reply messages from the command server" on page 481.

You can specify a time interval that an MQGET call waits for a reply message to be generated. If you do not get a reply, use the checklist beginning in topic "If you do not receive a reply" on page 481.

To use the MQGET call:

1. Set these parameters:

   Hconn
   The connection handle returned by the MQCONN or MQCONNX call.

   Hobj
   The object handle returned by the MQOPEN call for the reply-to queue.

   Buffer
   The name of the area to receive the reply.

   BufferLength
   The length of the buffer to receive the reply. This must be a minimum of 80 bytes.

2. To ensure that you only get the responses from the command that you issued, you must specify the appropriate *MsgId* and *CorrelId* fields. These depend on the report options, MQMD_REPORT, you specified in the MQPUT call:

   MQRO_NONE
   Binary zero, '00...00' (24 nulls).

   MQRO_NEW_MSG_ID
   Binary zero, '00...00' (24 nulls).

   This is the default if none of these options has been specified.

   MQRO_PASS_MSG_ID
   The *MsgId* from the **MQPUT**.

   MQRO_NONE
   The *MsgId* from the MQPUT call.

   MQRO_COPY_MSG_ID_TO_CORREL_ID
   The *MsgId* from the MQPUT call.

   This is the default if none of these options has been specified.

MQRO_PASS_CORREL_ID
The *CorrelId* from the MQPUT call.

For more details on report options, see Report options and message flags.

3. Set the following *GetMsgOpts* fields:

Options
MQGMO_WAIT

If you are not using the same code page as the queue manager, set MQGMO_CONVERT, and set *CodedCharSetId* as appropriate in the MQMD.

WaitInterval
For replies from the local queue manager, try 5 seconds. Coded in milliseconds, this becomes 5 000. For replies from a remote queue manager, and channel control and status commands, try 30 seconds. Coded in milliseconds, this becomes 30 000.

Discarded messages

If the command server finds that a request message is not valid, it discards this message and writes the message CSQN205I to the named reply-to queue. If there is no reply-to queue, the CSQN205I message is put onto the dead-letter queue. The return code in this message shows why the original request message was not valid:

00D5020F	It is not of type MQMT_REQUEST.
00D50210	It has zero length.
00D50212	It is longer than 32 762 bytes.
00D50211	It contains all blanks.
00D5483E	It needed converting, but *Format* was not MQFMT_STRING.
Other	See Command server codes

The command server reply message descriptor

For any reply message, the following MQMD message descriptor fields are set:

MsgType	MQMT_REPLY
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
Priority	As for the MQMD in the message you issued.
Persistence	As for the MQMD in the message you issued.
CorrelId	Depends on the MQPUT report options.
ReplyToQ	None.

The command server sets the *Options* field of the MQPMO structure to MQPMO_NO_SYNCPOINT. This means that you can retrieve the replies as they are created, rather than as a group at the next syncpoint.

z/OS Interpreting the reply messages from the command server

Each request message correctly processed by IBM MQ produces at least two reply messages. Each reply message contains a single IBM MQ user message.

The length of a reply depends on the command that was issued. The longest reply you can get is from a **DISPLAY NAMELIST** command, and that can be up to 15 000 bytes in length.

The first user message, CSQN205I, always contains:

- A count of the replies (in decimal), which you can use as a counter in a loop to get the rest of the replies. The count includes this first message.
- The return code from the command preprocessor.
- A reason code, which is the reason code from the command processor.

This message does not contain a CPF.

For example:

```
CSQN205I    COUNT=    4, RETURN=0000000C, REASON=00000008
```

The COUNT field is 8 bytes long and is right-justified. It always starts at position 18, that is, immediately after COUNT=. The RETURN field is 8 bytes long in character hexadecimal and is immediately after RETURN= at position 35. The REASON field is 8 bytes long in character hexadecimal and is immediately after REASON= at position 52.

If the RETURN= value is 00000000 and the REASON= value is 00000004, the set of reply messages is incomplete. After retrieving the replies indicated by the CSQN205I message, issue a further MQGET call to wait for a further set of replies. The first message in the next set of replies is again CSQN205I, indicating how many replies there are, and whether there are still more to come.

See the IBM MQ for z/OS messages, completion, and reason codes documentation for more details about the individual messages.

If you are using a non-English language feature, the text and layout of the replies are different from those shown here. However, the size and position of the count and return codes in message CSQN205I are the same.

z/OS If you do not receive a reply

There are a series of steps you can take if you do not receive a response to request to the command server.

If you do not receive a reply to your request message, work through this checklist:

- Is the command server running?
- Is the *WaitInterval* long enough?
- Are the system-command input and reply-to queues correctly defined?
- Were the MQOPEN calls to these queues successful?
- Are both the system-command input and reply-to queues enabled for MQPUT and MQGET calls?
- Have you considered increasing the MAXDEPTH and MAXMSGL attributes of your queues?
- Are you are using the *CorrelId* and *MsgId* fields correctly?
- Is the queue manager still running?
- Was the command built correctly?
- Are all your remote links defined and operating correctly?
- Were the MQPUT calls correctly defined?
- Has the reply-to queue been defined as a temporary dynamic queue instead of a permanent dynamic queue? (If the request message is persistent, you must use a permanent dynamic queue for the reply.)

When the command server generates replies but cannot write them to the reply-to queue that you specify, it writes them to the dead-letter queue.

z/OS Passing commands using MGCRE

With appropriate authorization, an application program can make requests to multiple queue managers using a z/OS service routine.

If you have the correct authorization, you can pass IBM MQ commands from your program to multiple queue managers by the MGCRE (SVC 34) z/OS service. See the *z/OS MVS Programming: Authorized Assembler Services Guide* for more information.

The value of the CPF identifies the particular queue manager to which the command is directed. For information about CPFs, see User IDs for command security and command resource security and "Issuing queue manager commands on z/OS" on page 444.

If you use MGCRE, you can use a Command and Response Token (CART) to get the direct responses to the command.

z/OS Examples of commands and their replies

Use this topic as a series of examples of commands to the command server and the responses from the command server.

Here are some examples of commands that could be built into IBM MQ messages, and the user messages that are the replies. Unless otherwise stated, each line of the reply is a separate message.

- Messages from a DEFINE command
- Messages from a DELETE command
- Messages from DISPLAY commands
- Messages from commands with CMDSCOPE
- Messages from commands that generate commands with CMDSCOPE

Messages from a DEFINE command

The following command:

```
DEFINE QLOCAL(Q1)
```

produces these messages:

```
CSQN205I    COUNT=    2, RETURN=00000000, REASON=00000000
CSQ9022I +CSQ1 CSQMMSGP ' DEFINE QLOCAL' NORMAL COMPLETION
```

These reply messages are produced on normal completion.

Messages from a DELETE command

The following command:

```
DELETE QLOCAL(Q2)
```

produces these messages:

```
CSQN205I    COUNT=    4, RETURN=0000000C, REASON=00000008
CSQM125I +CSQ1 CSQMUQLC QLOCAL (Q2) QSGDISP(QMGR) WAS NOT FOUND
CSQM090E +CSQ1 CSQMUQLC FAILURE REASON CODE X'00D44002'
CSQ9023E +CSQ1 CSQMUQLC ' DELETE QLOCAL' ABNORMAL COMPLETION
```

These messages indicate that a local queue called Q2 does not exist.

Messages from DISPLAY commands

The following examples show the replies from some DISPLAY commands.

Finding out the name of the dead-letter queue

If you want to find out the name of the dead-letter queue for a queue manager, issue this command from an application program:

```
DISPLAY QMGR DEADQ
```

The following three user messages are returned, from which you can extract the required name:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM409I +CSQ1 QMNAME(CSQ1) DEADQ(SYSTEM.DEAD.QUEUE          )
CSQ9022I +CSQ1 CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
```

Messages from the DISPLAY QUEUE command

The following examples show how the results from a command depend on the attributes specified in that command.

Example 1

You define a local queue using the command:

```
DEFINE QLOCAL(Q1) DESCR('A sample queue') GET(ENABLED) SHARE
```

If you issue the following command from an application program:

```
DISPLAY QUEUE(Q1) SHARE GET DESCR
```

these three user messages are returned:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM401I +CSQ1 QUEUE(Q1                       ) TYPE(
QLOCAL ) QSGDISP(QMGR   )
DESCR(A sample queue
) SHARE  GET(ENABLED  )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second message, CSQM401I, is shown here occupying four lines.

Example 2

Two queues have names beginning with the letter A:

- A1 is a local queue with its PUT attribute set to DISABLED.

- A2 is a remote queue with its PUT attribute set to ENABLED.

If you issue the following command from an application program:

```
DISPLAY QUEUE(A*) PUT
```

these four user messages are returned:

```
CSQN205I    COUNT=    4, RETURN=00000000, REASON=00000000
CSQM401I +CSQ1 QUEUE(A1                       ) TYPE(
QLOCAL ) QSGDISP(QMGR   )
PUT(DISABLED )
CSQM406I +CSQ1 QUEUE(A2                       ) TYPE(
QREMOTE ) PUT(ENABLED  )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second and third messages, CSQM401I and CSQM406I, are shown here occupying three and two lines.

Messages from the DISPLAY NAMELIST command

You define a namelist using the command:

```
DEFINE NAMELIST(N1) NAMES(Q1,SAMPLE_QUEUE)
```

If you issue the following command from an application program:

```
DISPLAY NAMELIST(N1) NAMES NAMCOUNT
```

the following three user messages are returned:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM407I +CSQ1 NAMELIST(N1                       ) QS
GDISP(QMGR   ) NAMCOUNT(     2) NAMES(Q1
,SAMPLE_QUEUE                 )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY NAMELIST' NORMAL COMPLETION
```

Note: The second message, CSQM407I, is shown here occupying three lines.

Messages from commands with CMDSCOPE

The following examples show the replies from commands that have been entered with the CMDSCOPE attribute.

Messages from the ALTER PROCESS command

The following command:

```
ALT PRO(V4) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'ALT PRO' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    5, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'ALT PRO' command responses from MQ26
CSQM125I !MQ26 CSQMMSGP PROCESS(V4) QSGDISP(QMGR) WAS NOT FOUND
CSQM090E !MQ26 CSQMMSGP FAILURE REASON CODE X'00D44002'
CSQ9023E !MQ26 CSQMMSGP ' ALT PRO' ABNORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'ALT PRO' command responses from MQ25
CSQ9022I !MQ25 CSQMMSGP ' ALT PRO' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=0000000C, REASON=00000008
CSQN123E !MQ25 'ALT PRO' command for CMDSCOPE(*) abnormal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). The command was successful on MQ25 but the process definition did not exist on MQ26, so the command failed on that queue manager.

Messages from the DISPLAY PROCESS command

The following command:

```
DIS PRO(V*) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'DIS PRO' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    5, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS PRO' command responses from MQ26
CSQM408I !MQ26 PROCESS(V2) QSGDISP(COPY)
CSQM408I !MQ26 PROCESS(V3) QSGDISP(QMGR)
CSQ9022I !MQ26 CSQMDRTS ' DIS PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    7, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS PRO' command responses from MQ25
CSQM408I !MQ25 PROCESS(V2) QSGDISP(COPY)
CSQM408I !MQ25 PROCESS(V2) QSGDISP(GROUP)
CSQM408I !MQ25 PROCESS(V3) QSGDISP(QMGR)
CSQM408I !MQ25 PROCESS(V4) QSGDISP(QMGR)
CSQ9022I !MQ25 CSQMDRTS ' DIS PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DIS PRO' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Information is displayed about all the processes on each queue manager with names starting with the letter V.

Messages from the DISPLAY CHSTATUS command

The following command:

```
DIS CHS(VT) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'DIS CHS' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    4, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS CHS' command responses from MQ25
CSQM422I !MQ25 CHSTATUS(VT) CHLDISP(PRIVATE) CONNAME( ) CURRENT STATUS(STOPPED)
CSQ9022I !MQ25 CSQXDRTS ' DIS CHS' NORMAL COMPLETION
CSQN205I  COUNT=    4, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS CHS' command responses from MQ26
CSQM422I !MQ26 CHSTATUS(VT) CHLDISP(PRIVATE) CONNAME( ) CURRENT STATUS(STOPPED)
CSQ9022I !MQ26 CSQXDRTS ' DIS CHS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DIS CHS' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Information is displayed about channel status on each queue manager.

Messages from the STOP CHANNEL command

The following command:

```
STOP CHL(VT) CMDSCOPE(*)
```

produces these messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'STOP CHL' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ25
CSQM134I !MQ25 CSQMTCHL STOP CHL(VT) COMMAND ACCEPTED
SQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ26
CSQM134I !MQ26 CSQMTCHL STOP CHL(VT) COMMAND ACCEPTED
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ26
CSQ9022I !MQ26 CSQXCRPS ' STOP CHL' NORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ25
CSQ9022I !MQ25 CSQXCRPS ' STOP CHL' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'STOP CHL' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Channel VT was stopped on each queue manager.

Messages from commands that generate commands with CMDSCOPE

The following command:

```
DEF PRO(V2) QSGDISP(GROUP)
```

produces these messages:

```
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQM122I !MQ25 CSQMMSGP ' DEF PRO' COMPLETED FOR QSGDISP(GROUP)
CSQN138I !MQ25 'DEFINE PRO' command generated for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DEFINE PRO' command responses from MQ25
CSQ9022I !MQ25 CSQMMSGP ' DEFINE PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DEFINE PRO' command responses from MQ26
CSQ9022I !MQ26 CSQMMSGP ' DEFINE PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DEFINE PRO' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25. When the object was created on the shared repository, another command was generated and sent to all the active queue managers in the queue sharing group (MQ25 and MQ26).

z/OS Managing IBM MQ resources on z/OS

Use the links in this topic to find out how to manage the resources used by IBM MQ for z/OS, for example, managing log files, data sets, page sets, buffer pools, and coupling facility structures.

Use the following links for details of the different administrative tasks you might have to complete while using IBM MQ for z/OS:

- “Managing the logs” on page 488
- “Managing the bootstrap data set (BSDS)” on page 496
- “Managing page sets” on page 504
- “How to back up and recover page sets” on page 510
- “How to back up and restore queues using CSQUTIL” on page 513
- “Managing buffer pools” on page 514
- “Managing queue sharing groups and shared queues on z/OS” on page 516

Related concepts

IBM MQ for z/OS concepts

“Administering IBM MQ for z/OS” on page 444
IBM MQ for z/OS can be controlled and managed by MQSC and PCF commands, by a set of utilities and programs provided with the product, and by authorized applications.

“Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS” on page 444
You can issue MQSC and PCF commands from the IBM MQ for z/OS console, the initialization input data sets, the batch utility CSQUTIL, or from authorized applications. Not all commands can be issued from all these sources.

“Recovery and restart on z/OS” on page 525
Use this topic to understand the recovery and restart mechanisms used by IBM MQ.

Related tasks

Planning your IBM MQ environment on z/OS

Configuring queue managers on z/OS

IBM MQ utilities on z/OS reference

Related reference

“Using the IBM MQ for z/OS utilities” on page 467
IBM MQ for z/OS provides a set of utility programs that you can use to help with system administration.

Programmable command formats reference

z/OS Managing the logs

Use this topic to understand how to manage your IBM MQ log files, including the log archiving process, using log record compression, log record recovery, and printing log records.

This topic describes the tasks involved in managing the IBM MQ logs. It contains these sections:

z/OS *Archiving logs with the ARCHIVE LOG command*

An authorized operator can archive the current IBM MQ active log data sets whenever required using the **ARCHIVE LOG** command.

When you issue the ARCHIVE LOG command, IBM MQ truncates the current active log data sets, then runs an asynchronous offload process, and updates the BSDS with a record of the offload process.

The **ARCHIVE LOG** command has a **MODE(QUIESCE)** option. With this option, IBM MQ jobs and users are quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is offloaded.

Consider using the **MODE(QUIESCE)** option when planning a backup strategy for off site recovery. It creates a system-wide point of consistency, which minimizes the number of data inconsistencies when the archive log is used with the most current backup page set copy during recovery. For example:

```
ARCHIVE LOG MODE(QUIESCE)
```

If you issue the **ARCHIVE LOG** command without specifying a **TIME** parameter, the quiesce time period defaults to the value of the **QUIESCE** parameter of the CSQ6ARVP macro. If the time required for the ARCHIVE LOG MODE(QUIESCE) to complete is less than the time specified, the command completes successfully; otherwise, the command fails when the time period expires. You can specify the time period explicitly by using the **TIME** option, for example:

```
ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

This command specifies a quiesce period of up to 60 seconds before **ARCHIVE LOG** processing occurs.

Attention: Using the **TIME** option when time is critical can significantly disrupt IBM MQ availability for all jobs and users that use IBM MQ resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other IBM MQ commands use the WAIT(YES) option with **QUIESCE**, but be aware that the z/OS console is locked from IBM MQ command input for the entire **QUIESCE** period.)

During the quiesce period:

- Jobs and users on the queue manager are allowed to go through commit processing, but are suspended if they try to update any IBM MQ resource after the commit.
- Jobs and users that only read data can be affected, since they might be waiting for locks held by jobs or users that were suspended.
- New tasks can start, but they cannot update data.

The output from the **DISPLAY LOG** command uses the message CSQV400I to indicate that a quiesce is in effect.

For example:

```
CSQJ322I +CSQ1 DISPLAY LOG report ...
Parameter   Initial value          SET value
----------- ---------------------- ----------------------
INBUFF      60
OUTBUFF     400
MAXRTU      2
MAXARCH     2
TWOACTV     YES
TWOARCH     YES
TWOBSDS     YES
OFFLOAD     YES
```

```
MAXCNOFF    0
WRTHRSH     20
DEALLCT     0
COMPLOG     NONE
ZHYWRITE    NO
End of LOG report
CSQJ370I +CSQ1 LOG status report ...
Copy %Full zHyperWrite Encrypted DSName
 1     68  NO          NO        VICY.CSQ1.LOGCOPY1.DS01
 2     68  NO          NO        VICY.CSQ1.LOGCOPY2.DS01
Restarted at 2019-08-15 09:49:30 using RBA=000000000891B000
Latest RBA=000000000891CCF8
Offload task is AVAILABLE
Full logs to offload - 0 of 4
CSQV400I +CSQ1 ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
CSQ9022I +CSQ1 CSQJC001 ' DISPLAY LOG' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. IBM MQ truncates the current active log data sets, switches to the next available active log data sets, and issues message CSQJ311I stating that the offload process started.

If updates cannot be quiesced before the quiesce period expires, IBM MQ issues message CSQJ317I, and **ARCHIVE LOG** processing terminates. The current active log data sets are not truncated, nor switched to the next available log data sets, and the offload process is not started.

Whether the quiesce was successful or not, all suspended users and jobs are then resumed, and IBM MQ issues message CSQJ312I, stating that the quiesce is ended and update activity is resumed.

If **ARCHIVE LOG** is issued when the current active log is the last available active log data set, the command is not processed, and IBM MQ issues the following message:

```
CSQJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
AVAILABLE ACTIVE LOG DATA SET. ARCHIVE LOG PROCESSING
WILL BE TERMINATED
```

If **ARCHIVE LOG** is issued when another **ARCHIVE LOG** command is already in progress, the new command is not processed, and IBM MQ issues the following message:

```
CSQJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS
```

For information about the messages issued during archiving, see Messages for IBM MQ for z/OS.

Restarting the log archive process after a failure

If there is a problem during the log archive process (for example, a problem with allocation or tape mounts), the archiving of the active log might be suspended. You can cancel the archive process and restart it by using the following command:

```
ARCHIVE LOG CANCEL OFFLOAD
```

This command cancels any offload processing currently in progress, and restarts the archive process. It starts with the oldest log data set that has not been archived, and proceeds through all active log data sets that need offloading. Any log archive operations that have been suspended are restarted.

Use this command only if you are sure that the current log archive task is no longer functioning, or if you want to restart a previous attempt that failed. This is because the command might cause an abnormal termination of the offload task, which might result in a dump.

z/OS *Controlling archiving and logging*

You can control compression, printing, archiving, recovery and logging with using the CSQ6LOGP, CSQ6ARVP, and CSQ6SYSP macros. Note, that changes to private objects only are logged in IBM MQlogs.

Changes to GROUP objects (like shared inbound channels) are also logged, because the definitions are propagated around the group and held locally.

Many aspects of archiving and logging are controlled by parameters set using the CSQ6LOGP, CSQ6ARVP and CSQ6SYSP macros of the system parameter module when the queue manager is customized. See Tailor your system parameter module for details of these macros.

Some of these parameters can be changed while a queue manager is running using the IBM MQ MQSC SET LOG, SET SYSTEM and SET ARCHIVE commands. They are shown in Table 30 on page 490:

Table 30. Archiving and logging parameters that can be changed while a queue manager is running

SET command	**Parameters**
LOG	WRTHRSH, MAXARCH, DEALLCT, MAXRTU, COMPLOG
ARCHIVE	All
SYSTEM	LOGLOAD

You can display the settings of all the parameters using the MQSC DISPLAY LOG, DISPLAY ARCHIVE and DISPLAY SYSTEM commands. These commands also show status information about archiving and logging.

Controlling log compression

You can enable and disable the compression of log records using either

- The SET and DISPLAY LOG commands in MQSC; see The MQSC commands
- Invoking PCF interface. See "Introduction to IBM MQ Programmable Command Formats" on page 25
- Using the CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

Printing log records

You can extract and print log records using the CSQ1LOGP utility. For instructions, see The log print utility.

Recovering logs

Normally, you do not need to back up and restore the IBM MQ logs, especially if you are using dual logging. However, in rare circumstances, such as an I/O error on a log, you might need to recover the logs. Use Access Method Services to delete and redefine the data set, and then copy the corresponding dual log into it.

z/OS *Discarding archive log data sets*

You can discard your archive log data sets and choose to discard the logs automatically or manually.

You must keep enough log data to be able to perform unit of work recovery, page set media recovery if a page set is lost, or CF structure media recovery if a CF structure is lost. Do not discard archive log data sets that might be required for recovery; if you discard these archive log data sets you might not be able to perform required recovery operations.

If you have confirmed that your archive log data sets can be discarded, you can do this in either of the following ways:

- Automatic archive log data set deletion
- Manually deleting archive log data sets

Automatic archive log data set deletion

You can use a DASD or tape management system to delete archive log data sets automatically. The retention period for IBM MQ archive log data sets is specified by the retention period field ARCRETN in the CSQ6ARVP installation macro (see the Using CSQ6ARVP for more information).

The default for the retention period specifies that archive logs are to be kept for 9999 days (the maximum).

Important: You can change the retention period but you must ensure that you can accommodate the number of backup cycles that you have planned for.

.

IBM MQ uses the retention period value as the value for the JCL parameter RETPD when archive log data sets are created.

The retention period set by the MVS/DFP storage management subsystem (SMS) can be overridden by this IBM MQ parameter. Typically, the retention period is set to the smaller value specified by either IBM MQ or SMS. The storage administrator and IBM MQ administrator must agree on a retention period value that is appropriate for IBM MQ.

Note: IBM MQ does not have an automated method to delete information about archive log data sets from the BSDS, because some tape management systems provide external manual overrides of retention periods. Therefore, information about an archive log data set can still be in the BSDS long after the data set retention period has expired and the data set has been scratched by the tape management system. Conversely, the maximum number of archive log data sets might have been exceeded and the data from the BSDS might have been dropped before the data set has reached its expiration date.

If archive log data sets are deleted automatically, remember that the operation does not update the list of archive logs in the BSDS. You can update the BSDS with the change log inventory utility, as described in “Changing the BSDS” on page 498. The update is not essential. Recording old archive logs wastes space in the BSDS, but does no other harm.

Manually deleting archive log data sets

You must keep all the log records as far back as the lowest RBA identified in messages CSQI024I and CSQI025I. This RBA is obtained using the DISPLAY USAGE command that you issued when creating a point of recovery using Method 1: Full backup.

Read Creating a point of recovery for non-shared resources before discarding any logs.

Locate and discard archive log data sets

Having established the minimum log RBA required for recovery, you can find archive log data sets that contain only earlier log records by performing the following procedure:

1. Use the print log map utility to print the contents of the BSDS. For an example of the output, see The print log map utility.
2. Find the sections of the output titled `ARCHIVE LOG COPY n DATA SETS`. If you use dual logging, there are two sections. The columns labeled STARTRBA and ENDRBA show the range of RBAs contained in each volume. Find the volumes with ranges that include the minimum RBA you found with messages CSQI024I and CSQI025I. These are the earliest volumes you need to keep. If you are using dual-logging, there are two such volumes.

   If no volumes have an appropriate range, one of the following cases applies:

   - The minimum RBA has not yet been archived, and you can discard all archive log volumes.
   - The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number allowed by the MAXARCH parameter of the CSQ6LOGP macro. If the BSDS does not register an archive log volume, that volume cannot be used for recovery. Therefore,

consider adding information about existing volumes to the BSDS. For instructions, see “Changes for archive logs” on page 500.

Also consider increasing the value of MAXARCH. For information, see the Using CSQ6LOGP.

3. Delete any archive log data set or volume with an ENDRBA value that is less than the STARTRBA value of the earliest volume you want to keep. If you are using dual logging, delete both such copies.

   Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the end. Look at the combination of date and time and compare their ages. Do not assume that you can discard all entries before the entry for the archive log containing the minimum LOGRBA.

   Delete the data sets. If the archives are on tape, erase the tapes. If they are on DASD, run a z/OS utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility (CSQJU003) to delete entries for the discarded volumes. See “Changes for archive logs” on page 500 for an example.

z/OS *The effect of log shunting*

Long running transactions can cause unit of work log records which span log data sets. IBM MQ handles this scenario by using log shunting, a technique which moves the log records to optimize the quantity of log data retained, and queue manager restart time.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This is known as *log shunting*. It is described more fully in Log files.

The queue manager uses these shunted log records instead of the originals after a failure, to ensure unit of work integrity. There are two benefits to this:

- the quantity of log data which must be retained for unit of work coordination is reduced
- less log data must be traversed at queue manager restart time, so the queue manager is restarted more quickly

Shunted log records do not contain sufficient information for media recovery operations.

Data held in the log is used for two distinct purposes; media recovery and unit of work coordination. If a media failure occurs which affects either a CF structure or page set, the queue manager can recover the media to the point of failure by restoring a prior copy and updating this using data contained in the log. Persistent activity performed in a unit of work is recorded on the log so that in the event of a failure, it can either be backed out or locks can be recovered on changed resources. The quantity of log data you need to retain to enable queue manager recovery is affected by these two elements.

For media recovery, you must retain sufficient log data to be able to perform media recovery from at least the most recent media copy and to be able to back out. (Your site may stipulate the ability to recover from older backups.) For unit of work integrity, you must retain the log data for your oldest in flight or indoubt units of work.

To assist you with managing the system, the queue manager detects old units of work at each log archive and reports them in messages CSQJ160 and CSQJ161. An internal task reads unit of work log information for these old units of work and rewrites it in a more succinct form to the current position in the log. Message CSQR026 indicates when this has happened. The MQSC command DISPLAY USAGE TYPE(DATASET) can also help you to manage the retention of log data. The command reports the following three pieces of recovery information:

1. How much of the log must be retained for unit of work recovery.
2. How much of the log must be retained for media recovery of page sets.
3. For a queue manager in a queue sharing group, how much of the log must be retained for media recovery of CF structures.

For each of these pieces of information, an attempt is made to map the oldest log data required into a data set. As new units of work start and stop, (1) would be expected to move to a more recent position in the log. If it is not moving, the long running UOW messages warn you that there is an issue. (2) relates

to page set media recovery if the queue manager were to be shut down now and restarted. It does not know about when you last backed up your page sets, or which backup you might have to use if there was a page set failure. It normally moves to a more recent position in the log during checkpoint processing as changes held in the buffer pools are written to the page sets. In (3), the queue manager does know about CF structure backups taken either on this queue manager or on other queue managers in the queue sharing group. However, CF structure recovery requires a merge of log data from all queue managers in the queue sharing group which have interacted with the CF structure since the last backup. This means that the log data is identified by a log record sequence number, (or LRSN), which is timestamp based and so applicable across the entire queue sharing group rather than an RBA which would be different on different queue managers in the queue sharing group. It normally moves to a more recent position in the log as BACKUP CFSTRUCT commands are performed on either this or other queue managers in the queue sharing group.

z/OS *Resetting the queue manager's log*

Use this topic to understand how to reset the queue manager's log.

You must not allow the queue manager log RBA to wrap around from the end of the log RBA range to 0, as this leads to a queue manager outage and all persistent data will become unrecoverable. The end of the log RBA is either a value of FFFFFFFFFFFF (if 6-byte RBAs as in use), or FFFFFFFFFFFFFFFF (if 8-byte RBAs are in use).

The queue manager issues messages CSQI045I, CSQI046E, CSQI047E, CSQJ031D, and CSQJ032E to indicate that the used log range is significant and that you should plan to take action to avoid an unplanned outage.

The queue manager terminates with reason code 00D10257 when the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use).

If 6-byte log RBAs are in use, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the process described in “Implementing the larger log Relative Byte Address” on page 495. Converting a queue manager to use 8-byte log RBAs requires a shorter outage than resetting the log, and increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6-byte or 8-byte log RBAs are in use.

The procedure to follow to reset the queue manager's log is as follows:

1. Resolve any unresolved units of work. The number of unresolved units of work is displayed at queue manager startup in message CSQR005I as the INDOUBT count. At each checkpoint, and at queue manager shutdown, the queue manager automatically issues the command

   DISPLAY CONN(*) TYPE(CONN) ALL WHERE(UOWSTATE EQ UNRESOLVED) to provide information about unresolved units of work.

   See How in-doubt units of recovery are resolved for information on resolving units of recovery. The ultimate recourse is to use the **RESOLVE INDOUBT** MQSC command to manually resolve indoubt units of recovery.

2. Shut down the queue manager cleanly.

   You can use either **STOP QMGR** or **STOP QMGR MODE(FORCE)** as both these commands flush any changed pages from bufferpools to the page sets.

3. If a queue manager is part of a queue sharing group, take CFSTRUCT backups on other queue managers for all structures in the queue sharing group. This ensures that the most recent backups are not in this queue manager's log, and that this queue manager's log is not required for CFSTRUCT recovery.

4. Define new logs and BSDS using CSQJU003 (see The change log inventory utility for more information on using the change log inventory utility).

5. Run **CSQUTIL RESETPAGE** against all the page sets for this queue manager (see Copying a page and resetting the log for more information on using this function). Note that page set RBAs can be reset

independently, so multiple concurrent jobs (for example, one per page set) can be submitted to reduce the elapsed time for this step.

6. Restart the queue manager

Warning messages

When IBM MQ detects that the end of the log is approaching, it issues console messages in the following order, which indicate that a log reset should be planned. In this section the messages show 6-byte log RBA values. The same console messages are issued when IBM MQ is running in 8-byte log RBA mode but with different values; see “Warning thresholds” on page 495 for the 8-byte log RBA thresholds.

1. When IBM MQ detects that the end of the log is approaching in the near future, (approximately 94% full) IBM MQ issues console message CSQI045I, as in the following example:

```
CSQI045I -CSQ7 CSQILCUR Log RBA has reached 0000F00000000000.
Plan a log reset
```

2. IBM MQ issues the following CSQI046E error console message when the end of the log is near (approximately 97% full). This informs the IBM MQ administrator to take action soon.

```
CSQI046E -CSQ7 CSQILCUR Log RBA has reached 0000F80000000000.
Perform a log reset
```

3. After the CSQI046E message is issued, at the next log switch, IBM MQ issues the following CSQJ032E console message with the word WARNING:

```
CSQJ032E -CSQ7 CSQJW307 WARNING - APPROACHING END OF
THE LOG RBA RANGE OF 0000FFFFFFFFFFFF. CURRENT LOG RBA IS 0000F80000022000.
```

4. After the CSQI046E and CSQJ032E console messages are issued, IBM MQ issues one more error message, which does not require immediate IBM MQ administrator intervention. IBM MQ issues console message CSQI047E (when the log is approximately 99% full):

```
CSQI047E -CSQ7 CSQILCUR Log RBA has reached 0000FF0000000000.
Stop queue manager and reset logs
```

5. When the log RBA reaches FF8000000000, IBM MQ increases the urgency of the situation and issues console message CSQJ032E with the word CRITICAL:

```
CSQJ032E -CSQ7 CSQJW009 CRITICAL - APPROACHING END OF THE LOG RBA RANGE OF 0000FFFFFFFFFFFF.
CURRENT LOG RBA IS 0000FFF7FFFFDFFF.
```

6. If the queue manager is started when the log RBA is almost at the maximum, the following CSQJ031D console message is issued. This stage requires the input of the IBM MQ administrator:.

```
CSQJ031D -CSQ7 CSQYSTRT THE LOG RBA RANGE MUST BE RESET.
REPLY 'Y' TO CONTINUE STARTUP OR 'N' TO SHUTDOWN
```

7. IBM MQ startup remains suspended until a reply is given to message CSQJ031D.

The purpose of these messages is to give the IBM MQ administrator time to plan for a system outage to reset the logs. In an ideal configuration, there are at least two queue managers, possibly in a queue sharing group (QSG), sharing the workload. When one is down for maintenance the other can continue to receive work.

The severity of console messages that IBM MQ issues becomes greater as the RBA gets closer to the end. Ideally your IBM MQ administrator should plan to reset the log RBA when the first console message is seen.

If the warning and error console messages are ignored, IBM MQ terminates with reason code 5C6-00D10257 when the log RBA reaches FFF800000000, at which point IBM MQ determines that the available range is too small for the queue manager to continue. When this point is reached, the only option is to take an outage and either reset the log or extend the size of the log RBA.

Note: When the end of the log is reached it is not possible to resolve any in-flight units of work (UOW); these are lost during the log reset process. Enough of the RBA range should be left to start the queue manager and resolve any UOW. Because IBM MQ issues console messages several times to inform that the end of the log is approaching, a log reset should be planned.

The preferred option to avoid losing any in-flight UOW is to extend the log RBA to use 8 bytes. This means that a log RBA reset will not be necessary for a long period.

Warning thresholds

The following table lists the thresholds, based on the length of the log RBA.

Console message	6-byte log RBA	8-byte log RBA
CSQI045I	0000F00000000000	FFFF800000000000
CSQI046E	0000F80000000000	FFFFC00000000000
CSQI047E	0000FF8000000000	FFFFFC0000000000
CSQJ032E	0000FF8000000000 0000FF8000000000	FFFFFC0000000000 FFFFFC0000000000
CSQJ031D	0000FF8000000000	FFFFFC0000000000

Notes:

1. For message CSQJ032E, the first number applies to the WARNING text and the second number applies to the CRITICAL text in the console message.
2. Message CSQJ031D is issued at IBM MQ initialization only.

Related concepts

“Implementing the larger log Relative Byte Address” on page 495
Before IBM MQ for z/OS 8.0, IBM MQ for z/OS used a 6 byte log RBA to identify the location of data within the log. From IBM MQ for z/OS 8.0, the log RBA can be 8 bytes long, increasing the period of time before you have to reset the log.

z/OS *Implementing the larger log Relative Byte Address*

Before IBM MQ for z/OS 8.0, IBM MQ for z/OS used a 6 byte log RBA to identify the location of data within the log. From IBM MQ for z/OS 8.0, the log RBA can be 8 bytes long, increasing the period of time before you have to reset the log.

Attention: You only have to carry out the following procedure to enable this feature if your queue managers were created before IBM MQ 9.3.0, as queue managers created at IBM MQ 9.3.0 and later already have this feature enabled.

See Planning to increase the maximum addressable log range for considerations when planning to enable 8 byte log RBA.

Perform these instructions, in the order shown, to enable 8 byte log RBA on a single IBM MQ for z/OS queue manager. For queue managers in a queue sharing group, perform the steps on each queue manager in turn.

1. Allocate new BSDS data sets with similar attributes to the current BSDS. You can tailor sample CSQ4BSDS and delete any irrelevant statement, or you can use your existing JCL, but change the BSDS name to something like `++HLQ++.NEW.BSDS01`.

   Notes:

   a. Check the attributes of your new BSDS before submitting the job to allocate the new BSDS. The only attribute that might change is the size of the BSDS.

b. The new BSDS contains more data that the current BSDS, therefore, you must ensure that the new data sets are allocated with sufficient available space. The sample JCL in thlqual.SCSQPROC(CSQ4BSDS) contains the recommended values when defining a new BSDS.

2. Shut down the queue manager cleanly.
3. Run the BSDS conversion utility (CSQJUCNV) to convert the existing BSDS to the new BSDS data sets. This usually takes a few seconds to run.

   Your existing BSDS will not be changed during this process, and you can use that for the initialization of the queue manager in the case of an unsuccessful conversion.
4. Rename the current BSDS to become the old BSDS, and the new BSDS to become the current BSDS, so that the new data sets are used when you next restart the queue manager. You can use the DFSMS Access Method Services ALTER command, for example:

   ```
   ALTER '++HLQ++.BSDS01' NEWNAME('++HLQ++.OLD.BSDS01')
   ALTER '++HLQ++.NEW.BSDS01' NEWNAME('++HLQ++.BSDS01')
   ```

   Ensure that you also issue commands to rename both the data and index portions of the VSAM cluster.
5. Restart the queue manager. It should start in the same amount of time as it would have done when using 6 byte log RBA.

   If the queue manager does not restart successfully due to a failure to access the converted BSDS, attempt to identify the cause of the failure, resolve the problem and retry the operation. If required, contact IBM Support for assistance.

   If necessary, the change can be backed out at this point by:

   a. Renaming the current BSDS to become the new BSDS.
   b. Renaming the old BSDS to become the current BSDS.
   c. Restarting the queue manager.
6. Once the queue manager has been successfully restarted with the converted BSDS, do not attempt to start the queue manager using the old BSDS.
7. Message CSQJ034I is issued during queue manager initialization to indicate the end of the log RBA for the queue manager as configured. Confirm that the end of the log RBA range displayed is FFFFFFFFFFFFFFFF. This indicates that 8 byte log RBA is in use.

Related tasks

Planning to increase the maximum addressable log range

Related reference

Larger log Relative Byte Address

The BSDS conversion utility (CSQJUCNV)

z/OS Managing the bootstrap data set (BSDS)

The bootstrap data set (BSDS) is used to reference log data sets, and log records. Use this topic to understand how you can examine, change, and recover the BSDS.

For more information, see The bootstrap data set.

This topic describes the tasks involved in managing the bootstrap data set. It contains these sections:

- “Finding out what the BSDS contains” on page 497
- “Changing the BSDS” on page 498
- “Recovering the BSDS” on page 502

z/OS *Finding out what the BSDS contains*

You can use the print log map utility (CSQJU004) to examine the contents of the BSDS.

The print log map utility (CSQJU004) is a batch utility that lists the information stored in the BSDS. For instructions on running it, see The print log map utility.

The BSDS contains:

- Time stamps
- Active log data set status

Time stamps in the BSDS

The output of the print log map utility shows the time stamps, which are used to record the date and time of various system events, that are stored in the BSDS.

The following time stamps are included in the header section of the report:

SYSTEM TIMESTAMP
Reflects the date and time the BSDS was last updated. The BSDS time stamp can be updated when:

- The queue manager starts.
- The write threshold is reached during log write activities. Depending on the number of output buffers you have specified and the system activity rate, the BSDS might be updated several times a second, or might not be updated for several seconds, minutes, or even hours. For details of the write threshold, see the WRTHRSH parameter of the CSQ6LOGP macro in Using CSQ6LOGP.
- IBM MQ drops into a single BSDS mode from its normal dual BSDS mode due to an error. This can occur when a request to get, insert, point to, update, or delete a BSDS record is unsuccessful. When this error occurs, IBM MQ updates the time stamp in the remaining BSDS to force a time stamp mismatch with the disabled BSDS.

UTILITY TIMESTAMP
The date and time the contents of the BSDS were altered by the change log inventory utility (CSQJU003).

The following time stamps are included in the active and archive log data sets portion of the report:

Active log date
The date the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Active log time
The time the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Archive log date
The date the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Archive log time
The time the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Active log data set status

The BSDS records the status of an active log data set as one of the following:

NEW
The data set has been defined but never used by IBM MQ, or the log was truncated to a point before the data set was first used. In either case, the data set starting and ending RBA values are reset to zero.

REUSABLE
Either the data set has been defined but never used by IBM MQ, or the data set has been offloaded. In the print log map output, the start RBA value for the last REUSABLE data set is equal to the start RBA value of the last archive log data set.

NOT REUSABLE
The data set contains records that have not been offloaded.

STOPPED
The offload processor encountered an error while reading a record, and that record could not be obtained from the other copy of the active log.

TRUNCATED
Either:

- An I/O error occurred, and IBM MQ has stopped writing to this data set. The active log data set is offloaded, beginning with the starting RBA and continuing up to the last valid record segment in the truncated active log data set. The RBA of the last valid record segment is lower than the ending RBA of the active log data set. Logging is switched to the next available active log data set, and continues uninterrupted.

or

- An ARCHIVE LOG function has been called, which has truncated the active log.

The status appears in the output from the print log map utility.

z/OS *Changing the BSDS*

You do not have to take special steps to keep the BSDS updated with records of logging events because IBM MQ does that automatically.

However, you might want to change the BSDS if you do any of the following:

- Add more active log data sets.
- Copy active log data sets to newly allocated data sets, for example, when providing larger active log allocations.
- Move log data sets to other devices.
- Recover a damaged BSDS.
- Discard outdated archive log data sets.

You can change the BSDS by running the change log inventory utility (CSQJU003). Only run this utility when the queue manager is inactive, or you might get inconsistent results. The action of the utility is controlled by statements in the SYSIN data set. This section shows several examples. For complete instructions, see The change log inventory utility.

You can copy an active log data set only when the queue manager is inactive because IBM MQ allocates the active log data sets as exclusive (DISP=OLD) at queue manager startup.

z/OS *Changes for active logs*

Use this topic to understand how you can change the active logs using the BSDS.

You can add to, delete from, and record entries in the BSDS for active logs using the change log utility. Examples only are shown here; replace the data set names shown with the ones you want to use. For more details of the utility, see The change log inventory utility.

See these sections for more information:

- Adding record entries to the BSDS
- Deleting information about the active log data set from the BSDS
- Recording information about the log data set in the BSDS
- Increasing the size of the active log
- The use of CSQJUFMT

Adding record entries to the BSDS

If an active log has been flagged as "stopped", it is not reused for logging; however, it continues to be used for reading. Use the access method services to define new active log data sets, then use the change log inventory utility to register the new data sets in the BSDS. For example, use:

```
NEWLOG DSNAME=MQM111.LOGCOPY1.DS10,COPY1
NEWLOG DSNAME=MQM111.LOGCOPY2.DS10,COPY2
```

If you are copying the contents of an old active log data set to the new one, you can also give the RBA range and the starting and ending time stamps on the NEWLOG function.

Deleting information about the active log data set from the BSDS

To delete information about an active log data set from the BSDS, you could use:

```
DELETE DSNAME=MQM111.LOGCOPY1.DS99
DELETE DSNAME=MQM111.LOGCOPY2.DS99
```

Recording information about the log data set in the BSDS

To record information about an existing active log data set in the BSDS, use:

```
NEWLOG DSNAME=MQM111.LOGCOPY1.DS10,COPY2,STARTIME=19930212205198,
ENDTIME=19930412205200,STARTRBA=6400,ENDRBA=94FF
```

You might need to insert a record containing this type of information in the BSDS because:

- The entry for the data set has been deleted, but is needed again.
- You are copying the contents of one active log data set to another data set.
- You are recovering the BSDS from a backup copy.

Increasing the size of the active log

There are two methods of achieving this process.

1. When the queue manager is active:
   a. Define new larger log data sets using JCL.
   b. Add the new log data sets to the active queue manager using the MQSC DEFINE LOG command.
   c. Use the MQSC ARCHIVE LOG command to move the current active log, to be a new larger log.
   d. Wait for the archive of the smaller active log data set to complete.
   e. Shut down the queue manager, using the CSQJU003 utility to remove the old small active logs.
   f. Restart the queue manager.
2. When the queue manager is inactive:
   a. Stop the queue manager. This step is required because IBM MQ allocates all active log data sets for its exclusive use when it is active.
   b. Use Access Method Services ALTER with the NEWNAME option to rename your active log data sets.
   c. Use Access Method Services DEFINE to define larger active log data sets.

      By reusing the old data set names, you do not have to run the change log inventory utility to establish new names in the BSDSs. The old data set names and the correct RBA ranges are already in the BSDSs.

d. Use Access Method Services REPRO to copy the old (renamed) data sets into their appropriate new data sets.

   Note: This step can take a long time, so your enterprise could be out of action for this period.

e. Start the queue manager.

If all your log data sets are the same size, your system will be operationally more consistent and efficient. If the log data sets are not the same size, it is more difficult to track your system's logs, and so space can be wasted.

The use of CSQJUFMT

Do not run a CSQJUFMT format when increasing the size of an active log.

If you run CSQJUFMT (in order to provide a performance advantage the first time the queue manager writes to the new active log) you receive messages:

```
IEC070I 203-204,XS95GTLX,REPRO02,OUTPUT,B857,SPMG02, 358
IEC070I MG.W.MG4E.LOGCOPY1.DS02,MG.W.MG4E.LOGCOPY1.DS02.DATA,
IDC3302I ACTION ERROR ON MG.W.MG4E.LOGCOPY1.DS02
IDC3351I ** VSAM I/O RETURN CODE IS 28 - RPLFDBWD = X'2908001C'
IDC31467I MAXIMUM ERROR LIMIT REACHED.

IDC0005I NUMBER OF RECORDS PROCESSED WAS 0
```

In addition, if you use the Access Method Services REPRO, ensure that you define a new empty log.

If you use REPRO to copy the old (renamed) data set into its respective new data set, the default is NOREPLACE.

This means that REPRO does not replace a record that is already on the designated data set. When formatting is done on the data set, the RBA value is reset. The net result is a data set that is not empty after formatting.

z/OS *Changes for archive logs*

Use this topic to understand how to change the archive logs.

You can add to, delete from, and change the password of, entries in the BSDS for archive logs. Examples only are shown here; replace the data set names shown with the ones you want to use. For more details of the utility, see The change log inventory utility.

- Adding an archive log
- Deleting an archive log
- Changing the password of an archive log

Adding an archive log

When the recovery of an object depends on reading an existing archive log data set, the BSDS must contain information about that data set so that IBM MQ can find it. To register information about an existing archive log data set in the BSDS, use:

```
NEWLOG DSNAME=CSQARC1.ARCHLOG1.E00021.T2205197.A0000015,COPY1VOL=CSQV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=NO
```

Deleting an archive log

To delete an entire archive log data set on one or more volumes, use:

```
DELETE DSNAME=CSQARC1.ARCHLOG1.E00021.T2205197.A0000015,COPY1VOL=CSQV04
```

Changing the password of an archive log

If you change the password of an existing archive log data set, you must also change the information in the BSDS.

1. List the BSDS, using the print log map utility.
2. Delete the entry for the archive log data set with the changed password, using the DELETE function of the CSQJU003 utility (see topic The change log inventory utility).
3. Name the data set as for a new archive log data set. Use the NEWLOG function of the CSQJU003 utility (see topic The change log inventory utility), and give the new password, the starting and ending RBAs, and the volume serial numbers (which can be found in the print log map utility output, see The print log map utility).

To change the password for new archive log data sets, use:

```
ARCHIVE PASSWORD= password
```

To stop placing passwords on new archive log data sets, use:

```
ARCHIVE NOPASSWD
```

Note: Only use the ARCHIVE utility function if you do not have an external security manager.

z/OS *Changing the high-level qualifier (HLQ) for the logs and BSDS*

Use this topic to understand the procedure required to change the high-level qualifier (HLQ).

Before you begin

You must end the queue manager normally before copying any of the logs or data sets to the new data sets. This is to ensure that the data is consistent and no recovery is needed during restart.

About this task

This task provides information about how to change the HLQ for the logs and BSDS. To do this, follow these steps:

Procedure

1. Run the log print utility CSQJU004 to record the log data set information. This information is needed later.
2. You can either:
   a) run DSS backup and restore with rename on the log and BSDS data sets to be renamed, or
   b) use AMS DEFINE and REPRO to create the HLQ data sets and copy the data from the old data sets.
3. Modify the MSTR and CHIN procedures to point to the new data sets.
4. Delete the old log information in the new copy of the BSDS using CSQJU003.
5. Define the new log data sets to the new BSDS using the NEWLOG function of CSQJU003.
   Keep all information about each log the same, apart from the HLQ.
6. The new BSDS should reflect the same information that was recorded for the old logs in the old BSDS.
   The HLQ should be the only thing that has changed.

What to do next

Compare the CSQJU004 output for the old and new BSDS to ensure that they look EXACTLY the same (except for the HLQs) before starting the queue manager.

Note: Care must be taken when performing these operations. Incorrect actions might lead to unrecoverable situations. Check the PRINT LOG MAP UTILITY output and make sure that all the information needed for recovery or restart has been included.

z/OS *Recovering the BSDS*

If IBM MQ is operating in dual BSDS mode and one BSDS becomes damaged, forcing IBM MQ into single BSDS mode, IBM MQ continues to operate without a problem (until the next restart).

To return the environment to dual BSDS mode:

1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the damaged BSDS. Example control statements can be found in job CSQ4BREC in thlqual.SCSQPROC.
2. Issue the IBM MQ command RECOVER BSDS to make a copy of the valid BSDS in the newly allocated data set and to reinstate dual BSDS mode.

If IBM MQ is operating in single BSDS mode and the BSDS is damaged, or if IBM MQ is operating in dual BSDS mode and both BSDSs are damaged, the queue manager stops and does not restart until the BSDS data sets are repaired. In this case:

1. Locate the BSDS associated with the most recent archive log data set. The data set name of the most recent archive log appears on the job log in the last occurrence of message CSQJ003I, which indicates that offload processing has been completed successfully. In preparation for the rest of this procedure, it is a good practice to keep a log of all successful archives noted by that message:
   - If archive logs are on DASD, the BSDS is allocated on any available DASD. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in this example:

     Archive log name
     CSQ.ARCHLOG1. ***A*** 0000001

     BSDS copy name
     CSQ.ARCHLOG1. ***B*** 0000001
   - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.
2. If the most recent archive log data set has no copy of the BSDS (for example, because an error occurred when offloading it), locate an earlier copy of the BSDS from earlier offload processing.
3. Rename *damaged* BSDSs using the Access Method Services ALTER command with the NEWNAME option. If you want to delete a damaged BSDS, use the Access Method Services DELETE command. For each damaged BSDS, use Access Method Services to define a new BSDS as a replacement data set. Job CSQ4BREC in thlqual.SCSQPROC contains Access Method Services control statements to define a new BSDS.
4. Use the Access Method Services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs you defined in step “3” on page 502. Do not copy any data to the second replacement BSDS, you do that in step “5” on page 503.

   a. Print the contents of the replacement BSDS.

      Use the print log map utility (CSQJU004) to print the contents of the replacement BSDS. This enables you to review the contents of the replacement BSDS before continuing your recovery work.

   b. Update the archive log data set inventory in the replacement BSDS.

      Examine the output from the print log map utility and check that the replacement BSDS does not contain a record of the archive log from which the BSDS was copied. If the replacement BSDS is an old copy, its inventory might not contain all archive log data sets that were created more recently. The BSDS inventory of the archive log data sets must be updated to reflect the current subsystem inventory.

      Use the change log inventory utility (CSQJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. If the archive log data

set is password-protected, use the PASSWORD option of the NEWLOG function. Also, if the archive log data set is cataloged, ensure that the CATALOG option of the NEWLOG function is properly set to CATALOG=YES. Use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.

c. Update passwords in the replacement BSDS.

   The BSDS contains passwords for the archive log data sets and for the active log data sets. To ensure that the passwords in the replacement BSDS reflect the current passwords used by your installation, use the change log inventory ARCHIVE utility function with the PASSWORD option.

d. Update the active log data set inventory in the replacement BSDS.

   In unusual circumstances, your installation might have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets your installation currently has in use.

   If you need to delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE function.

   If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG function. Ensure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, use the PASSWORD option.

   If you need to rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE function, followed by the NEWLOG function. Ensure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, use the PASSWORD option.

e. Update the active log RBA ranges in the replacement BSDS.

   Later, when the queue manager restarts, it compares the RBAs of the active log data sets listed in the BSDS with the RBAs found in the actual active log data sets. If the RBAs do not agree, the queue manager does not restart. The problem is magnified when an old copy of the BSDS is used. To solve this problem, use the change log inventory utility (CSQJU003) to adjust the RBAs found in the BSDS using the RBAs in the actual active log data sets. You do this by:

   - Using the print log records utility (CSQ1LOGP) to print a summary report of the active log data set. This shows the starting and ending RBAs.
   - Comparing the actual RBA ranges with the RBA ranges you have just printed, when the RBAs of all active log data sets are known.

     If the RBA ranges are equal for all active log data sets, you can proceed to the next recovery step without any additional work.

     If the RBA ranges are not equal, adjust the values in the BSDS to reflect the actual values. For each active log data set that needs to have the RBA range adjusted, use the change log inventory utility DELETE function to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG function to redefine the active log data set to the BSDS. If the active log data sets are password-protected, use the PASSWORD option of the NEWLOG function.

f. If only two active log data sets are specified for each copy of the active log, IBM MQ can have difficulty during queue manager restart. The problem can arise when one of the active log data sets is full and has not been offloaded, while the second active log data set is close to filling. In this case, add a new active log data set for each copy of the active log and define each new active log data set in the replacement BSDS log inventory.

   Use the Access Method Services DEFINE command to define a new active log data set for each copy of the active log and use the change log inventory utility NEWLOG function to define the new active log data sets in the replacement BSDS. You do not need to specify the RBA ranges on the NEWLOG statement. However, if the active log data sets are password-protected, use the PASSWORD option of the NEWLOG function. Example control statements to accomplish this task can be found in job CSQ4LREC in thlqual.SCSQPROC.

5. Copy the updated BSDS to the second new BSDS data set. The BSDSs are now identical.

Use the print log map utility (CSQJU004) to print the contents of the second replacement BSDS at this point.

6. See Active log problems for information about what to do if you have lost your current active log data set.
7. Restart the queue manager using the newly constructed BSDS. IBM MQ determines the current RBA and what active logs need to be archived.

z/OS Managing page sets

Use this topic to understand how to manage the page sets associated with a queue manager.

This topic describes how to add, copy, and generally manage the page sets associated with a queue manager. It contains these sections:

- "How to change the high-level qualifier (HLQ) for the page sets" on page 504
- "How to add a page set to a queue manager" on page 504
- "What to do when one of your page sets becomes full" on page 505
- "How to balance loads on page sets" on page 505
- How to increase the size of a page set
- "How to reduce a page set" on page 509
- "How to reintroduce a page set" on page 509
- "How to back up and recover page sets" on page 510
- "How to delete page sets" on page 513
- "How to back up and restore queues using CSQUTIL" on page 513

See Page sets for a description of page sets, storage classes, buffers, and buffer pools, and some of the performance considerations that apply.

How to change the high-level qualifier (HLQ) for the page sets

This task gives information on how to change the HLQ for the page sets. To perform this task, do the following:

1. Define the new HLQ page sets.
2. If the size allocation is the same as the old page sets, copy the existing page set using REPRO to the empty new HLQ page sets.
3. If you are increasing the size of the page sets, use the FORMAT function of CSQUTIL to format the destination pages, and then the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set.

   For more information, see Formatting page sets (FORMAT), and Expanding a page set (COPYPAGE).
4. Change the CSQP00xx DD statement in the queue manager procedure to point to the new HLQ page sets.

Restart the queue manager and verify the changes to the page sets.

How to add a page set to a queue manager

This description assumes that you have a queue manager that is already running. You might need to add a page set if, for example, your queue manager has to cope with new applications using new queues.

To add a new page set, use the following procedure:

1. Define and format the new page set. You can use the sample JCL in thlqual.SCSQPROC(CSQ4PAGE) as a basis. For more information, see Formatting page sets (FORMAT).

   Take care not to format any page sets that are in use, unless this is what you intend. If so, use the FORCE option of the FORMAT utility function.
2. Use the DEFINE PSID command with the DSN option to associate the page set with a buffer pool.
3. Add the appropriate storage class definitions for your page set by issuing DEFINE STGCLASS commands.
4. Optionally, to document how your queue manager is configured:
   a. Add the new page set to the started task procedure for your queue manager.
   b. Add a definition for the new page set to your CSQINP1 initialization data set.
   c. Add a definition for the new storage class to your CSQ4INYR initialization data set member.

For details of the DEFINE PSID and DEFINE STGCLASS commands, see DEFINE PSID and DEFINE STGCLASS.

What to do when one of your page sets becomes full

You can find out about the utilization of page sets by using the IBM MQ command DISPLAY USAGE. For example, the command:

```
DISPLAY USAGE PSID(03)
```

displays the current state of the page set 03. This tells you how many free pages this page set has.

If you have defined secondary extents for your page sets, they are dynamically expanded each time they fill up. Eventually, all secondary extents are used, or no further disk space is available. If this happens, an application receives the return code MQRC_STORAGE_MEDIUM_FULL.

If an application receives a return code of MQRC_STORAGE_MEDIUM_FULL from an MQI call, this is a clear indication that there is not enough space remaining on the page set. If the problem persists or is likely to recur, you must do something to solve it.

You can approach this problem in a number of ways:

- Balance the load between page sets by moving queues from one page set to another.
- Expand the page set. See “How to increase the size of a page set” on page 507 for instructions.
- Redefine the page set so that it can expand beyond 4 GB to a maximum size of 64 GB. See Defining a page set to be larger than 4 GB for instructions.

How to balance loads on page sets

Load balancing on page sets means moving the messages associated with one or more queues from one page set to another, less used, page set. Use this technique if it is not practical to expand the page set.

To identify which queues are using a page set, use the appropriate IBM MQ commands. For example, to find out which queues are mapped to page set 02, first, find out which storage classes map to page set 02, by using the command:

```
DISPLAY STGCLASS(*) PSID(02)
```

Then use the following command to find out which queues use which storage class:

```
DISPLAY QUEUE(*) TYPE(QLOCAL) STGCLASS
```

Moving a non-shared queue

To move queues and their messages from one page set to another, use the MQSC MOVE QLOCAL command (described in MOVE QLOCAL). When you have identified the queue or queues that you want to move to a new page set, follow this procedure for each of these queues:

1. Ensure that the queue you want to move is not in use by any applications (that is, IPPROCS and OPPROCS values from the DISPLAY QSTATUS command are zero) and that it has no uncommitted messages (the UNCOM value from the DISPLAY QSTATUS command is NO).

   Note: The only way to ensure that this state continues is to change the security authorization of the queue temporarily. See Profiles for queue security for more information.

   If you cannot do this, later stages in this procedure might fail if applications start to use the queue despite precautionary steps such as setting PUT(DISABLED). However, messages can never be lost by this procedure.
2. Prevent applications from putting messages on the queue being moved by altering the queue definition to disable MQPUT s. Change the queue definition to PUT(DISABLED).
3. Define a temporary queue with the same attributes as the queue that is being moved, using the command:

   ```
   DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
   ```

   Note: If this temporary queue already exists from a previous run, delete it before doing the define.
4. Move the messages to the temporary queue using the following command:

   ```
   MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
   ```
5. Delete the queue you are moving, using the command:

   ```
   DELETE QLOCAL(QUEUE_TO_MOVE)
   ```
6. Define a new storage class that maps to the required page set, for example:

   ```
   DEFINE STGCLASS(NEW) PSID(nn)
   ```

   Add the new storage class definition to the CSQINP2 data sets ready for the next queue manager restart.
7. Redefine the queue that you are moving, by changing the storage class attribute:

   ```
   DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW)
   ```

   When the queue is redefined, it is based on the temporary queue created in step “3” on page 506.
8. Move the messages back to the new queue, using the command:

   ```
   MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
   ```
9. The queue created in step “3” on page 506 is no longer required. Use the following command to delete it:

```
DELETE QL(TEMP_QUEUE)
```

10. If the queue being moved was defined in the CSQINP2 data sets, change the STGCLASS attribute of the appropriate DEFINE QLOCAL command in the CSQINP2 data sets. Add the REPLACE keyword so that the existing queue definition is replaced.

Figure 29 on page 507 shows an extract from a load balancing job.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE STGCLASS(NEW) PSID(2)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 29. Extract from a load balancing job for a page set

How to increase the size of a page set

You can initially allocate a page set larger than 4 GB, See Defining a page set to be larger than 4 GB

A page set can be defined to be automatically expanded as it becomes full by specifying EXPAND(SYSTEM) or EXPAND(USER). If your page set was defined with EXPAND(NONE), you can expand it in either of two ways:

- Alter its definition to allow automatic expansion. See Altering a page set to allow automatic expansion
- Create a new, larger page set and copy the messages from the old page set to the new one. See Moving messages to a new, larger page set

Defining a page set to be larger than 4 GB

IBM MQ can use a page set up to 64 GB in size, provided the data set is defined with 'extended addressability' to VSAM. Extended addressability is an attribute which is conferred by an SMS data class.

Note: Page sets and active log data sets are eligible to reside in the extended addressing space (EAS) part of an extended address volumes (EAV) and, from z/OS V1.12, an archive log dataset can also reside in the EAS.

In the example shown in the following sample JCL, the management class 'EXTENDED' is defined to SMS with 'Extended addressability'. If your existing page set is not currently defined as having extended addressability, use the following method to migrate to an extended addressability format data set.

1. Stop the queue manager.
2. Use Access Method Services to rename the existing page set.
3. Define a destination page set, the same size as the existing page set, but with DATACLAS(EXTENDED).

Note: Extended-format data sets must be SMS managed. These are the mechanisms for requesting extended format for VSAM data sets:

- Using a data class that has a DSNTYPE value of EXT and the subparameter R or P to indicate required or preferred.
- Coding DSNTYPE=EXTREQ (extended format is required) or DSNTYPE=EXTPREF (extended format is preferred) on the DD statement.
- Coding the LIKE= parameter on the DD statement to refer to an existing extended format data set.

For more information, see Restrictions on Defining Extended-Format Data Sets.

4. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. See Expanding a page set (COPYPAGE) for more details.
5. Restart the queue manager.
6. Alter the page set to use system expansion, to allow it to continue growing beyond its current allocation.

The following JCL shows example Access Method Services commands:

```
//S1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
ALTER 'VICY.CSQ1.PAGE01' -
NEWNAME('VICY.CSQ1.PAGE01.OLD')
ALTER 'VICY.CSQ1.PAGE01.DATA' -
NEWNAME('VICY.CSQ1.PAGE01.DATA.OLD')
DEFINE CLUSTER (NAME('VICY.CSQ1.PAGE01') -
MODEL('VICY.CSQ1.PAGE01.OLD') -
DATACLAS(EXTENDED))
/*
```

Altering a page set to allow automatic expansion

Use the ALTER PSID command with the EXPAND(USER) or EXPAND(SYSTEM) options. See ALTER PSID and Expanding a page set (COPYPAGE) for general information on expanding page sets.

Moving messages to a new, larger page set

This technique involves stopping and restarting the queue manager. This deletes any nonpersistent messages that are not on shared queues at restart time. If you have nonpersistent messages that you do not want to be deleted, use load balancing instead. For more details, see "How to balance loads on page sets" on page 505. In this description, the page set that you want to expand is referred to as the *source* page set; the new, larger page set is referred to as the *destination* page set.

Follow these steps:

1. Stop the queue manager.
2. Define the destination page set, ensuring that it is larger than the source page set, with a larger secondary extent value.
3. Use the FORMAT function of CSQUTIL to format the destination page set. See Formatting page sets (FORMAT) for more details.
4. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. See Expanding a page set (COPYPAGE) for more details.
5. Restart the queue manager using the destination page set by doing one of the following:
   - Change the queue manager started task procedure to reference the destination page set.
   - Use Access Method Services to delete the source page set and then rename the destination page set, giving it the same name as that of the source page set.

Attention:

Before you delete any IBM MQ page set, be sure that you have made the required backup copies.

How to reduce a page set

Prevent all users, other than the IBM MQ administrator, from using the queue manager. For example; by changing the access security settings.

If you have a large page set that is mostly empty (as shown by the DISPLAY USAGE command), you might want to reduce its size. The procedure to do this involves using the COPY, FORMAT, and LOAD functions of CSQUTIL (see IBM MQ utility program). This procedure does not work for page set zero (0), as it is not practical to reduce the size of this page set; the only way to do so is by reinitializing your queue manager (see “Reinitializing a queue manager” on page 532). The prerequisite of this procedure is to try and remove all users from the system so that all UOWs are complete and the page sets are consistent.

1. Use the STOP QMGR command with the QUIESCE or FORCE attribute to stop the queue manager.
2. Run the SCOPY function of CSQUTIL with the PSID option, to copy all message data from the large page set and save them in a sequential data set.
3. Define a new smaller page set data set to replace the large page set.
4. Run the FORMAT TYPE(NEW) function of CSQUTIL against the page set that you created in step “3” on page 509.
5. Restart the queue manager using the page set created in step “3” on page 509.
6. Run the LOAD function of CSQUTIL to load back all the messages saved during step “2” on page 509.
7. Allow all users access to the queue manager.
8. Delete the old large page set.

How to reintroduce a page set

In certain scenarios it is useful to be able to bring an old page set online again to the queue manager. Unless specific action is taken, when the old page set is brought online the queue manager will recognize that the page set recovery RBA stored in the page set itself and in the checkpoint records is old, and will therefore automatically start media recovery of the page set to bring it up to date.

Such media recovery can only be performed at queue manager restart, and is likely to take a considerable length of time, especially if archive logs held on tape must be read. However, normally in this circumstance, the page set has been offline for the intervening period and so the log contains no information pertinent to the page set recovery.

The following three choices are available:

Allow full media recovery to be performed.

1. Stop the queue manager.
2. Ensure definitions are available for the page set in both the started task procedure for the queue manager and in the CSQINP1 initialization data set.
3. Restart the queue manager.

Allow any messages on the page set to be destroyed.

This choice is useful where a page set has been offline for a long time (some months, for example) and it has now been decided to reuse it for a different purpose.

1. Format the page set using the FORMAT function of CSQUTIL with the TYPE(NEW) option.
2. Add definitions for the page set to both the started task procedure for the queue manager and the CSQINP1 initialization data set.
3. Restart the queue manager.

Using the TYPE(NEW) option for formatting clears the current contents of the page set and tells the queue manager to ignore any historical information in the checkpoint about the page set.

Bring the page set online avoiding the media recovery process.

Use this technique only if you are sure that the page set has been offline since a clean shutdown of the queue manager. This choice is most appropriate where the page set has been offline for a short period, typically due to operational issues such as a backup running while the queue manager is being started.

1. Format the page set using the FORMAT function of CSQUTIL with the TYPE(REPLACE) option.
2. Either add the page set back into the queue manager dynamically using the DEFINE PSID command with the DSN option or allow it to be added at a queue manager restart.

Using the TYPE(REPLACE) option for formatting checks that the page set was cleanly closed by the queue manager, and marks it so that media recovery will not be performed. No other changes are made to the contents of the page set.

z/OS How to back up and recover page sets

There are different mechanisms available for back up and recovery. Use this topic to understand these mechanisms.

This section describes the following topics:

- "Creating a point of recovery for non-shared resources" on page 510
- "Backing up page sets" on page 511
- "Recovering page sets" on page 512
- How to delete page sets

For information about how to create a point of recovery for shared resources, see "Recovering shared queues" on page 519.

Creating a point of recovery for non-shared resources

IBM MQ can recover objects and non-shared persistent messages to their current state if both:

1. Copies of page sets from an earlier point exist.
2. All the IBM MQ logs are available to perform recovery from that point.

These represent a point of recovery for non-shared resources.

Both objects and messages are held on page sets. Multiple objects and messages from different queues can exist on the same page set. For recovery purposes, objects and messages cannot be backed up in isolation, so a page set must be backed up as a whole to ensure the correct recovery of the data.

The IBM MQ recovery log contains a record of all persistent messages and changes made to objects. If IBM MQ fails (for example, due to an I/O error on a page set), you can recover the page set by restoring the backup copy and restarting the queue manager. IBM MQ applies the log changes to the page set from the point of the backup copy.

There are two ways of creating a point of recovery:

Full backup

Stop the queue manager, which forces all updates on to the page sets.

This allows you to restart from the point of recovery, using only the backed up page set data sets and the logs from that point on.

Fuzzy backup

Take *fuzzy* backup copies of the page sets without stopping the queue manager.

If you use this method, and your associated logs later become damaged or lost, you cannot to use the fuzzy page set backup copies to recover. This is because the fuzzy page set backup copies contain an inconsistent view of the state of the queue manager and are dependent on the logs being available. If

the logs are not available, you need to return to the last set of backup page set copies taken while the subsystem was inactive (Method 1) and accept the loss of data from that time.

Method 1: Full backup

This method involves shutting the queue manager down. This forces all updates on to the page sets so that the page sets are in a consistent state.

1. Stop all the IBM MQ applications that are using the queue manager (allowing them to complete first). This can be done by changing the access security or queue settings, for example.
2. When all activity has completed, display and resolve any in-doubt units of recovery. (Use the commands DISPLAY CONN and RESOLVE INDOUBT, as described in DISPLAY CONN and RESOLVE INDOUBT.)

   This brings the page sets to a consistent state; if you do not do this, your page sets might not be consistent, and you are effectively doing a fuzzy backup.
3. Issue the ARCHIVE LOG command to ensure that the latest log data is written out to the log data sets.
4. Issue the STOP QMGR MODE(QUIESCE) command. Record the lowest RBA value in the CSQI024I or CSQI025I messages (see CSQI024I and CSQI025I for more information). You should keep the log data sets starting from the one indicated by the RBA value up to the current log data set.
5. Take backup copies of all the queue manager page sets (see “Backing up page sets” on page 511).

Method 2: Fuzzy backup

This method does not involve shutting the queue manager down. Therefore, updates might be in virtual storage buffers during the backup process. This means that the page sets are not in a consistent state, and can only be used for recovery with the logs.

1. Issue the DISPLAY USAGE TYPE(ALL) command, and record the RBA value in the CSQI024I or CSQI025I messages (see CSQI024I and CSQI025I for more information).
2. Take backup copies of the page sets (see “Backing up page sets” on page 511).
3. Issue the ARCHIVE LOG command, to ensure that the latest log data is written out to the log data sets. To restart from the point of recovery, you must keep the log data sets starting from the log data set indicated by the RBA value up to the current log data set.

Backing up page sets

To recover a page set, IBM MQ needs to know how far back in the log to go. IBM MQ maintains a log RBA number in page zero of each page set, called the *recovery log sequence number* (LSN). This number is the starting RBA in the log from which IBM MQ can recover the page set. When you back up a page set, this number is also copied.

If the copy is later used to recover the page set, IBM MQ must have access to all the log records from this RBA value to the current RBA. That means you must keep enough of the log records to enable IBM MQ to recover from the oldest backup copy of a page set you intend to keep.

Use ADRDSSU COPY function to copy the page sets.

For more information, see the COPY DATASET Command Syntax for Logical Data Set documentation .

For example:

```
//STEP2 EXEC PGM=ADRDSSU,REGION=6M
//SYSPRINT DD SYSOUT=H
//SYSIN DD *
 COPY -
 DATASET(INCLUDE(SCENDATA.MQPA.PAGESET.*)) -
 RENAMEU(SCENDATA.MQPA.PAGESET.**,SCENDATA.MQPA.BACKUP1.**) -
 SPHERE -
 REPUNC -
 FASTREPLICATION(PREF )-
 CANCELERROR -
```

```
 TOL(ENQF)
/*
//
```

If you copy the page set while the queue manager is running you must use a copy utility that copies page zero of the page set first. If you do not do this you could corrupt the data in your page set.

If the process of dynamically expanding a page set is interrupted, for example by power to the system being lost, you can still use ADRDSSU to take a backup of a page set.

If you perform an Access Method Services IDCAMS `LISTCAT ENT('page set data set name') ALLOC`, you will see that the `HI-ALLOC-RBA` is higher than the `HI-USED-RBA`.

The next time this page set fills up it is extended again, if possible, and the pages between the high used RBA and the highest allocated RBA are used, along with another new extent.

Backing up your object definitions

You should also back up copies of your object definitions. To do this, use the MAKEDEF feature of the CSQUTIL COMMAND function (described in Issuing commands to IBM MQ (COMMAND)).

Back up your object definitions whenever you take a backup copy of your queue manager, and keep the most current version.

Recovering page sets

If the queue manager has terminated due to a failure, the queue manager can normally be restarted with all recovery being performed during restart. However, such recovery is not possible if any of your page sets or log data sets are not available. The extent to which you can now recover depends on the availability of backup copies of page sets and log data sets.

To restart from a point of recovery you must have:

- A backup copy of the page set that is to be recovered.
- If you used the "fuzzy" backup process described in “Method 2: Fuzzy backup” on page 511, the log data set that included the recorded RBA value, the log data set that was made by the ARCHIVE LOG command, and all the log data sets between these.
- If you used full backup, but you do not have the log data sets following that made by the ARCHIVE LOG command, you do **not** need to run the FORMAT TYPE(REPLACE) function of the CSQUTIL utility against all your page sets.

To recover a page set to its current state, you must also have all the log data sets and records since the ARCHIVE LOG command.

There are two methods for recovering a page set. To use either method, the queue manager must be stopped.

Simple recovery

This is the simpler method, and is appropriate for most recovery situations.

1. Delete the page set you want to restore from backup.
2. Use the ADRDSSU COPY function to recover your page set from the backup copy..

   Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.
3. Restart the queue manager.
4. When the queue manager has restarted successfully, you can restart your applications
5. Reinstate your normal backup procedures for the restored page.

Advanced recovery

This method provides performance advantages if you have a large page set to recover, or if there has been much activity on the page set since the last backup copy was taken. However, it requires more manual intervention than the simple method, which might increase the risk of error and the time taken to perform the recovery.

1. Delete and redefine the page set you want to restore from backup.
2. Use ADRDSSU to copy the backup copy of the page set into the new page set. Define your new page set with a secondary extent value so that it can be expanded dynamically.

   Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.
3. Change the CSQINP1 definitions for your queue manager to make the buffer pool associated with the page set being recovered as large as possible. By making the buffer pool large, you might be able to keep all the changed pages resident in the buffer pool and reduce the amount of I/O to the page set.
4. Restart the queue manager.
5. When the queue manager has restarted successfully, stop it (using quiesce) and then restart it using the normal buffer pool definition for that page set. After this second restart completes successfully, you can restart your applications
6. Reinstate your normal backup procedures for the restored page.

What happens when the queue manager is restarted

When the queue manager is restarted, it applies all changes made to the page set that are registered in the log, beginning at the restart point for the page set. IBM MQ can recover multiple page sets in this way. The page set is dynamically expanded, if required, during media recovery.

During restart, IBM MQ determines the log RBA to start from by taking the lowest value from the following:

- Recovery LSN from the checkpoint log record for each page set.
- Recovery LSN from page zero in each page set.
- The RBA of the oldest incomplete unit of recovery in the system at the time the backup was taken.

All object definitions are stored on page set zero. Messages can be stored on any available page set.

Note: The queue manager cannot restart if page set zero is not available.

How to delete page sets

You delete a page set by using the DELETE PSID command; see DELETE PSID for details of this command.

You cannot delete a page set that is still referenced by any storage class. Use DISPLAY STGCLASS to find out which storage classes reference a page set.

The data set is deallocated from IBM MQ but is not deleted. It remains available for future use, or can be deleted using z/OS facilities.

Remove the page set from the started task procedure for your queue manager.

Remove the definition of the page set from your CSQINP1 initialization data set.

z/OS How to back up and restore queues using CSQUTIL

Use this topic as a reference for further information about back up and restore using CSQUTIL.

You can use the CSQUTIL utility functions for backing up and restoring queues. To back up a queue, use the COPY or SCOPY function to copy the messages from a queue onto a data set. To restore the queue, use the complementary function LOAD or SLOAD. For more information, see IBM MQ utility program.

z/OS Managing buffer pools

Use this topic if you want to change or delete your buffer pools.

This topic describes how to alter and delete buffer pools. It contains these sections:

- "How to change the number of buffers in a buffer pool" on page 514
- "How to delete a buffer pool" on page 514

Buffer pools are defined during queue manager initialization, using DEFINE BUFFPOOL commands issued from the initialization input data set CSQINP1. Their attributes can be altered in response to business requirements while the queue manager is running, using the processes detailed in this topic. The queue manager records the current buffer pool attributes in checkpoint log records. These are automatically restored on subsequent queue manager restart, unless the buffer pool definition in CSQINP1 includes the REPLACE attribute.

Use the DISPLAY USAGE command to display the current buffer attributes.

You can also define buffer pools dynamically using the DEFINE PSID command with the DSN option.

If you change buffer pools dynamically, you should also update their definitions in the initialization data set CSQINP1.

See Planning on z/OS for a description of page sets, storage classes, buffers, and buffer pools, and some of the performance considerations that apply.

Note: Buffer pools use significant storage. When you increase the size of a buffer pool or define a new buffer pool ensure that sufficient storage is available. For more information, see Address space storage.

How to change the number of buffers in a buffer pool

If a buffer pool is too small, the condition can result in message CSQP020E on the console, you can allocate more buffers to it using the ALTER BUFFPOOL command as follows:

1. Determine how much space is available for new buffers by looking at the CSQY220I messages in the log. The available space is reported in MB. As a buffer has a size of 4 KB, each MB of available space allows you to allocate 256 buffers. Do not allocate all the free space to buffers, as some is required for other tasks.

   If the buffer pool uses fixed 4 KB pages, that is, its PAGECLAS attribute is FIXED4KB, ensure that there is sufficient real storage available on the LPAR.
2. If the reported free space is inadequate, release some buffers from another buffer pool using the command

   ```
   ALTER BUFFPOOL(buf-pool-id) BUFFERS(integer)
   ```

   where *buf-pool-id* is the buffer pool from which you want to reclaim space and *integer* is the new number of buffers to be allocated to this buffer pool, which must be smaller than the original number of buffers allocated to it.
3. Add buffers to the buffer pool you want to expand using the command

   ```
   ALTER BUFFPOOL(buf-pool-id) BUFFERS(integer)
   ```

   where *buf-pool-id* is the buffer pool to be expanded and *integer* is the new number of buffers to be allocated to this buffer pool, which must be larger than the original number of buffers allocated to it.

How to delete a buffer pool

When a buffer pool is no longer used by any page sets, delete it to release the virtual storage allocated to it.

You delete a buffer pool using the DELETE BUFFPOOL command. The command fails if any page sets are using this buffer pool.

See “How to delete page sets” on page 513 for information about how to delete page sets.

z/OS *Changing the buffer pools location value*

Use this topic if you want to change your buffer pools **Location** value to above or below the virtual line called “the bar”. In a 64-bit address space, the bar marks the 2 GB address. The bar separates storage below the 2 GB address, called “below the bar”, from storage above the 2 GB address, called “above the bar”. The storage below the bar uses 31-bit addressability and the storage above the bar uses 64-bit addressability.

Attention: Deprecated From IBM MQ 9.1, **LOCATION**(BELOW) is deprecated and you should use **LOCATION**(ABOVE) only.

How to change the Location value of a buffer pool

1. You can use the **ALTER BUFFPOOL** command to dynamically change the **LOCATION** of the buffer pool to ABOVE. By default, the **LOCATION** value is set to BELOW. You can increase the number of buffers, depending on your system requirements. Specify the correct size and location value to maintain the queue manager load.
2. Before applying the changes, make sure that enough storage is available for the new buffer pool configuration. For **LOCATION**(ABOVE), you might need to adjust the amount of available 64-bit storage by tuning the **MEMLIMIT** parameter. For more information, see Storage configuration and Queue manager storage configuration.

   Attention: This task should be done during periods of low activity, when the queue manager is not handling heavy workloads. Increasing the buffer size or changing the location can be intensive.
3. If you reduce the buffer pool size, recycle the queue manager immediately to clear any storage issues caused by the size change. If you fail to recycle the queue manager, the error `ABEND878-10 - Virtual private region depleted` may appear by the fragmented region storage.
4. You can also apply the buffer pool changes with the **DEFINE BUFFPOOL** command in the CSQINP1 concatenation initialization dataset, but you must temporarily use the **REPLACE** option. When the queue manager is active, it records current buffer pool attributes in checkpoint log records. These are automatically restored upon subsequent queue manager restarts unless the buffer pool definition in CSQINP1 includes the **REPLACE** attribute. Return to the **NOREPLACE** option after queue manager restart. If you change the buffer pool size later by using the **ALTER BUFFPOOL** command and **REPLACE** is still specified in CSQINP1, the buffer pool will revert to the CSQINP1 definition with the next restart.
5. If you need to apply the changes at queue manager startup, **ALTER BUFFPOOL** can be issued from the CSQINP2 concatenation. It cannot be issued from CSQINPT or the CSQINP1 DD concatenation. If you make changes dynamically to the buffer pool by using the **ALTER BUFFPOOL** command, it is a good idea to update the **DEFINE BUFFPOOL** command in the CSQINP1 file, without the **REPLACE** attribute, to reflect the final configuration. This configuration is used in the rare event of a cold start.

Changing the buffer pool **LOCATION** value to ABOVE the bar provides the following advantages:

- It relieves the 31-bit virtual storage constraints, providing more space for common storage areas.
- During a high workload, adding buffers to a buffer pool above the bar is easier and safer than moving a pool to above the bar after it’s already in use.

z/OS Managing queue sharing groups and shared queues on z/OS

IBM MQ can use different types of shared resources, for example queue sharing groups, shared queues, and the coupling facility. Use this topic to review the procedures needed to manage these shared resources.

This section contains information about the following topics:

- "Managing queue sharing groups" on page 516
- "Managing shared queues" on page 519
- "Managing group objects" on page 523
- "Managing the coupling facility" on page 523

z/OS *Managing queue sharing groups*

You can add or remove a queue manager to a queue sharing group (QSG), and manage the associated Db2 tables.

This topic has sections about the following tasks:

- "Setting up a queue sharing group" on page 516
- "Adding a queue manager to a queue sharing group" on page 517
- "Removing a queue manager from a queue sharing group" on page 518
- "Removing a queue sharing group from the Db2 tables" on page 518
- "Validating the consistency of Db2 definitions" on page 519

Setting up a queue sharing group

Each queue sharing group has a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Follow these steps to set up a queue sharing group:

1. If this is the first queue sharing group to use the Db2 data-sharing group, set up the Db2 environment.
2. Set up the coupling facility.
3. Add the queue sharing group to the Db2 tables. Use the ADD QSG function of the queue sharing group utility (CSQ5PQSG). This program is described in The queue sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45AQS).
4. Add a queue manager to the queue sharing group by following the steps in "Adding a queue manager to a queue sharing group" on page 517
5. Define application structures to IBM MQ by following the steps in "Adding a coupling facility structure" on page 524.
6. If required, migrate non-shared queues to shared queues.
7. For availability, create shared channels into and out of the queue sharing group.
   - For connections into the queue sharing group:
     - Set up a VIPA socket or hardware router to distribute workload between the available queue managers in the QSG.
     - Define a receiver channel with QSGDISP(GROUP), to ensure the channel definition is available on all queue managers in the QSG.
     - Start a listener with INDISP(GROUP), on each queue manager, for MCA channel connections into the QSG. Client connections into the QSG should still connect to a listener started with INDISP(QMGR).
     - Change applications to connect using the QSG name, rather than a specific queue manager name.

  - Ensure that the channel authentication rules on all queue managers in the QSG are the same, to allow applications to connect to any queue manager in the QSG.
- For connections out of the queue sharing group:
  - Define a shared transmission queue.
  - Define the outbound channel with QSGDISP(GROUP) and DEFCDISP(SHARED).

  If you convert an existing channel to a shared channel, you might need to issue the RESET CHANNEL command before starting the channel as the synchronization queue used by the channel will have changed.

Adding a queue manager to a queue sharing group

A queue manager can be added to an existing queue sharing group.

Note that:

- The queue sharing group must exist before you can add queue managers to it.
- A queue manager can be a member of only one queue sharing group.

Follow these steps to add a queue manager to a queue sharing group:

1. Perform the tasks in implement ESM security controls for the queue sharing group to grant the appropriate access to the queue manager and channel initiator user IDs.
2. If the queue sharing group has CF structures configured to offload data to SMDS, perform the tasks in set up the SMDS environment.
3. Stop the queue manager.
4. Use the ADD QMGR function of the queue sharing group utility (CSQ5PQSG). This program is described in the queue sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45AQM).
5. Change your system parameter module to add queue sharing group data:
   a. Modify CSQ6SYSP to specify the QSGDATA parameter. See using CSQ6SYSP for more information.
   b. Assemble and link the system parameter module. You might want to use a different name for the load module.
   c. Change your startup process to use the new module.
6. Copy and tailor sample member thlqual.SCSQPROC(CSQ4INSS), which defines required CF structures and SYSTEM queues. Add the customized member to the CSQINP2 DD in the queue manager startup JCL.
7. Restart your queue manager using the queue sharing group system parameter module.
8. Optionally, migrate to security profiles prefixed by the queue sharing group name, instead of the queue manager name.
9. If shared channels are used for connections into the QSG, create channel authentication rules that mirror those on the other queue managers in the QSG, to allow applications to connect to any queue manager in the QSG.
10. 10. Optionally, do either of the following to allow applications connected to the queue manager in the QSG to put messages to queues hosted by other queue managers in the QSG:
    - Turn on intra-group queuing by issuing the command ALTER QMGR IGQ(ENABLED).
    - Define transmission queues and channels to the other queue managers in the QSG. Defining transmission queues with the same name as the target queue managers avoids the need to define remote queues and queue manager aliases.

Note: To add a queue manager to an existing queue sharing group containing queue managers running earlier versions of IBM MQ, you must first apply the coexistence PTF for the highest version of IBM MQ in the group to every earlier version queue manager in the group.

Removing a queue manager from a queue sharing group

You can only remove a queue manager from a queue sharing group if the queue manager's logs are not needed by another process, and all SMDS owned by the queue manager are empty.

See Deleting shared message data sets and DELETE CFSTRUCT for more information.

The logs are needed if they contain:

- The latest backup of one of the coupling facility (CF) application structures used by the queue sharing group
- Data needed by a future restore process, that is, the queue manager has used a recoverable structure since the time described by the last backup exclusion interval value.

If either or both of these points apply, or an SMDS owned by the queue manager contains messages, the queue manager cannot be removed. To determine which queue managers' logs are needed for a future restore process, use the MQSC DISPLAY CFSTATUS command with the TYPE(BACKUP) option (for details of this command, see DISPLAY CFSTATUS).

Use the following steps to remove a queue manager from a queue sharing group:

1. Stop any applications connected to the queue manager that put messages to shared queues.
2. Resolve any indoubt units of work involving this queue manager.
3. Determine if there are any messages in any SMDS owned by the queue manager by issuing the command DISPLAY USAGE TYPE(SMDS).
4. If there are offloaded messages for any application structure, wait until those messages have been retrieved from the queue. The number of offloaded messages reported by DISPLAY USAGE TYPE(SMDS) should be zero before proceeding.
5. Shut the queue manager down cleanly using STOP QMGR MODE(QUIESCE).
6. Wait for an interval at least equivalent to the value of the EXCLINT parameter you will specify in the BACKUP CFSTRUCT command in the next step.
7. On another queue manager, run a CF structure backup for each recoverable CF structure by using the MQSC BACKUP CFSTRUCT command and specifying an EXCLINT value as required in the previous step.
8. Confirm that the queue manager's logs are not needed to restore any CF structures, by inspecting the output from the command DISPLAY CFSTATUS(*) TYPE(BACKUP).
9. Use the REMOVE QMGR function of the CSQ5PQSG utility to remove the queue manager from the queue sharing group. This program is described in The queue sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45RQM).
10. Before restarting the queue manager, reset the QSGDATA system parameter to its default value, and recreate the system parameter module. See Using CSQ6SYSP for information about how to tailor your system parameters.

Note, that when removing the last queue manager in a queue sharing group, you must use the FORCE option, rather than REMOVE. This removes the queue manager from the queue sharing group, while not performing the consistency checks of the queue manager logs being required for recovery. You should only perform this operation if you are deleting the queue sharing group.

Removing a queue sharing group from the Db2 tables

To remove a queue sharing group from the Db2 tables, use the REMOVE QSG function of the queue sharing group utility (CSQ5PQSG). This program is described in The queue sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45RQS).

You can only remove a queue sharing group from the common Db2 data-sharing group tables after you have removed all the queue managers from the queue sharing group (as described in “Removing a queue manager from a queue sharing group” on page 518).

When the queue sharing group record is deleted from the queue sharing group administration table, all objects and administrative information relating to that queue sharing group are deleted from other IBM MQ Db2 tables. This includes shared queue and group object information.

Validating the consistency of Db2 definitions

Problems for shared queues within a queue sharing group can occur if the Db2 object definitions have, for any reason, become inconsistent.

To validate the consistency of the Db2 object definitions for queue managers, CF structures, and shared queues, use the VERIFY QSG function of the queue sharing group utility (CSQ5PQSG). This program is described in The queue sharing group utility.

z/OS *Managing shared queues*

Use this topic to understand how to recover, move, and migrate shared queues.

This section describes the following tasks:

- “Recovering shared queues” on page 519
- “Moving shared queues” on page 520
- “Migrating non-shared queues to shared queues” on page 522
- Suspending a Db2 connection

Recovering shared queues

IBM MQ can recover persistent messages on shared queues if all:

- Backups of the CF structures containing the messages have been performed.
- All the logs for all queue managers in the queue sharing group are available, to perform recovery from the point the backups are taken.
- Db2 is available and the structure backup table is more recent that the most recent CF structure backup.

The messages on a shared queue are stored in a coupling facility (CF) structure. Persistent messages can be put onto shared queues, and like persistent messages on non-shared queues, they are copied to the queue manager log. The MQSC BACKUP CFSTRUCT and RECOVER CFSTRUCT commands are provided to allow the recovery of a CF structure in the unlikely event of a coupling facility failure. In such circumstances, any nonpersistent messages stored in the affected structure are lost, but persistent messages can be recovered. Any further application activity using the structure is prevented until the structure has been recovered.

To enable recovery, you must back up your coupling facility list structures frequently using the MQSC BACKUP CFSTRUCT command. The messages in the CF structure are written onto the active log data set of the queue manager making the backup. It writes a record of the backup to Db2: the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager log, and the backup time. Back up CF list structures even if you are not actively using shared queues, for example, if you have set up a queue sharing group intending to use it in the future.

You can recover a CF structure by issuing an MQSC RECOVER CFSTRUCT command to the queue manager that can perform the recovery; you can use any queue manager in the queue sharing group. You can specify a single CF structure to be recovered, or you can recover several CF structures simultaneously.

As noted previously, it is important that you back up your CF list structures frequently, otherwise recovering a CF structure can take a long time. Moreover, the recovery process cannot be canceled.

The definition of a shared queue is kept in a Db2 database and can therefore be recovered if necessary using standard Db2 database procedures. See Shared queues and queue sharing groups for more information.

Moving shared queues

This section describes how to perform load balancing by moving a shared queue from one coupling facility structure to another. It also describes how to move a non-shared queue to a shared queue, and how to move a shared queue to a non-shared queue.

When you move a queue, you need to define a temporary queue as part of the procedure. This is because every queue must have a unique name, so you cannot have two queues of the same name, even if the queues have different queue dispositions. IBM MQ tolerates having two queues with the same name (as in step "2" on page 520), but you cannot use the queues.

- Moving a queue from one coupling facility structure to another
- Moving a non-shared queue to a shared queue
- Moving a shared queue to a non-shared queue

Moving a queue from one coupling facility structure to another

To move queues and their messages from one CF structure to another, use the MQSC MOVE QLOCAL command. When you have identified the queue or queues that you want to move to a new CF structure, use the following procedure to move each queue:

1. Ensure that the queue you want to move is not in use by any applications, that is, the queue attributes IPPROCS and OPPROCS are zero on all queue managers in the queue sharing group.
2. Prevent applications from putting messages on the queue being moved by altering the queue definition to disable MQPUT s. Change the queue definition to PUT(DISABLED).
3. Define a temporary queue with the same attributes as the queue that is being moved using the following command:

```
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
```

   Note: If this temporary queue exists from a previous run, delete it before doing the define.
4. Move the messages to the temporary queue using the following command:

```
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
```

5. Delete the queue you are moving, using the command:

```
DELETE QLOCAL(QUEUE_TO_MOVE)
```

6. Redefine the queue that is being moved, changing the CFSTRUCT attribute, using the following command:

```
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
```

   When the queue is redefined, it is based on the temporary queue created in step "3" on page 520.
7. Move the messages back to the new queue using the command:

```
MOVE QLOCAL(TEMP) TOQLOCAL(QUEUE_TO_MOVE)
```

8. The queue created in step "3" on page 520 is no longer required. Use the following command to delete it:

```
DELETE QL(TEMP_QUEUE)
```

9. If the queue being moved was defined in the CSQINP2 data sets, change the CFSTRUCT attribute of the appropriate DEFINE QLOCAL command in the CSQINP2 data sets. Add the REPLACE keyword so that the existing queue definition is replaced.

Figure 30 on page 521 shows a sample job for moving a queue from one CF structure to another.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 30. Sample job for moving a queue from one CF structure to another

Moving a non-shared queue to a shared queue

The procedure for moving a non-shared queue to a shared queue is like the procedure for moving a queue from one CF structure to another (see “Moving a queue from one coupling facility structure to another” on page 520). Figure 31 on page 521 gives a sample job to do this.

Note: Remember that messages on shared queues are subject to certain restrictions on the maximum message size, message persistence, and queue index type, so you might not be able to move some non-shared queues to a shared queue.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 31. Sample job for moving a non-shared queue to a shared queue

Moving a shared queue to a non-shared queue

The procedure for moving a shared queue to a non-shared queue is like the procedure for moving a queue from one CF structure to another (see “Moving a queue from one coupling facility structure to another” on page 520).

Figure 32 on page 522 gives a sample job to do this.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW) QSGDISP(QMGR)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 32. Sample job for moving a shared queue to a non-shared queue

Migrating non-shared queues to shared queues

There are two stages to migrating non-shared queues to shared queues:

- Migrating the first (or only) queue manager in the queue sharing group
- Migrating any other queue managers in the queue sharing group

Migrating the first (or only) queue manager in the queue sharing group

Figure 31 on page 521 shows an example job for moving a non-shared queue to a shared queue. Do this for each queue that needs migrating.

Note:

1. Messages on shared queues are subject to certain restrictions on the maximum message size, message persistence, and queue index type, so you might not be able to move some non-shared queues to a shared queue.
2. You must use the correct index type for shared queues. If you migrate a transmission queue to be a shared queue, the index type must be MSGID.

If the queue is empty, or you do not need to keep the messages that are on it, migrating the queue is simpler. Figure 33 on page 522 shows an example job to use in these circumstances.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 33. Sample job for moving a non-shared queue without messages to a shared queue

Migrating any other queue managers in the queue sharing group

1. For each queue that does not have the same name as an existing shared queue, move the queue as described in Figure 31 on page 521 or Figure 33 on page 522.
2. For queues that have the same name as an existing shared queue, move the messages to the shared queue using the commands shown in Figure 34 on page 523.

```
MOVE QLOCAL(QUEUE_TO_MOVE) QSGDISP(QMGR) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QLOCAL(QUEUE_TO_MOVE) QSGDISP(QMGR)
```

Figure 34. Moving messages from a non-shared queue to an existing shared queue

Suspending a connection to Db2

If you want to apply maintenance or service to the Db2 tables or package related to shared queues without stopping your queue manager, you must temporarily disconnect queue managers in the data sharing group (DSG) from Db2.

To do this:

1. Use the MQSC command SUSPEND QMGR FACILITY(Db2).
2. Do the binds.
3. Reconnect to Db2 using the MQSC command RESUME QMGR FACILITY(Db2)

Note that there are restrictions on the use of these commands.

Attention: While the Db2 connection is suspended, the following operations will not be available. Therefore, you need to do this work during a time when your enterprise is at its least busy.

- Access to Shared queue objects for administration (define, delete,alter)
- Starting shared channels
- Storing messages in Db2
- Backup or recover CFSTRUCT

z/OS *Managing group objects*

Use this topic to understand how to work with group objects.

IBM MQ automatically copies the definition of a group object to page set zero of each queue manager that uses it. You can alter the copy of the definition temporarily, and IBM MQ allows you to refresh the page set copies from the repository copy. IBM MQ always tries to refresh the page set copies from the repository copy on start-up (for channel objects, this is done when the channel initiator restarts). This ensures that the page set copies reflect the version on the repository, including any changes that were made when the queue manager was inactive.

There are circumstances under which the refresh is not performed, for example:

- If a copy of the queue is open, a refresh that would change the usage of the queue fails.
- If a copy of a queue has messages on it, a refresh that would delete that queue fails.

In these circumstances, the refresh is not performed on that copy, but is performed on the copies on all other queue managers. Check for and correct any problems with copy objects after adding, changing, or deleting a group object, and at queue manager or channel initiator restart.

z/OS *Managing the coupling facility*

Use this topic to understand how to add or remove coupling facility (CF) structures.

This section describes the following tasks:

- “Adding a coupling facility structure” on page 524
- “Removing a coupling facility structure” on page 524

Adding a coupling facility structure

To add a coupling facility structure, use the following procedure:

1. Define the CF structure in the CFRM policy data set. The information about setting up the coupling facility in Set up the coupling facility describes the rules for naming coupling facility structures, and how to define structures in the CFRM policy data set.
2. If you want to configure the structure to offload message data to SMDS, allocate and preformat data sets. See creating a shared message data set for details.
3. Define the structure to IBM MQ using the DEFINE CFSTRUCT command.

Removing a coupling facility structure

To remove a coupling facility structure, use the following procedure:

1. Use the following command to get a list of all the queues using the coupling facility structure that you want to delete:

```
DISPLAY QUEUE(*) QSGDISP(SHARED) CFSTRUCT(structure-name)
```

2. Delete all the queues that use the structure.
3. Delete the CF structure from IBM MQ using the DELETE CFSTRUCT command.
4. If the structure was configured to offload message data to SMDS, delete the SMDS.
5. Remove the structure definition from your CFRM policy data set and run the IXCMIAPU utility. (This is the reverse of the customization task set up the coupling facility, described in Set up the coupling facility.)

z/OS *Tuning coupling facility list monitoring*

Use this topic to understand coupling facility list monitoring

Coupling facility (CF) list monitoring is used to monitor the state of list structures containing IBM MQ shared queues. When a message is added to a shared queue, and the queue's depth transitions from zero to non-zero, the CF notifies all queue managers in the queue sharing group. When notified the queue managers might perform a number of actions, including notifying trigger monitors that are using TRIGGER(FIRST), or applications which are performing a get-wait.

By default, the CF notifies all queue managers in the queue sharing group at the same time. In certain configurations this can cause problems, such as:

- Skewed workload distribution, where a large percentage of messages go to a particular queue manager in the queue sharing group, often the queue manager running on the fastest LPAR, or which is closest to the CF, or
- A large number of failed gets, resulting in wasted CPU time.

z/OS V2R3 introduces a new coupling facility resource manager (CFRM) attribute called **KEYRNOTIFYDELAY**, which can be used with list structures containing shared queues (that is, application structures, and not the admin structure), and which can, for certain workloads, minimize the effects of workload skewing and empty MQGET calls, or empty MQGET calls.

KEYRNOTIFYDELAY can only be set on structures in a CF, running at CFLEVEL 22 or higher.

Its value must be one to seven decimal digits, in a range from 0 to 1,000,000 microseconds. If set to a non-zero value and the depth of a queue transitions from zero to non-zero, the CF selects a single

queue manager from the queue sharing group, and notifies that queue manager before all the other queue managers in the group.

The queue manager is selected in a round-robin manner. If the selected queue manager does not process the message inside the time interval described by **KEYRNOTIFYDELAY** all the other queue managers in the queue sharing group will also be notified.

More information on **KEYRNOTIFYDELAY** is available here: Understanding Keyrange Monitoring Notification Delay.

Note that there are two similar CFRM attributes called **LISTNOTIFYDELAY** and **SUBNOTIFYDELAY**. Neither of these has any measurable effect on IBM MQ workload.

z/OS Recovery and restart on z/OS

Use this topic to understand the recovery and restart mechanisms used by IBM MQ.

z/OS Restarting IBM MQ

After a queue manager terminates there are different restart procedures needed depending on how the queue manager terminated. Use this topic to understand the different restart procedures that you can use.

This topic contains information about how to restart your queue manager in the following circumstances:

- “Restarting after a normal shutdown” on page 525
- “Restarting after an abnormal termination” on page 525
- “Restarting if you have lost your page sets” on page 526
- “Restarting if you have lost your log data sets” on page 526
- Restarting if you have lost your CF structures

Restarting after a normal shutdown

If the queue manager was stopped with the STOP QMGR command, the system finishes its work in an orderly way and takes a termination checkpoint before stopping. When you restart the queue manager, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

To restart the queue manager, issue the START QMGR command as described in “Using MQSC to start and stop a queue manager on z/OS” on page 451.

Restarting after an abnormal termination

IBM MQ automatically detects whether restart follows a normal shutdown or an abnormal termination.

Starting the queue manager after it has terminated abnormally is different from starting it after the STOP QMGR command has been issued. If the queue manager terminates abnormally, it terminates without being able to finish its work or take a termination checkpoint.

To restart the queue manager, issue the START QMGR command as described in “Using MQSC to start and stop a queue manager on z/OS” on page 451. When you restart a queue manager after an abnormal termination, it refreshes its knowledge of its status at termination using information in the log, and notifies you of the status of various tasks.

Normally, the restart process resolves all inconsistent states. But, in some cases, you must take specific steps to resolve inconsistencies. This is described in “Recovering units of work manually” on page 539.

Restarting if you have lost your page sets

If you have lost your page sets, you need to restore them from your backup copies before you can restart the queue manager. This is described in "How to back up and recover page sets" on page 510.

The queue manager might take a long time to restart under these circumstances because of the length of time needed for media recovery.

Restarting if you have lost your log data sets

If, after stopping a queue manager (using the STOP QMGR command), both copies of the log are lost or damaged, you can restart the queue manager providing you have a consistent set of page sets (produced using Method 1: Full backup).

Follow this procedure:

1. Define new page sets to correspond to each existing page set in your queue manager. See Task 15: Define your page sets for information about page set definition.

   Ensure that each new page set is larger than the corresponding source page set.
2. Use the FORMAT function of CSQUTIL to format the destination page set. See Formatting page sets for more details.
3. Use the RESETPAGE function of CSQUTIL to copy the existing page sets or reset them in place, and reset the log RBA in each page. See Copying a page set and resetting the log for more information about this function.
4. Redefine your queue manager log data sets and BSDS using CSQJU003 (see The change log inventory utility).
5. Restart the queue manager using the new page sets. To do this, you do one of the following:
   - Change the queue manager started task procedure to reference the new page sets. See Task 6: Create procedures for the IBM MQ queue manager for more information.
   - Use Access Method Services to delete the old page sets and then rename the new page sets, giving them the same names as the old page sets.

Attention: Before you delete any IBM MQ page set, ensure that you have made the required backup copies.

If the queue manager is a member of a queue sharing group, GROUP and SHARED object definitions are not normally affected by lost or damaged logs. However, if any shared-queue messages are involved in a unit of work that was covered by the lost or damaged logs, the effect on such uncommitted messages is unpredictable.

Note: If logs are damaged and the queue manager is a member of a queue sharing group, the ability to recover shared persistent messages might be lost. Issue a BACKUP CFSTRUCT command immediately on another active queue manager in the queue sharing group for all CF structures with the RECOVER(YES) attribute.

Restarting if you have lost your CF structures

You do not need to restart if you lose your CF structures, because the queue manager does not terminate.

z/OS *Alternative site recovery on z/OS*

You can recover a single queue manager or a queue sharing group, or consider disk mirroring.

See the following sections for more details:

- Recovering a single queue manager at an alternative site
- Recovering a queue sharing group.

  - CF structure media recovery
  - Backing up the queue sharing group at the prime site
  - Recovering a queue sharing group at the alternative site
- Using disk mirroring

Recovering a single queue manager at an alternative site

If a total loss of an IBM MQ computing center occurs, you can recover on another queue manager or queue sharing group at a recovery site. (See "Recovering a queue sharing group at the alternative site" on page 530 for the alternative site recovery procedure for a queue sharing group.)

To recover on another queue manager at a recovery site, you must regularly back up the page sets and the logs. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time, as possible.

At the recovery site:

- The recovery queue managers **must** have the same names as the lost queue managers.
- The system parameter module (for example, CSQZPARM) used on each recovery queue manager must contain the same parameters as the corresponding lost queue manager.

When you have done this, reestablish all your queue managers as described in the following procedure. This can be used to perform disaster recovery at the recovery site for a single queue manager. It assumes that all that is available are:

- Copies of the archive logs and BSDSs created by normal running at the primary site (the active logs will have been lost along with the queue manager at the primary site).
- Copies of the page sets from the queue manager at the primary site that are the same age or older than the most recent archive log copies available.

You can use dual logging for the active and archive logs, in which case you need to apply the BSDS updates to both copies:

1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the most recent archived BSDS into it.
4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and does not appear as an archived log. Record the STARTRBA and ENDRBA of this log.
5. Use the change log inventory utility, CSQJU003, to register this latest archive log data set in the BSDS that you have just restored, using the STARTRBA and ENDRBA recorded in Step "4" on page 527.
6. Use the DELETE option of CSQJU003 to remove all active log information from the BSDS.
7. Use the NEWLOG option of CSQJU003 to add active logs to the BSDS, do not specify STARTRBA or ENDRBA.
8. Use CSQJU003 to add a restart control record to the BSDS. Specify `CRESTART CREATE,ENDRBA=highrba`, where `highrba` is the high RBA of the most recent archive log available (found in Step "4" on page 527), plus 1.

   The BSDS now describes all active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.
9. Restart the queue manager with the **START QMGR** command. During initialization, an operator reply message such as the following is issued:

   ```
   CSQJ245D +CSQ1 RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.
   REPLY Y TO CONTINUE, N TO CANCEL
   ```

Type Y to start the queue manager. The queue manager starts, and recovers data up to ENDRBA specified in the CRESTART statement.

See IBM MQ utilities on z/OS reference for information about using CSQJU003 and CSQJU004.

The following example shows sample input statements for CSQJU003 for steps 6, 7, and 8:

```
* Step 6
DELETE DSNAME=MQM2.LOGCOPY1.DS01
DELETE DSNAME=MQM2.LOGCOPY1.DS02
DELETE DSNAME=MQM2.LOGCOPY1.DS03
DELETE DSNAME=MQM2.LOGCOPY1.DS04
DELETE DSNAME=MQM2.LOGCOPY2.DS01
DELETE DSNAME=MQM2.LOGCOPY2.DS02
DELETE DSNAME=MQM2.LOGCOPY2.DS03
DELETE DSNAME=MQM2.LOGCOPY2.DS04

* Step 7
NEWLOG DSNAME=MQM2.LOGCOPY1.DS01,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS02,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS03,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS04,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY2.DS01,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS02,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS03,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS04,COPY2

* Step 8
CRESTART CREATE,ENDRBA=063000
```

The things you need to consider for restarting the channel initiator at the recovery site are like those faced when using ARM to restart the channel initiator on a different z/OS image. See “Using ARM in an IBM MQ network” on page 536 for more information. Your recovery strategy should also cover recovery of the IBM MQ product libraries and the application programming environments that use IBM MQ (CICS , for example).

Other functions of the change log inventory utility (CSQJU003) can also be used in disaster recovery scenarios. The HIGHRBA function allows the update of the highest RBA written and highest RBA offloaded values within the bootstrap data set. The CHECKPT function allows the addition of new checkpoint queue records or the deletion of existing checkpoint queue records in the BSDS.

Attention: These functions might affect the integrity of your IBM MQ data. Only use them in disaster recovery scenarios under the guidance of IBM service personnel.

Fast copy techniques

If copies of all the page sets and logs are made while the queue manager is frozen, the copies will be a consistent set that can be used to restart the queue manager at an alternative site. They typically enable a much faster restart of the queue manager, as there is little media recovery to be performed.

Use the **SUSPEND QMGR LOG** command to freeze the queue manager. This command flushes buffer pools to the page sets, takes a checkpoint, and stops any further log write activity. Once log write activity has been suspended, the queue manager is effectively frozen until you issue a **RESUME QMGR LOG** command. While the queue manager is frozen, the page sets and logs can be copied.

By using copying tools such as FLASHCOPY or SNAPSHOT to rapidly copy the page sets and logs, the time during which the queue manager is frozen can be reduced to a minimum.

Within a queue sharing group, however, the **SUSPEND QMGR LOG** command might not be such a good solution. To be effective, the copies of the logs must all contain the same point in time for recovery, which means that the **SUSPEND QMGR LOG** command must be issued on all queue managers within the queue sharing group simultaneously, and therefore the entire queue sharing group will be frozen for some time.

Recovering a queue sharing group

In the event of a prime site disaster, you can restart a queue sharing group at a remote site using backup data sets from the prime site. To recover a queue sharing group you need to coordinate the recovery across all the queue managers in the queue sharing group, and coordinate with other resources, primarily Db2. This section describes these tasks in detail.

- CF structure media recovery
- Backing up the queue sharing group at the prime site
- Recovering a queue sharing group at the alternative site

CF structure media recovery

Media recovery of a CF structure used to hold persistent messages on a shared queue, relies on having a backup of the media that can be forward recovered by the application of logged updates. Take backups of your CF structures periodically using the MQSC **BACKUP CFSTRUCT** command. All updates to shared queues (MQGETs and MQPUTs) are written on the log of the queue manager where the update is performed. To perform media recovery of a CF structure you must apply logged updates to that backup from the logs of all the queue managers that have used that CF structure. When you use the MQSC **RECOVER CFSTRUCT** command, IBM MQ automatically merges the logs from relevant queue managers, and applies the updates to the most recent backup.

The CF structure backup is written to the log of the queue manager that processed the **BACKUP CFSTRUCT** command, so there are no additional data sets to be collected and transported to the alternative site.

Backing up the queue sharing group at the prime site

At the prime site you need to establish a consistent set of backups on a regular basis, which can be used in the event of a disaster to rebuild the queue sharing group at an alternative site. For a single queue manager, recovery can be to an arbitrary point in time, typically the end of the logs available at the remote site. However, where persistent messages have been stored on a shared queue, the logs of all the queue managers in the queue sharing group must be merged to recover shared queues, as any queue manager in the queue sharing group might have performed updates (MQPUT s or MQGET s) on the queue.

For recovery of a queue sharing group, you need to establish a point in time that is within the log range of the log data of all queue managers. However, as you can only **forward** recover media from the log, this point in time must be after the **BACKUP CFSTRUCT** command has been issued and after any page set backups have been performed. (Typically, the point in time for recovery might correspond to the end of a business day or week.)

The following diagram shows time lines for two queue managers in a queue sharing group. For each queue manager, fuzzy backups of page sets are taken (see Method 2: Fuzzy backup). On queue manager A, a **BACKUP CFSTRUCT** command is issued. Subsequently, an **ARCHIVE LOG** command is issued on each queue manager to truncate the active log, and copy it to media offline from the queue manager, which can be transported to the alternative site. End of log identifies the time at which the **ARCHIVE LOG** command was issued, and therefore marks the extent of log data typically available at the alternative site. The point in time for recovery must lie between the end of any page set or CF structure backups, and the earliest end of log available at the alternative site.

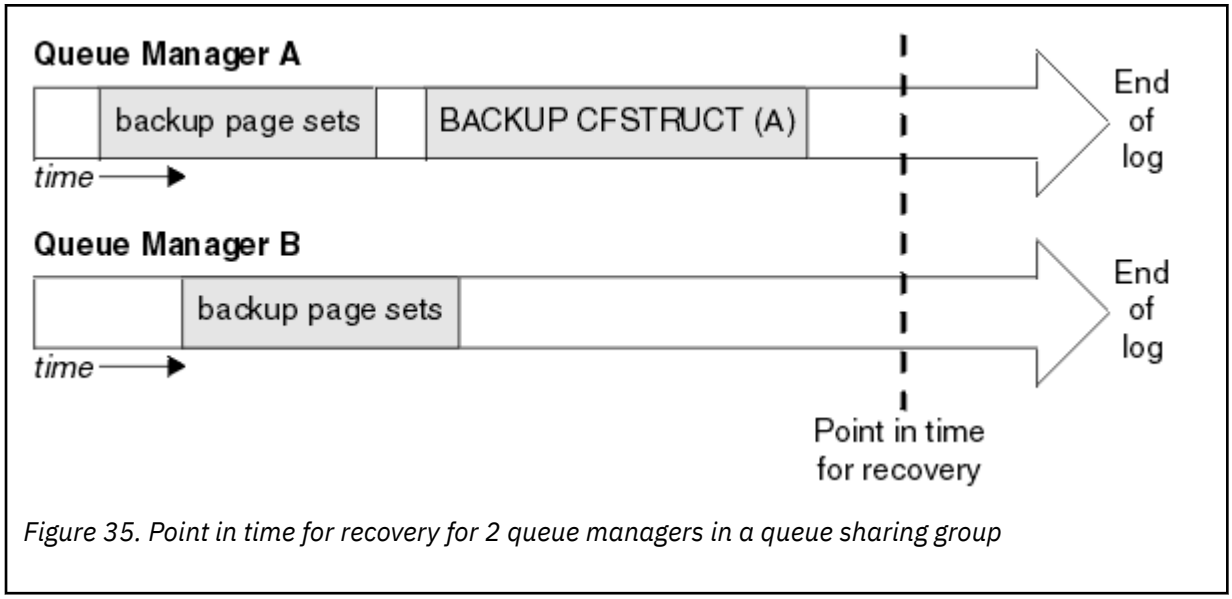


Figure 35. Point in time for recovery for 2 queue managers in a queue sharing group

IBM MQ records information associated with the CF structure backups in a table in Db2. Depending on your requirements, you might want to coordinate the point in time for recovery of IBM MQ with that for Db2, or it might be sufficient to take a copy of the IBM MQ CSQ.ADMIN_B_STRBACKUP table after the **BACKUP CFSTRUCT** commands have finished.

To prepare for a recovery:

1. Create page set backups for each queue manager in the queue sharing group.
2. Issue a **BACKUP CFSTRUCT** command for each CF structure with the RECOVER(YES) attribute. You can issue these commands from a single queue manager, or from different queue managers within the queue sharing group to balance the workload.
3. Once all the backups have completed, issue an **ARCHIVE LOG** command to switch the active log and create copies of the logs and BSDSs of each queue manager in the queue sharing group.
4. Transport the page set backups, the archived logs, the archived BSDS of all the queue managers in the queue sharing group, and your chosen Db2 backup information, off-site.

Recovering a queue sharing group at the alternative site

Before you can recover the queue sharing group, you need to prepare the environment:

1. If you have old information in your coupling facility from practice startups when you installed the queue sharing group, you need to clean this out first:

   Note: If you do not have old information in the coupling facility, you can omit this step.

   a. Enter the following z/OS command to display the CF structures for this queue sharing group:

   ```
   D XCF,STRUCTURE,STRNAME= qsgname
   ```

   b. For all structures that start with the queue sharing group name, use the z/OS command **SETXCF FORCE CONNECTION** to force the connection off those structures:

   ```
   SETXCF FORCE,CONNECTION,STRNAME= strname,CONNAME=ALL
   ```

   c. Delete all the CF structures using the following command for each structure:

   ```
   SETXCF FORCE,STRUCTURE,STRNAME= strname
   ```

2. Restore Db2 systems and data-sharing groups.
3. Recover the CSQ.ADMIN_B_STRBACKUP table so that it contains information about the most recent structure backups taken at the prime site.

Note: It is important that the STRBACKUP table contains the most recent structure backup information. Older structure backup information might require data sets that you have discarded as a result of the information given by a recent **DISPLAY USAGE TYPE(DATASET)** command, which would mean that your recovered CF structure would not contain accurate information.

4. If APAR PH64232 is not applied, V 10.0.0 or the queue manager is not running at IBM MQ 9.4.3 or later, run the **ADD QMGR** command of the CSQ5PQSG utility for every queue manager in the queue sharing group. This restores the XCF group entry for each queue manager. If APAR PH64232 is applied, V 10.0.0 or the queue manager is running at IBM MQ 9.4.3 or later, the first queue manager to start restores the XCF group entry for each queue manager.

   When you run the CSQ5PQSG utility in this scenario, the following messages are normal:

```
CSQU566I Unable to get attributes for admin structure, CF not found
or not allocated
CSQU546E Unable to add QMGR queue_manager_name entry,
already exists in DB2 table CSQ.ADMIN_B_QMGR
CSQU148I CSQ5PQSG Utility completed, return code=4
```

To recover the queue managers in the queue sharing group:

1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the most recent archived BSDS into it.
4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and does not appear as an archived log. Record the STARTRBA, STARTLRSN, ENDRBA, and ENDLRSN values of this log.
5. Use the change log inventory utility, CSQJU003, to register this latest archive log data set in the BSDS that you have just restored, using the values recorded in Step “4” on page 531.
6. Use the DELETE option of CSQJU003 to remove all active log information from the BSDS.
7. Use the NEWLOG option of CSQJU003 to add active logs to the BSDS, do not specify STARTRBA or ENDRBA.
8. Calculate the *recoverylrsn* for the queue sharing group. The *recoverylrsn* is the lowest of the ENDLRSNs across all queue managers in the queue sharing group (as recorded in Step “4” on page 531), minus 1. For example, if there are two queue managers in the queue sharing group, and the ENDLRSN for one of them is B713 3C72 22C5, and for the other is B713 3D45 2123, the *recoverylrsn* is B713 3C72 22C4.
9. Use CSQJU003 to add a restart control record to the BSDS. Specify:

   ```
   CRESTART CREATE,ENDLRSN= recoverylrsn
   ```

   where *recoverylrsn* is the value you recorded in Step “8” on page 531.

   The BSDS now describes all active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.

   You must add the CRESTART record to the BSDS for each queue manager within the queue sharing group.
10. Restart each queue manager in the queue sharing group with the START QMGR command. During initialization, an operator reply message such as the following is issued:

    ```
    CSQJ245D +CSQ1 RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.
    REPLY Y TO CONTINUE, N TO CANCEL
    ```

    Reply Y to start the queue manager. The queue manager starts, and recovers data up to ENDRBA specified in the CRESTART statement.

The first IBM MQ queue manager started can rebuild the admin structure partitions for other members of the queue sharing group as well as its own, and it is no longer necessary to restart each queue manager in the queue sharing group at this stage.

11. When the admin structure data for all queue managers has been rebuilt, issue a RECOVER CFSTRUCT command for each CF application structure.

    If you issue the RECOVER CFSTRUCT command for all structures on a single queue manager, the log merge process is only performed once, so is quicker than issuing the command on a different queue manager for each CF structure, where each queue manager has to perform the log merge step.

When conditional restart processing is used in a queue sharing group, IBM MQ queue managers, performing peer admin rebuild, check that peers BSDS contain the same CRESTART LRSN as their own. This is to ensure the integrity of the rebuilt admin structure. It is therefore important to restart other peers in the QSG, so they can process their own CRESTART information, before the next unconditional restart of any member of the group.

Using disk mirroring

Many installations now use disk mirroring technologies such as IBM Metro Mirror (formerly PPRC) to make synchronous copies of data sets at an alternative site. In such situations, many of the steps detailed become unnecessary as the IBM MQ page sets and logs at the alternative site are effectively identical to those at the prime site. Where such technologies are used, the steps to restart a queue sharing group at an alternative site may be summarized as:

- Clear IBM MQ CF structures at the alternative site. (These often contain residual information from any previous disaster recovery exercise).
- Restore Db2 systems and all tables in the database used by the IBM MQ queue sharing group.
- Restart queue managers. Before IBM WebSphere MQ 7.0.1, it is necessary to restart each queue manager defined in the queue sharing group as each queue manage recovers its own partition of the admin structure during queue manager restart. After each queue manager has been restarted, those not on their home LPAR can be shut down again. The first IBM MQ queue manager started rebuilds the admin structure partitions for other members of the queue sharing group as well as its own, and it is no longer necessary to restart each queue manager in the queue sharing group.
- After the admin structure has been rebuilt, recover the application structures.

IBM MQ for z/OS supports use of zHyperWrite when writing to active logs mirrored using Metro Mirror. zHyperWrite can help reduce the performance impact of using Metro Mirror; see Using Metro Mirror with IBM MQ for more information.

z/OS *Reinitializing a queue manager*

If the queue manager has terminated abnormally you might not be able to restart it. This could be because your page sets or logs have been lost, truncated, or corrupted. If this has happened, you might have to reinitialize the queue manager (perform a cold start).

Attention

Only perform a cold start if you cannot restart the queue manager any other way. Performing a cold start enables you to recover your queue manager and your object definitions; you will **not** be able to recover your message data. Check that none of the other restart scenarios described in this topic work for you before you do this.

CAUTION: Prepare your queue manager for a possible data loss with no media recovery or unit of work recovery as cold starts use newly defined logs and page sets. It is possible to use the **CSQUTIL SCOPY** command to copy messages from queues on to page sets. However, the **CSQUTIL SCOPY** command does not work for shared queues. You can replay messages from application logs by using the CSQ4LOGS sample.

For persistent messages, you can only replay messages from prior active and archive logs. For more information, see Using CSQ4LOGS to process output from CSQ1LOGP EXTRACT.

When you have restarted, all your IBM MQ objects are defined and available for use, but there is no message data.

Note: Do not reinitialize a queue manager while it is part of a cluster. You must first remove the queue manager from the cluster (using RESET CLUSTER commands on the other queue managers in the cluster), then reinitialize it, and finally reintroduce it to the cluster as a new queue manager.

This is because during reinitialization, the queue manager identifier (QMID) is changed, so any cluster object with the old queue manager identifier must be removed from the cluster.

For further information see the following sections:

- Reinitializing a queue manager that is not in a queue sharing group
- Reinitializing queue managers in a queue sharing group

Reinitializing a queue manager that is not in a queue sharing group

To reinitialize a queue manager, follow this procedure:

1. Prepare the object definition statements that to be used when you restart the queue manager. To do this, either:
   - If page set zero is available, use the CSQUTIL SDEFS function (see Producing a list of IBM MQ define commands). You must get definitions for all object types (authentication information objects, CF structures, channels, namelists, processes, queues, and storage classes).
   - If page set zero is not available, use the definitions from the last time you backed up your object definitions.
2. Redefine your queue manager data sets (do not do this until you have completed step “1” on page 533).

   See creating the bootstrap and log data sets and defining your page sets for more information.
3. Restart the queue manager using the newly defined and initialized log data sets, BSDS, and page sets. Use the object definition input statements that you created in step “1” on page 533 as input in the CSQINP2 initialization input data set.

Reinitializing queue managers in a queue sharing group

In a queue sharing group, reinitializing a queue manager is more complex. It might be necessary to reinitialize one or more queue managers because of page set or log problems, but there might also be problems with Db2 or the coupling facility to deal with. Because of this, there are a number of alternatives:

Cold start
Reinitializing the entire queue sharing group involves forcing all the coupling facilities structures, clearing all object definitions for the queue sharing group from Db2, deleting or redefining the logs and BSDS, and formatting page sets for all the queue managers in the queue sharing group.

Shared definitions retained
Delete or redefine the logs and BSDS, format page sets for all queue managers in the queue sharing group, and force all the coupling facilities structures. On restart, all messages will have been deleted. The queue managers re-create COPY objects that correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist and can be used.

Single queue manager reinitialized
Delete or redefine the logs and BSDS, and format page sets for the single queue manager (this deletes all its private objects and messages). On restart, the queue manager re-creates COPY objects that

correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist, as do the messages on them, and can be used.

Point in time recovery of a queue sharing group
This is the alternative site disaster recovery scenario.

Shared objects are recovered to the point in time achieved by Db2 recovery (described in A Db2 system fails). Each queue manager can be recovered to a point in time achievable from the backup copies available at the alternative site.

Persistent messages can be used in queue sharing groups, and can be recovered using the MQSC RECOVER CFSTRUCT command. Note that this command recovers to the time of failure. However, there is no recovery of nonpersistent shared queue messages; they are lost unless you have made backup copies independently using the COPY function of the CSQUTIL utility program.

It is not necessary to try to restore each queue manager to the same point in time because there are no interdependencies between the local objects on different queue managers (which are what is actually being recovered), and the queue manager resynchronization with Db2 on restart creates or deletes COPY objects as necessary on a queue manager by queue manager basis.

z/OS Using the z/OS Automatic Restart Manager (ARM)

Use this topic to understand how you can use ARM to automatically restart your queue managers.

This section contains information about the following topics:

- "What is the ARM?" on page 534
- "ARM policies" on page 535
- "Using ARM in an IBM MQ network" on page 536

What is the ARM?

The z/OS Automatic Restart Manager (ARM) is a z/OS recovery function that can improve the availability of your queue managers. When a job or task fails, or the system on which it is running fails, ARM can restart the job or task without operator intervention.

If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS, and hence a whole group of related subsystems and applications have failed, ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a *cross-system restart*.

Restart the channel initiator by ARM only in exceptional circumstances. If the queue manager is restarted by ARM, restart the channel initiator from the CSQINP2 initialization data set (see "Using ARM in an IBM MQ network" on page 536).

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure. The network implications of IBM MQ ARM restart on a different z/OS image are described in "Using ARM in an IBM MQ network" on page 536.

To enable automatic restart:

- Set up an ARM couple data set.
- Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
- Start the ARM policy.

Also, IBM MQ must register with ARM at startup (this happens automatically).

Note: If you want to restart queue managers in different z/OS images automatically, you must define every queue manager as a subsystem in each z/OS image on which that queue manager might be restarted, with a sysplex wide unique four character subsystem name.

ARM couple data sets

Ensure that you define the couple data sets required for ARM, and that they are online and active before you start any queue manager for which you want ARM support. IBM MQ automatic ARM registration fails if the couple data sets are not available at queue manager startup. In this situation, IBM MQ assumes that the absence of the couple data set means that you do not want ARM support, and initialization continues.

See *z/OS MVS Setting up a Sysplex* for information about ARM couple data sets.

z/OS *ARM policies*

The Automatic Restart Manager policies are user-defined rules that control ARM functions that can control any restarts of a queue manager.

ARM functions are controlled by a user-defined *ARM policy*. Each z/OS image running a queue manager instance that is to be restarted by ARM must be connected to an ARM couple data set with an active ARM policy.

IBM provides a default ARM policy. You can define new policies, or override the policy defaults by using the *administrative data utility* (IXCMIAPU) provided with z/OS. *z/OS MVS Setting up a Sysplex* describes this utility, and includes full details of how to define an ARM policy.

Figure 36 on page 535 shows an example of an ARM policy. This sample policy restarts any queue manager within a sysplex, if either the queue manager failed, or a whole system failed.

```
//IXCMIAPU EXEC PGM=IXCMIAPU,REGION=2M
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
DATA TYPE(ARM)
DEFINE POLICY NAME(ARMPOL1) REPLACE(YES)
RESTART_GROUP(DEFAULT)
ELEMENT(*)
RESTART_ATTEMPTS(0)  /* Jobs not to be restarted by ARM  */
RESTART_GROUP(GROUP1)
ELEMENT(SYSMQMGRMQ*)  /* These jobs to be restarted by ARM */
/*
```

Figure 36. Sample ARM policy

For more information see:

- Defining an ARM policy
- Activating an ARM policy
- Registering with ARM

Defining an ARM policy

Set up your ARM policy as follows:

- Define RESTART_GROUPs for each queue manager instance that also contain any CICS or IMS subsystems that connect to that queue manager instance. If you use a subsystem naming convention, you might be able to use the '?' and '*' wild-card characters in your element names to define RESTART_GROUPs with minimum definition effort.
- Specify TERMTYPE(ELEMTERM) for your channel initiators to indicate that they will be restarted only if the channel initiator has failed and the z/OS image has not failed.
- Specify TERMTYPE(ALLTERM) for your queue managers to indicate that they will be restarted if either the queue manager has failed or the z/OS image has failed.

- Specify RESTART_METHOD(BOTH, PERSIST) for both queue managers and channel initiators. This tells ARM to restart using the JCL it saved (after resolution of system symbols) during the last startup. It tells ARM to do this irrespective of whether the individual element failed, or the z/OS image failed.
- Accept the default values for all the other ARM policy options.

Activating an ARM policy

To start your automatic restart management policy, issue the following z/OS command:

```
SETXCF START,POLICY,TYPE=ARM,POLNAME= mypol
```

When the policy is started, all systems connected to the ARM couple data set use the same active policy.

Use the SETXCF STOP command to disable automatic restarts.

Registering with ARM

IBM MQ registers automatically as an *ARM element* during queue manager startup (subject to ARM availability). It deregisters during its shutdown phase, unless requested not to.

At startup, the queue manager determines whether ARM is available. If it is, IBM MQ registers using the name SYSMQMGR *ssid*, where *ssid* is the four character queue manager name, and SYSMQMGR is the element type.

The STOP QMGR MODE(QUIESCE) and STOP QMGR MODE(FORCE) commands deregister the queue manager from ARM (if it was registered with ARM at startup). This prevents ARM restarting this queue manager. The STOP QMGR MODE(RESTART) command does not deregister the queue manager from ARM, so it is eligible for immediate automatic restart.

Each channel initiator address space determines whether ARM is available, and if so registers with the element name SYSMQCH *ssid*, where *ssid* is the queue manager name, and SYSMQCH is the element type.

The channel initiator is always deregistered from ARM when it stops normally, and remains registered only if it ends abnormally. The channel initiator is always deregistered if the queue manager fails.

z/OS *Using ARM in an IBM MQ network*

You can set up your queue manager so that the channel initiators and associated listeners are started automatically when the queue manager is restarted.

To ensure fully automatic queue manager restart on the same z/OS image for both LU 6.2 and TCP/IP communication protocols:

- Start your listeners automatically by adding the appropriate START LISTENER command to the CSQINPX data set.
- Start your channel initiator automatically by adding the appropriate START CHINIT command to the CSQINP2 data set.

For restarting a queue manager with TCP/IP or LU6.2, see

- “Restarting on a different z/OS image with TCP/IP” on page 537
- “Restarting on a different z/OS image with LU 6.2” on page 538

See Task 13: Customize the initialization input data sets for information about the CSQINP2 and CSQINPX data sets.

Restarting on a different z/OS image with TCP/IP

If you are using TCP/IP as your communication protocol, and you are using virtual IP addresses, you can configure these to recover on other z/OS images, allowing channels connecting to that queue manager to reconnect without any changes. Otherwise, you can reallocate a TCP/IP address after moving a queue manager to a different z/OS image only if you are using clusters or if you are connecting to a queue sharing group using a WLM dynamic Domain Name System (DNS) logical group name.

- When using clustering
- When connecting to a queue sharing group

When using clustering

z/OS ARM responds to a system failure by restarting the queue manager on a different z/OS image in the same sysplex; this system has a different TCP/IP address to the original z/OS image. The following explains how you can use IBM MQ clusters to reassign a queue manager's TCP/IP address after it has been moved by ARM restart to a different z/OS image.

When a client queue manager detects the queue manager failure (as a channel failure), it responds by reallocating suitable messages on its cluster transmission queue to a different server queue manager that hosts a different instance of the target cluster queue. However, it cannot reallocate messages that are bound to the original server by affinity constraints, or messages that are in doubt because the server queue manager failed during end-of-batch processing. To process these messages, do the following:

1. Allocate a different cluster-receiver channel name and a different TCP/IP port to each z/OS queue manager. Each queue manager needs a different port so that two systems can share a single TCP/IP stack on a z/OS image. One of these is the queue manager originally running on that z/OS image, and the other is the queue manager that ARM will restart on that z/OS image following a system failure. Configure each port on each z/OS image, so that ARM can restart any queue manager on any z/OS image.
2. Create a different channel initiator command input file (CSQINPX) for each queue manager and z/OS image combination, to be referenced during channel initiator startup.

   Each CSQINPX file must include a START LISTENER PORT(port) command specific to that queue manager, and an ALTER CHANNEL command for a cluster-receiver channel specific to that queue manager and z/OS image combination. The ALTER CHANNEL command needs to set the connection name to the TCP/IP name of the z/OS image on which it is restarted. It must include the port number specific to the restarted queue manager as part of the connection name.

   The start-up JCL of each queue manager can have a fixed data set name for this CSQINPX file, and each z/OS image must have a different version of each CSQINPX file on a non-shared DASD volume.

If an ARM restart occurs, IBM MQ advertises the changed channel definition to the cluster repository, which in turn publishes it to all the client queue managers that have expressed an interest in the server queue manager.

The client queue manager treats the server queue manager failure as a channel failure, and tries to restart the failed channel. When the client queue manager learns the new server connection-name, the channel restart reconnects the client queue manager to the restarted server queue manager. The client queue manager can then resynchronize its messages, resolve any in-doubt messages on the client queue manager's transmission queue, and normal processing can continue.

When connecting to a queue sharing group

When connecting to a queue sharing group through a TCP/IP dynamic Domain Name System (DNS) logical group name, the connection name in your channel definition specifies the logical group name of your queue sharing group, not the host name or IP address of a physical machine. When this channel starts, it connects to the dynamic DNS and is then connected to one of the queue managers in the queue sharing group. This process is explained in Setting up communication for IBM MQ for z/OS using queue sharing groups.

In the unlikely event of an image failure, one of the following occurs:

- The queue managers on the failing image de-register from the dynamic DNS running on your sysplex. The channel responds to the connection failure by entering RETRYING state and then connects to the dynamic DNS running on the sysplex. The dynamic DNS allocates the inbound request to one of the remaining members of the queue sharing group that is still running on the remaining images.
- If no other queue manager in the queue sharing group is active and ARM restarts the queue manager and channel initiator on a different image, the group listener registers with dynamic DNS from this new image. This means that the logical group name (from the connection name field of the channel) connects to the dynamic DNS and is then connected to the same queue manager, now running on a different image. No change was required to the channel definition.

For this type of recovery to occur, the following points must be noted:

- On z/OS, the dynamic DNS runs on one of the z/OS images in the sysplex. If this image were to fail, the dynamic DNS needs to be configured so that there is a secondary name server active in the sysplex, acting as an alternative to the primary name server. Information about primary and secondary dynamic DNS servers can be found in the *OS/390® SecureWay CS IP Configuration* manual
- The TCP/IP group listener might have been started on a particular IP address that might not be available on this z/OS image. If so, the listener might need to be started on a different IP address on the new image. If you are using virtual IP addresses, you can configure these to recover on other z/OS images so that no change to the START LISTENER command is required.

Restarting on a different z/OS image with LU 6.2

If you use only LU 6.2 communication protocols, carry out the following procedure to enable network reconnect after automatic restart of a queue manager on a different z/OS image within the sysplex:

- Define each queue manager within the sysplex with a unique subsystem name.
- Define each channel initiator within the sysplex with a unique LUNAME. This is specified in both the queue manager attributes and in the START LISTENER command.

  Note: The LUNAME names an entry in the APPC side table, which in turn maps this to the actual LUNAME.
- Set up a shared APPC side table, which is referenced by each z/OS image within the sysplex. This should contain an entry for each channel initiator's LUNAME. See *z/OS MVS Planning: APPC/MVS Management* for information about this.
- Set up an APPCPM *xx* member of SYS1.PARMLIB for each channel initiator within the sysplex to contain an LUADD to activate the APPC side table entry for that channel initiator. These members should be shared by each z/OS image. The appropriate SYS1.PARMLIB member is activated by a z/OS command SET APPC= *xx*, which is issued automatically during ARM restart of the queue manager (and its channel initiator) on a different z/OS image, as described in the following text.
- Use the LU62ARM queue manager attribute to specify the *xx* suffix of this SYS1.PARMLIB member for each channel initiator. This causes the channel initiator to issue the required z/OS command SET APPC= *xx* to activate its LUNAME.

Define your ARM policy so that it restarts the channel initiator only if it fails while its z/OS image stays up; the user ID associated with the XCFAS address space must be authorized to issue the IBM MQ command START CHINIT. Do not restart the channel initiator automatically if its z/OS image also fails, instead use commands in the CSQINP2 and CSQINPX data sets to start the channel initiator and listeners.

z/OS Recovering units of work manually

You can manually recover units of work CICS, IMS, RRS, or other queue managers in a queue sharing group. You can use queue manager commands to display the status of the units of work associated with each connection to the queue manager.

This topic contains information about the following subjects:

- "Displaying connections and threads" on page 539
- "Recovering CICS units of recovery manually" on page 539
- "Recovering IMS units of recovery manually" on page 543
- "Recovering RRS units of recovery manually" on page 544
- "Recovering units of recovery on another queue manager in the queue sharing group" on page 545

Displaying connections and threads

You can use the DISPLAY CONN command to get information about connections to queue managers and their associated units of work. You can display active units of work to see what is currently happening, or to see what needs to be terminated to allow the queue manager to shut down, and you can display unresolved units of work to help with recovery.

Active units of work

To display only active units of work, use

```
DISPLAY CONN(*) WHERE(UOWSTATE EQ ACTIVE)
```

Unresolved units of work

An unresolved unit of work, also known as an "in-doubt thread", is one that is in the second pass of the two-phase commit operation. Resources are held in IBM MQ on its behalf. To display unresolved units of work, use

```
DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

External intervention is needed to resolve the status of unresolved units of work. This might only involve starting the recovery coordinator (CICS, IMS, or RRS) or might involve more, as described in the following sections.

z/OS *Recovering CICS units of recovery manually*

Use this topic to understand what happens when the CICS adapter restarts, and then explains how to deal with any unresolved units of recovery that arise.

What happens when the CICS adapter restarts

Whenever a connection is broken, the adapter has to go through a *restart phase* during the *reconnect process*. The restart phase resynchronizes resources. Resynchronization between CICS and IBM MQ enables in-doubt units of work to be identified and resolved.

Resynchronization can be caused by:

- An explicit request from the distributed queuing component
- An implicit request when a connection is made to IBM MQ

If the resynchronization is caused by connecting to IBM MQ, the sequence of events is:

1. The connection process retrieves a list of in-doubt units of work (UOW) IDs from IBM MQ.

2. The UOW IDs are displayed on the console in CSQC313I messages.
3. The UOW IDs are passed to CICS.
4. CICS initiates a resynchronization task (CRSY) for each in-doubt UOW ID.
5. The result of the task for each in-doubt UOW is displayed on the console.

You need to check the messages that are displayed during the connect process:

CSQC313I
Shows that a UOW is in doubt.

CSQC400I
Identifies the UOW and is followed by one of these messages:

- CSQC402I or CSQC403I shows that the UOW was resolved successfully (committed or backed out).
- CSQC404E, CSQC405E, CSQC406E, or CSQC407E shows that the UOW was not resolved.

CSQC409I
Shows that all UOWs were resolved successfully.

CSQC408I
Shows that not all UOWs were resolved successfully.

CSQC314I
Warns that UOW IDs highlighted with a * are not resolved automatically. These UOWs must be resolved explicitly by the distributed queuing component when it is restarted.

Figure 37 on page 540 shows an example set of restart messages displayed on the z/OS console.

```
CSQ9022I +CSQ1 CSQYASCP ' START QMGR' NORMAL COMPLETION
+CSQC323I VICIC1 CSQCQCON CONNECT received from TERMID=PB62 TRANID=CKCN
+CSQC303I VICIC1 CSQCCON CSQCSERV loaded. Entry point is 850E8918
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F0E2178D25 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F055B2AC25 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6EFFD60D425 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F07AB56D22 is in doubt
+CSQC307I VICIC1 CSQCCON Successful connection to subsystem VC2
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAD18) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAA10) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BA708) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAE88) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAB80) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA878) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA570) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA268) connect
successful
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F0E2178D25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F055B2AC25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F07AB56D22
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6EFFD60D425
+CSQC409I VICIC1 CSQCTRUE Resynchronization completed successfully
```

Figure 37. Example restart messages

The total number of CSQC313I messages should equal the total number of CSQC402I plus CSQC403I messages. If the totals are not equal, there are UOWs that the connection process cannot resolve. Those UOWs that cannot be resolved are caused by problems with CICS (for example, a cold start) or with IBM MQ, or by distributing queuing. When these problems have been fixed, you can initiate another resynchronization by disconnecting and then reconnecting.

Alternatively, you can resolve each outstanding UOW yourself using the RESOLVE INDOUBT command and the UOW ID shown in message CSQC400I. You must then initiate a disconnect and a connect to clean up the *unit of recovery descriptors* in CICS. You need to know the correct outcome of the UOW to resolve UOWs manually.

All messages that are associated with unresolved UOWs are locked by IBM MQ and no Batch, TSO, or CICS task can access them.

If CICS fails and an emergency restart is necessary, *do not* vary the GENERIC APPLID of the CICS system. If you do and then reconnect to IBM MQ, data integrity with IBM MQ cannot be guaranteed. This is because IBM MQ treats the new instance of CICS as a different CICS (because the APPLID is different). In-doubt resolution is then based on the wrong CICS log.

How to resolve CICS units of recovery manually

If the adapter ends abnormally, CICS and IBM MQ build in-doubt lists either dynamically or during restart, depending on which subsystem caused the abend.

Note: If you use the DFH$INDB sample program to show units of work, you might find that it does not always show IBM MQ UOWs correctly.

When CICS connects to IBM MQ, there might be one or more units of recovery that have not been resolved.

One of the following messages is sent to the console:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E
- CSQC408I

For details of what these messages mean, see the CICS adapter and Bridge messages messages.

CICS retains details of units of recovery that were not resolved during connection startup. An entry is purged when it no longer appears on the list presented by IBM MQ.

Any units of recovery that CICS cannot resolve must be resolved manually using IBM MQ commands. This manual procedure is rarely used within an installation, because it is required only where operational errors or software problems have prevented automatic resolution. *Any inconsistencies found during in-doubt resolution must be investigated.*

To resolve the units of recovery:

1. Obtain a list of the units of recovery from IBM MQ using the following command:

```
+CSQ1  DISPLAY CONN( * ) WHERE(UOWSTATE EQ UNRESOLVED)
```

   You receive the following message:

```
CSQM201I +CSQ1 CSQMDRTC DISPLAY CONN DETAILS
CONN(BC85772CBE3E0001)
EXTCONN(C3E2D8C3C7D9F0F94040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2005-02-04)
UOWLOGTI(10.17.44)
UOWSTDA(2005-02-04)
UOWSTTI(10.17.44)
UOWSTATE(UNRESOLVED)
NID(IYRCSQ1 .BC8571519B60222D)
EXTURID(BC8571519B60222D)
QMURID(0000002BDA50)
URTYPE(CICS)
USERID(MQTEST)
APPLTAG(IYRCSQ1)
ASID(0000)
APPLTYPE(CICS)
TRANSID(GP02)
TASKNO(0000096)
END CONN DETAILS
```

For CICS connections, the NID consists of the CICS applid and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is stored in records written to both the CICS system log and the IBM MQ log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

2. Scan the CICS log for entries related to a particular unit of recovery.

   Look for a PREPARE record for the task-related installation where the recovery token field (JCSRMTKN) equals the value obtained from the network ID. The network ID is supplied by IBM MQ in the DISPLAY CONN command output.

   The PREPARE record in the CICS log for the units of recovery provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

   You can use the CICS journal print utility DFHJUP when scanning the log. For details of using this program, see the *CICS Operations and Utilities Guide*.

3. Scan the IBM MQ log for records with the NID related to a particular unit of recovery. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

   When scanning the IBM MQ log, note that the IBM MQ startup message CSQJ001I provides the start RBA for this session.

   The print log records program (CSQ1LOGP) can be used for that purpose.

4. If you need to, do in-doubt resolution in IBM MQ.

   IBM MQ can be directed to take the recovery action for a unit of recovery using an IBM MQ RESOLVE INDOUBT command.

   To recover all threads associated with a specific *connection-name*, use the NID(*) option.

   The command produces one of the following messages showing whether the thread is committed or backed out:

```
CSQV414I +CSQ1 THREAD network-id COMMIT SCHEDULED
CSQV415I +CSQ1 THREAD network-id ABORT SCHEDULED
```

When performing in-doubt resolution, CICS and the adapter are not aware of the commands to IBM MQ to commit or back out units of recovery, because only IBM MQ resources are affected. However, CICS keeps details about the in-doubt threads that could not be resolved by IBM MQ. This information is purged either when the list presented is empty, or when the list does not include a unit of recovery of which CICS has details.

z/OS *Recovering IMS units of recovery manually*

Use this topic to understand what happens when the IMS adapter restarts, and then explains how to deal with any unresolved units of recovery that arise.

What happens when the IMS adapter restarts

Whenever the connection to IBM MQ is restarted, either following a queue manager restart or an IMS / START SUBSYS command, IMS initiates the following resynchronization process:

1. IMS presents the list of unit of work (UOW) IDs that it believes are in doubt to the IBM MQ IMS adapter one at a time with a resolution parameter of Commit or Backout.
2. The IMS adapter passes the resolution request to IBM MQ and reports the result back to IMS.
3. Having processed all the IMS resolution requests, the IMS adapter gets from IBM MQ a list of all UOWs that IBM MQ still holds in doubt that were initiated by the IMS system. These are reported to the IMS master terminal in message CSQQ008I.

Note: While a UOW is in doubt, any associated IBM MQ message is locked by IBM MQ and is not available to any application.

How to resolve IMS units of recovery manually

When IMS connects to IBM MQ, IBM MQ might have one, or more in-doubt units of recovery that have not been resolved.

If IBM MQ has in-doubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal:

```
CSQQ008I nn units of recovery are still in doubt in queue manager qmgr-name
```

If this message is issued, IMS was either cold-started or it was started with an incomplete log tape. This message can also be issued if IBM MQ or IMS terminates abnormally because of a software error or other subsystem failure.

After receiving the CSQQ008I message:

- The connection remains active.
- IMS applications can still access IBM MQ resources.
- Some IBM MQ resources remain locked out.

If the in-doubt thread is not resolved, IMS message queues can start to build up. If the IMS queues fill to capacity, IMS terminates. You must be aware of this potential difficulty, and you must monitor IMS until the in-doubt units of recovery are fully resolved.

Recovery procedure

Use the following procedure to recover the IMS units of work:

1. Force the IMS log closed, using /SWI OLDS, and then archive the IMS log. Use the utility, DFSERA10, to print the records from the previous IMS log tape. Type X'3730' log records indicate a phase-2 commit request and type X'38' log records indicate an abort request. Record the requested action for the last transaction in each dependent region.
2. Run the DL/I batch job to back out each PSB involved that has not reached a commit point. The process might take some time because transactions are still being processed. It might also lock up a number of records, which could affect the rest of the processing and the rest of the message queues.

3. Produce a list of the in-doubt units of recovery from IBM MQ using the following command:

```
+CSQ1 DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

You receive the following message:

```
CSQM201I +CSQ1 CSQMDRTC DISPLAY CONN DETAILS
CONN(BC45A794C4290001)
EXTCONN(C3E2D8C3E2C5C3F24040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2005-02-15)
UOWLOGTI(16.39.43)
UOWSTDA(2005-02-15)
UOWSTTI(16.39.43)
UOWSTATE(UNRESOLVED)
NID(IM8F  .BC45A794D3810344)
EXTURID(
0000052900000000
)
QMURID(00000354B76E)
URTYPE(IMS)
USERID(STCPI)
APPLTAG(IM8F)
ASID(0000)
APPLTYPE(IMS)
PSTID(0004)
PSBNAME(GP01MPP)
```

For IMS, the NID consists of the IMS connection name and a unique number provided by IMS. The value is referred to in IMS as the *recovery token*. For more information, see the IMS documentation.

4. Compare the NIDs (IMSID plus OASN in hexadecimal) displayed in the DISPLAY THREAD messages with the OASNs (4 bytes decimal) shown in the DFSERA10 output. Decide whether to commit or back out.
5. Perform in-doubt resolution in IBM MQ with the RESOLVE INDOUBT command, as follows:

```
RESOLVE INDOUBT( connection-name )
ACTION(COMMIT|BACKOUT)
NID( network-id )
```

To recover all threads associated with *connection-name*, use the NID(*) option. The command results in one of the following messages to indicate whether the thread is committed or backed out:

```
CSQV414I  THREAD network-id COMMIT SCHEDULED
CSQV415I  THREAD network-id BACKOUT SCHEDULED
```

When performing in-doubt resolution, IMS and the adapter are not aware of the commands to IBM MQ to commit or back out in-doubt units of recovery because only IBM MQ resources are affected.

z/OS *Recovering RRS units of recovery manually*

Use this topic to understand the how to determine if there are in-doubt RRS units of recovery, and how to manually resolve those units of recovery.

When RRS connects to IBM MQ, IBM MQ might have one, or more in-doubt units of recovery that have not been resolved. If IBM MQ has in-doubt units of recovery that RRS did not resolve, one of the following messages is issued at the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

Both IBM MQ and RRS provide tools to display information about in-doubt units of recovery, and techniques for manually resolving them.

In IBM MQ, use the DISPLAY CONN command to display information about in-doubt IBM MQ threads. The output from the command includes RRS unit of recovery IDs for those IBM MQ threads that have RRS as a coordinator. This can be used to determine the outcome of the unit of recovery.

Use the RESOLVE INDOUBT command to resolve the IBM MQ in-doubt thread manually. This command can be used to either commit or back out the unit of recovery after you have determined what the correct decision is.

z/OS *Recovering units of recovery on another queue manager in the queue sharing group*

Use this topic to identify, and manually recover units of recovery on other queue managers in a queue sharing group.

If a queue manager that is a member of a queue sharing group fails and cannot be restarted, other queue managers in the group can perform peer recovery, and take over from it. However, the queue manager might have in-doubt units of recovery that cannot be resolved by peer recovery because the final disposition of that unit of recovery is known only to the failed queue manager. These units of recovery are resolved when the queue manager is eventually restarted, but until then, they remain in doubt.

This means that certain resources (for example, messages) might be locked, making them unavailable to other queue managers in the group. In this situation, you can use the DISPLAY THREAD command to display these units of work on the inactive queue manager. If you want to resolve these units of recovery manually to make the messages available to other queue managers in the group, you can use the RESOLVE INDOUBT command.

When you issue the DISPLAY THREAD command to display units of recovery that are in doubt, you can use the QMNAME keyword to specify the name of the inactive queue manager. For example, if you issue the following command:

```
+CSQ1 DISPLAY THREAD(*) TYPE(INDOUBT) QMNAME(QM01)
```

You receive the following messages:

```
CSQV436I +CSQ1 INDOUBT THREADS FOR QM01 -
NAME   THREAD-XREF    URID NID
USER1  00000000000000000000000 CSQ:0001.0
USER2  00000000000000000000000 CSQ:0002.0
DISPLAY THREAD REPORT COMPLETE
```

If the queue manager specified is active, IBM MQ does not return information about in-doubt threads, but issues the following message:

```
CSQV435I CANNOT USE QMNAME KEYWORD, QM01 IS ACTIVE
```

Use the IBM MQ command RESOLVE INDOUBT to resolve the in-doubt threads manually. Use the QMNAME keyword to specify the name of the inactive queue manager in the command.

This command can be used to commit or back out the unit of recovery. The command resolves the shared portion of the unit of recovery only; any local messages are unaffected and remain locked until the queue manager restarts, or reconnects to CICS, IMS, or RRS batch.

z/OS IBM MQ and IMS

IBM MQ provides two components to interface with IMS, the IBM MQ - IMS adapter, and the IBM MQ - IMS bridge. These components are commonly called the IMS adapter, and the IMS bridge.

z/OS Operating the IMS adapter

Use this topic to understand how to operate the IMS adapter, which connects IBM MQ to IMS systems.

Note: The IMS adapter does not incorporate any operations and control panels.

This topic contains the following sections:

- "Controlling IMS connections" on page 546
- "Connecting from the IMS control region" on page 546
- "Displaying in-doubt units of recovery" on page 548
- "Controlling IMS dependent region connections" on page 550
- "Disconnecting from IMS" on page 552
- "Controlling the IMS trigger monitor" on page 553

z/OS *Controlling IMS connections*

Use this topic to understand the IMS operator commands which control and monitor the connection to IBM MQ.

IMS provides the following operator commands to control and monitor the connection to IBM MQ:

/CHANGE SUBSYS
Deletes an in-doubt unit of recovery from IMS.

/DISPLAY OASN SUBSYS
Displays outstanding recovery elements.

/DISPLAY SUBSYS
Displays connection status and thread activity.

/START SUBSYS
Connects the IMS control region to a queue manager.

/STOP SUBSYS
Disconnects IMS from a queue manager.

/TRACE
Controls the IMS trace.

For more information about these commands, see the *IMS/ESA® Operator's Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

z/OS *Connecting from the IMS control region*

Use this topic to understand the mechanisms available to connect from IMS to IBM MQ.

IMS makes one connection from its control region to each queue manager that uses IMS. IMS must be enabled to make the connection in one of these ways:

- Automatically during either:
  - A cold start initialization.
  - A warm start of IMS, if the IBM MQ connection was active when IMS was shut down.
- In response to the IMS command:

```
/START SUBSYS sysid
```

where *sysid* is the queue manager name.

The command can be issued regardless of whether the queue manager is active.

The connection is not made until the first IBM MQ API call to the queue manager is made. Until that time, the IMS command /DIS SUBSYS shows the status as 'NOT CONN'.

The order in which you start IMS and the queue manager is not significant.

IMS cannot re-enable the connection to the queue manager automatically if the queue manager is stopped with a STOP QMGR command, the IMS command /STOP SUBSYS, or an abnormal end. Therefore, you must make the connection by using the IMS command /START SUBSYS.

If an IMS command is seen in the queue manager console log similar to this:

```
MODIFY IMS*,SS*
```

check the IMS master log and ensure that IBM MQ has RACF authority to issue IMS Adapter MODIFY commands.

Initializing the adapter and connecting to the queue manager

The adapter is a set of modules loaded into the IMS control and dependent regions, using the IMS external Subsystem Attach Facility.

This procedure initializes the adapter and connects to the queue manager:

1. Read the subsystem member (SSM) from IMS.PROCLIB. The SSM chosen is an IMS EXEC parameter. There is one entry in the member for each queue manager to which IMS can connect. Each entry contains control information about an IBM MQ adapter.
2. Load the IMS adapter.

   Note: IMS loads one copy of the adapter modules for each IBM MQ instance that is defined in the SSM member.
3. Attach the external subsystem task for IBM MQ.
4. Run the adapter with the CTL EXEC parameter (IMSID) as the connection name.

The process is the same whether the connection is part of initialization or a result of the IMS command / START SUBSYS.

If the queue manager is active when IMS tries to make the connection, the following messages are sent:

- to the z/OS console:

```
DFS3613I ESS TCB INITIALIZATION COMPLETE
```

- to the IMS master terminal:

```
CSQQ000I IMS/TM imsid connected to queue manager ssnm
```

When IMS tries to make the connection and *the queue manager is not active*, the following messages are sent to the IMS master terminal each time an application makes an MQI call:

```
CSQQ001I IMS/TM imsid not connected to queue manager ssnm.
Notify message accepted
DFS3607I MQM1   SUBSYSTEM ID EXIT FAILURE, FC = 0286, RC = 08,
JOBNAME = IMSEMPR1
```

If you get DFS3607I messages when you start the connection to IMS or on system startup, this indicates that the queue manager is not available. To prevent a large number of messages being generated, you must do one of the following:

1. Start the relevant queue manager.
2. Issue the IMS command:

   ```
   /STOP SUBSYS
   ```

   so that IMS does not expect to connect to the queue manager.

If you do neither, a DFS3607I message and the associated CSQQ001I message are issued each time a job is scheduled in the region and each time a connection request to the queue manager is made by an application.

Thread attachment

In an MPP or IFP region, IMS makes a thread connection when the first application program is scheduled into that region, even if that application program does not make an IBM MQ call. In a BMP region, the thread connection is made when the application makes its first IBM MQ call (MQCONN or MQCONNX). This thread is retained for the duration of the region or until the connection is stopped.

For both the message driven and non-message driven regions, the recovery thread cross-reference identifier, *Thread-xref*, associated with the thread is:

```
PSTid + PSBname
```

where:

PSTid
Partition specification table region identifier

PSBname
Program specification block name

You can use connection IDs as unique identifiers in IBM MQ commands, in which case IBM MQ automatically inserts these IDs into any operator message that it generates.

z/OS *Displaying in-doubt units of recovery*

You can display in-doubt of units of recovery and attempt to recover them.

The operational steps used to list and recover in-doubt units of recovery in this topic are for relatively simple cases only. If the queue manager ends abnormally while connected to IMS, IMS might commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*. A decision must be made about the status of the work.

To display a list of in-doubt units of recovery, issue the command:

```
+CSQ1  DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

IBM MQ responds with a message like the following:

```
CSQM201I +CSQ1 CSQMDRTC DIS CONN DETAILS
CONN(BC0F6125F5A30001)
EXTCONN(C3E2D8C3C3E2D8F14040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2004-11-02)
UOWLOGTI(12.27.58)
UOWSTDA(2004-11-02)
UOWSTTI(12.27.58)
UOWSTATE(UNRESOLVED)
NID(CSQ1CHIN.BC0F5F1C86FC0766)
EXTURID(000000000000001F000000007472616E5F6964547565204E6F762020...)
QMURID(000000026232)
URTYPE(XA)
USERID( )
APPLTAG(CSQ1CHIN)
ASID(0000)
APPLTYPE(CHINIT)
CHANNEL( )
CONNAME( )
END CONN DETAILS
```

For an explanation of the attributes in this message, see the description of the DISPLAY CONN command.

Recovering in-doubt units of recovery

To recover in-doubt units of recovery, issue this command:

```
+CSQ1  RESOLVE INDOUBT( connection-name ) ACTION(COMMIT|BACKOUT)
NID( net-node.number )
```

where:

connection-name
The IMS system ID.

ACTION
Indicates whether to commit (COMMIT) or back out (BACKOUT) this unit of recovery.

net-node.number
The associated net-node.number.

When you have issued the RESOLVE INDOUBT command, one of the following messages is displayed:

```
CSQV414I +CSQ1 THREAD network-id COMMIT SCHEDULED

CSQV415I +CSQ1 THREAD network-id BACKOUT SCHEDULED
```

Resolving residual recovery entries

At given times, IMS builds a list of residual recovery entries (RREs). RREs are units of recovery about which IBM MQ might be in doubt. They arise in several situations:

- If the queue manager is not active, IMS has RREs that cannot be resolved until the queue manager is active. These RREs are not a problem.

- If the queue manager is active and connected to IMS, and if IMS backs out the work that IBM MQ has committed, the IMS adapter issues message CSQQ010E. If the data in the two systems must be consistent, there is a problem. For information about resolving this problem, see "Recovering IMS units of recovery manually" on page 543.
- If the queue manager is active and connected to IMS, there might still be RREs even though no messages have informed you of this problem. After the IBM MQ connection to IMS has been established, you can issue the following IMS command to find out if there is a problem:

```
/DISPLAY OASN SUBSYS sysid
```

To purge the RRE, issue one of the following IMS commands:

```
/CHANGE SUBSYS sysid RESET
/CHANGE SUBSYS sysid RESET OASN nnnn
```

where *nnnn* is the originating application sequence number listed in response to your +CSQ1 DISPLAY command. This is the schedule number of the program instance, giving its place in the sequence of invocations of that program since the last IMS cold start. IMS cannot have two in-doubt units of recovery with the same schedule number.

These commands reset the status of IMS ; they do not result in any communication with IBM MQ.

z/OS *Controlling IMS dependent region connections*

You can control, monitor, and, when necessary, terminate connections between IMS and IBM MQ.

Controlling IMS dependent region connections involves the following activities:

- Connecting from dependent regions
- Region error options
- Monitoring the activity on connections
- Disconnecting from dependent regions

Connecting from dependent regions

The IMS adapter used in the control region is also loaded into dependent regions. A connection is made from each dependent region to IBM MQ. This connection is used to coordinate the commitment of IBM MQ and IMS work. To initialize and make the connection, IMS does the following:

1. Reads the subsystem member (SSM) from IMS.PROCLIB.

   A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region is never likely to connect to IBM MQ, to avoid loading the adapter, specify a member with no entries.
2. Loads the IBM MQ adapter.

   For a batch message program, the load is not done until the application issues its first messaging command. At that time, IMS tries to make the connection.

   For a message-processing program region or IMS fast-path region, the attempt is made when the region is initialized.

Region error options

If the queue manager is not active, or if resources are not available when the first messaging command is sent from application programs, the action taken depends on the error option specified on the SSM entry. The options are:

R
The appropriate return code is sent to the application.

Q
The application ends abnormally with abend code U3051. The input message is re-queued.

A
The application ends abnormally with abend code U3047. The input message is discarded.

Monitoring the activity on connections

A thread is established from a dependent region when an application makes its first successful IBM MQ request. You can display information about connections and the applications currently using them by issuing the following command from IBM MQ:

```
+CSQ1 DISPLAY CONN(*) ALL
```

The command produces a message like the following:

```
CONN(BC45A794C4290001)
EXTCONN(C3E2D8C3C3E2D8F14040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2004-12-15)
UOWLOGTI(16.39.43)
UOWSTDA(2004-12-15)
UOWSTTI(16.39.43)
UOWSTATE(ACTIVE)
NID( )
EXTURID(
0000052900000000
)
QMURID(00000354B76E)
URTYPE(IMS)
USERID(STCPI)
APPLTAG(IM8F)
ASID(0049)
APPLTYPE(IMS)
PSTID(0004)
PSBNAME(GP01MPP)
```

For the control region, *thread-xref* is the special value CONTROL. For dependent regions, it is the PSTid concatenated with the PSBname. *auth-id* is either the user field from the job card, or the ID from the z/OS started procedures table.

For an explanation of the displayed list, see the description of message CSQV402I in the IBM MQ for z/OS messages, completion, and reason codes documentation.

IMS provides a display command to monitor the connection to IBM MQ. It shows which program is active on each dependent region connection, the LTERM user name, and the control region connection status. The command is:

```
/DISPLAY SUBSYS name
```

The status of the connection between IMS and IBM MQ is shown as one of:

```
CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
STOP IN PROGRESS
INVALID SUBSYSTEM NAME= name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
```

The thread status from each dependent region is one of the following:

```
CONN
CONN, ACTIVE (includes LTERM of user)
```

Disconnecting from dependent regions

To change values in the SSM member of IMS.PROCLIB, you disconnect a dependent region. To do this, you must:

1. Issue the IMS command:

   ```
   /STOP REGION
   ```

2. Update the SSM member.
3. Issue the IMS command:

   ```
   /START REGION
   ```

z/OS *Disconnecting from IMS*

The connection is ended when either IMS or the queue manager terminates. Alternatively, the IMS master terminal operator can explicitly break the connection.

To terminate the connection between IMS and IBM MQ, use the following IMS command:

```
/STOP SUBSYS sysid
```

The command sends the following message to the terminal that issued it, typically the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The IMS command:

```
/START SUBSYS sysid
```

is required to reestablish the connection.

Note: The IMS command /STOP SUBSYS is not completed if an IMS trigger monitor is running.

z/OS *Controlling the IMS trigger monitor*

You can use the CSQQTRMN transaction to stop, and start the IMS trigger monitor.

The IMS trigger monitor (the CSQQTRMN transaction) is described in the Setting up the IMS trigger monitor.

To control the IMS trigger monitor see:

- Starting CSQQTRMN
- Stopping CSQQTRMN

Starting CSQQTRMN

1. Start a batch-oriented BMP that runs the program CSQQTRMN for each initiation queue you want to monitor.
2. Modify your batch JCL to add a DDname of CSQQUT1 that points to a data set containing the following information:

```
QMGRNAME=q_manager_name    Comment: queue manager name
INITQUEUENAME=init_q_name   Comment: initiation queue name
LTERM=lterm          Comment: LTERM to remove error messages
CONSOLEMESSAGES=YES      Comment: Send error messages to console
```

   where:

`q_manager_name`	The name of the queue manager (if this is blank, the default nominated in CSQQDEFV is assumed)
`init_q_name`	The name of the initiation queue to be monitored
`lterm`	The IMS LTERM name for the destination of error messages (if this is blank, the default value is MASTER).
`CONSOLEMESSAGES=YES`	Requests that messages sent to the nominated IMS LTERM are also sent to the z/OS console. If this parameter is omitted or misspelled, the default is NOT to send messages to the console.

3. Add a DD name of CSQQUT2 if you want a printed report of the processing of CSQQUT1 input.

Note:

1. The data set CSQQUT1 is defined with LRECL=80. Other DCB information is taken from the data set. The DCB for data set CSQQUT2 is RECFM=VBA and LRECL=125.
2. You can put only one keyword on each record. The keyword value is delimited by the first blank following the keyword; this means that you can include comments. An asterisk in column 1 means that the whole input record is a comment.
3. If you misspell either of the `QMGRNAME` or `LTERM` keywords, CSQQTRMN uses the default for that keyword.
4. Ensure that the subsystem is started in IMS (by the /START SUBSYS command) before submitting the trigger monitor BMP job. If it is not started, your trigger monitor job terminates with abend code U3042.

Stopping CSQQTRMN

Once started, CSQQTRMN runs until either the connection between IBM MQ and IMS is broken due to one of the following events:

- the queue manager ending
- IMS ending

or a z/OS STOP **jobname** command is entered.

z/OS Controlling the IMS bridge

Use this topic to understand the IMS commands that you can use to control the IMS bridge.

There are no IBM MQ commands to control the IBM MQ-IMS bridge. However, you can stop messages being delivered to IMS in the following ways:

- For non-shared queues, by using the ALTER QLOCAL(xxx) GET(DISABLED) command for all bridge queues.
- For clustered queues, by using the SUSPEND QMGR CLUSTER(xxx) command. This is effective only when another queue manager is also hosting the clustered bridge queue.
- For clustered queues, by using the SUSPEND QMGR FACILITY(IMSBRIDGE) command. No further messages are sent to IMS, but the responses for any outstanding transactions are received from IMS.

  To start sending messages to IMS again, issue the RESUME QMGR FACILITY(IMSBRIDGE) command.

You can also use the MQSC command DISPLAY SYSTEM to display whether the bridge is suspended.

See MQSC commands for details of these commands.

For further information see:

- “Starting and stopping the IMS bridge” on page 554
- “Controlling IMS connections” on page 554
- Controlling bridge queues
- “Resynchronizing the IMS bridge” on page 556
- Working with tpipe names
- Deleting messages from IMS
- Deleting tpipes
- “IMS Transaction Expiration” on page 557

Starting and stopping the IMS bridge

Start the IBM MQ bridge by starting OTMA. Either use the IMS command:

```
/START OTMA
```

or start it automatically by specifying OTMA=YES in the IMS system parameters. If OTMA is already started, the bridge starts automatically when queue manager startup has completed. An IBM MQ event message is produced when OTMA is started.

Use the IMS command:

```
/STOP OTMA
```

to stop OTMA communication. When this command is issued, an IBM MQ event message is produced.

Controlling IMS connections

IMS provides these operator commands to control and monitor the connection to IBM MQ:

/DEQUEUE TMEMBER *tmember* TPIPE *tpipe*
Removes messages from a Tpipe. Specify PURGE to remove all messages or PURGE1 to remove the first message only.

/DISPLAY OTMA
Displays summary information about the OTMA server and clients, and client status.

/DISPLAY TMEMBER *name*
Displays information about an OTMA client.

/DISPLAY TRACE TMEMBER *name*
Displays information about what is being traced.

/SECURE OTMA
Sets security options.

/START OTMA
Enables communications through OTMA.

/START TMEMBER *tmember* TPIPE *tpipe*
Starts the named Tpipe.

/STOP OTMA
Stops communications through OTMA.

/STOP TMEMBER *tmember* TPIPE *tpipe*
Stops the named Tpipe.

/TRACE
Controls the IMS trace.

For more information about these commands, see the *IMS/ESA Operators Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

Controlling bridge queues

To stop communicating with the queue manager with XCF member name *tmember* through the bridge, issue the following IMS command:

```
/STOP TMEMBER tmember TPIPE ALL
```

To resume communication, issue the following IMS command:

```
/START TMEMBER tmember TPIPE ALL
```

The Tpipes for a queue can be displayed using the MQ DISPLAY QUEUE command.

To stop communication with the queue manager on a single Tpipe, issue the following IMS command:

```
/STOP TMEMBER tmember TPIPE tpipe
```

One or two Tpipes are created for each active bridge queue, so issuing this command stops communication with the IBM MQ queue. To resume communication, use the following IMS command :

```
/START TMEMBER tmember TPIPE tpipe
```

Alternatively, you can alter the attributes of the IBM MQ queue to make it get inhibited.

Resynchronizing the IMS bridge

The IMS bridge is automatically restarted whenever the queue manager, IMS, or OTMA are restarted.

The first task undertaken by the IMS bridge is to resynchronize with IMS. This involves IBM MQ and IMS checking sequence numbers on every synchronized Tpipe. A synchronized Tpipe is used when persistent messages are sent to IMS from an IBM MQ - IMS bridge queue using commit mode zero (commit-then-send).

If the bridge cannot resynchronize with IMS, the IMS sense code is returned in message CSQ2023E and the connection to OTMA is stopped. If the bridge cannot resynchronize with an individual IMS Tpipe, the IMS sense code is returned in message CSQ2025E and the Tpipe is stopped. If a Tpipe has been cold started, the recoverable sequence numbers are automatically reset to 1.

If the bridge discovers mismatched sequence numbers when resynchronizing with a Tpipe, message CSQ2020E is issued. Use the IBM MQ command RESET TPIPE to initiate resynchronization with the IMS Tpipe. You need to provide the XCF group and member name, and the name of the Tpipe; this information is provided by the message.

You can also specify:

- A new recoverable sequence number to be set in the Tpipe for messages sent by IBM MQ, and to be set as the partner's receive sequence number. If you do not specify this, the partner's receive sequence number is set to the current IBM MQ send sequence number.
- A new recoverable sequence number to be set in the Tpipe for messages received by IBM MQ, and to be set as the partner's send sequence number. If you do not specify this, the partner's send sequence number is set to the current IBM MQ receive sequence number.

If there is an unresolved unit of recovery associated with the Tpipe, this is also notified in the message. Use the IBM MQ command RESET TPIPE to specify whether to commit the unit of recovery, or back it out. If you commit the unit of recovery, the batch of messages has already been sent to IMS, and is deleted from the bridge queue. If you back the unit of recovery out, the messages are returned to the bridge queue, to be later sent to IMS.

Commit mode 1 (send-then-commit) Tpipes are not synchronized.

Considerations for Commit mode 1 transactions

In IMS, commit mode 1 (CM1) transactions send their output replies before sync point.

A CM1 transaction might not be able to send its reply, for example because:

- The Tpipe on which the reply is to be sent is stopped
- OTMA is stopped
- The OTMA client (that is, the queue manager) has gone away
- The reply-to queue and dead-letter queue are unavailable

For these reasons, the IMS application sending the message pseudo-abends with code U0119. The IMS transaction and program are not stopped in this case.

These reasons often prevent messages being sent into IMS, as well as replies being delivered from IMS. A U0119 abend can occur if:

- The Tpipe, OTMA, or the queue manager is stopped while the message is in IMS
- IMS replies on a different Tpipe to the incoming message, and that Tpipe is stopped
- IMS replies to a different OTMA client, and that client is unavailable.

Whenever a U0119 abend occurs, both the incoming message to IMS and the reply messages to IBM MQ are lost. If the output of a CM0 transaction cannot be delivered for any of these reasons, it is queued on the Tpipe within IMS.

Working with tpipe names

Many of the commands used to control the IBM MQ - IMS bridge require the *tpipe* name. Use this topic to understand how you can find further details of the tpipe name.

You need *tpipe* names for many of the commands that control the IBM MQ - IMS bridge. You can get the tpipe names from DISPLAY QUEUE command and note the following points:

- tpipe names are assigned when a local queue is defined
- a local queue is given two tpipe names, one for sync and one for non-sync
- tpipe names will not be known to IMS until after some communication between IMS and IBM MQ specific to that particular local queue takes place
- For a tpipe to be available for use by the IBM MQ - IMS bridge its associated queue must be assigned to a Storage Class that has the correct XCF group and member name fields completed

Deleting messages from IMS

A message that is destined for IBM MQ through the IMS bridge can be deleted if the Tmember/Tpipe is stopped. To delete one message for the queue manager with XCF member name *tmember*, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE1
```

To delete all the messages on the Tpipe, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE
```

Deleting tpipes

You cannot delete IMS tpipes yourself. They are deleted by IMS at the following times:

- Synchronized tpipes are deleted when IMS is cold started.
- Non-synchronized tpipes are deleted when IMS is restarted.

IMS Transaction Expiration

An expiration time is associated with a transaction; any IBM MQ message can have an expiration time associated with it. The expiration interval is passed from the application, to IBM MQ, using the MQMD.Expiry field. The time is the duration of a message before it expires, expressed as a value in tenths of a second. An attempt to perform the MQGET of a message, later than it has expired, results in the message being removed from the queue and expiry processing performed. The expiration time decreases as a message flows between queue managers on an IBM MQ network. When an IMS message is passed across the IMS bridge to OTMA, the remaining message expiry time is passed to OTMA as a transaction expiration time.

If a transaction has an expiration time specified, OTMA expires the input transactions in three different places in IMS:

- input message receiving from XCF
- input message enqueuing time
- application GU time

No expiration is performed after the GU time.

The transaction EXPRTIME can be provided by:

- IMS transaction definition
- IMS OTMA message header
- IMS DFSINSX0 user exit
- IMS CREATE or UPDATE TRAN commands

IMS indicates that it has expired a transaction by abending a transaction with 0243, and issuing a message. The message issued is either DFS555I in the non-shared-queues environment, or DFS2224I in the shared-queues environment.

z/OS Operating Advanced Message Security on z/OS

The Advanced Message Security address space accepts commands using the z/OS MODIFY command.

Procedure

- Modify Advanced Message Security on z/OS.

  To enter commands for the Advanced Message Security (AMS) address space, use the z/OS MODIFY command.

  For example:

  ```
  F qmgrAMSM, cmd
  ```

  where *qmgr* is the prefix of the started task name.

  The following table describes the MODIFY commands that are accepted:

  Table 31. Advanced Message Security address space MODIFY commands

Command	Option	Description
DISPLAY		Display version information
REFRESH	KEYRING POLICY ALL	Refresh the key ring certificates, security policies, or both.
SMFAUDIT	SUCCESS FAILURE ALL	Set whether SMF auditing is required when AMS successfully protects or unprotects messages, when AMS fails to protect or unprotect messages, or both.
SMFTYPE	0 - 255	Set the SMF record type to be generated when AMS protects or unprotects messages. To disable SMF auditing specify a record type of 0.

  Note: To specify an option it must be separated by a comma. For example:

  ```
  F qmgrAMSM,REFRESH KEYRING
  F qmgrAMSM,SMFAUDIT ALL
  F qmgrAMSM,SMFTYPE 180
  ```

- Refresh Advanced Message Security on z/OS.

  Changes that are made effective by issuing the **REFRESH** command apply to applications that issue MQOPEN after the **REFRESH** command has completed. Existing applications that have a queue open, continue to use the options from when the application opened the queue. To use the new values, the application has to close and reopen the queue.

- Start and stop AMS on z/OS.

You do not need to enter a command to start or stop the Advanced Message Security address space. The AMS address space is started automatically when the queue manager is started if AMS has been enabled with the **SPLCAP** parameter of CSQ6SYSP, and is stopped when the queue manager is stopped.

Administering IBM MQ Internet Pass-Thru

This section describes how to administer IBM MQ Internet Pass-Thru (MQIPT).

Configure MQIPT by making changes to the `mqipt.conf` configuration file as described in Configuring IBM MQ Internet Pass-Thru. To administer MQIPT, including refreshing MQIPT to bring configuration changes into effect without restarting MQIPT, use the **mqiptAdmin** command. For information about administering MQIPT using the **mqiptAdmin** command, see "Administering MQIPT by using the command line" on page 561.

Starting and stopping MQIPT

You can start MQIPT either from the command line, or make it start automatically when the system is started. You can stop MQIPT by using the **mqiptAdmin** command.

Starting MQIPT from the command line

MQIPT is installed into an installation directory, such as:

- Windows `C:\MQIPT` on Windows systems, with executable scripts in `C:\MQIPT\bin`
- Linux AIX `/opt/mqipt` on AIX and Linux systems, with executable scripts in `/opt/mqipt/bin`

MQIPT also uses a home directory, which contains the configuration file `mqipt.conf` and any files that are output by MQIPT when it is running. The following subdirectories of the MQIPT home directory are created automatically when MQIPT is invoked for the first time:

- An `errors` directory in which any First Failure Support Technology (FFST) and trace files are written
- A `logs` directory in which the connection log is kept

The user ID under which MQIPT runs must have permission to create these directories, or alternatively the directories must already exist and the user ID must have permission to create, read, and write files in them.

You can use the installation directory as a home directory. If you use this directory, you must ensure that the user ID under which MQIPT runs has the appropriate permissions.

To start MQIPT, use the **mqipt** command, which is located in the `bin` directory of the MQIPT installation directory. For example, the following command starts an instance of MQIPT that uses the directory `C:\mqiptHome` as the home directory:

```
mqipt C:\mqiptHome
```

For more information about the **mqipt** command, see mqipt (start MQIPT).

You can use the **mqipt** command to specify a name to be given to the MQIPT instance that is being started. The name of the MQIPT instance is used to administer local instances of MQIPT with the **mqiptAdmin** command without needing to use a command port. If this parameter is not specified, the name of the MQIPT home directory is used as the name of the MQIPT instance.

Console messages show the status of MQIPT. If an error occurs, see Troubleshooting IBM MQ Internet Pass-Thru. The following messages are an example of the output when MQIPT starts successfully:

```
5724-H72 (C) Copyright IBM Corp. 2000, 2026. All Rights Reserved
MQCPI001 IBM MQ Internet Pass-Thru 9.4.0.0 starting
MQCPI004 Reading configuration information from mqipt.conf
MQCPI152 MQIPT name is C:\mqiptHome
MQCPI021 Password checking has been enabled on the command port
```

```
MQCPI144 MQ Advanced capabilities not enabled
MQCPI011 The path C:\mqiptHome\logs will be used to store the log files
MQCPI006 Route 1414 is starting and will forward messages to :
MQCPI034 ....examplehost(1414)
MQCPI035 ....using MQ protocols
MQCPI057 ....trace level 5 enabled
MQCPI078 Route 1414 ready for connection requests
```

Starting MQIPT automatically

You can install MQIPT as a system service that starts automatically when the system is started. Use the **mqiptService** command to install and uninstall the MQIPT service.

- Windows On Windows systems, the **mqiptService** command installs MQIPT as a Windows service.
- Linux AIX On AIX and Linux systems, the **mqiptService** command installs MQIPT as a System V init service that starts when the system boots.

  Note: Linux

  1. Deprecated The MQIPT service on Linux is deprecated. The service on Linux uses System V init, which is not supported on some recent versions of Linux.
  2. The MQIPT service is not supported on the following versions of Linux which do not support System V init:
     - Red Hat Enterprise Linux 9.
     - SUSE Linux Enterprise Server 15.

     On these versions of Linux, use another method, such as `systemd`, to manage MQIPT as a service.

When the MQIPT service is started, all active MQIPT routes start. When the service is stopped, all routes are subjected to immediate shutdown.

You can only install one MQIPT service on a system, even if there is more than one installation of MQIPT on the system.

For more information about the **mqiptService** command, see mqiptService (manage the MQIPT service).

Stopping MQIPT

You can stop MQIPT by using the **mqiptAdmin** command with the **-stop** parameter.

For example, the following command stops an instance of MQIPT with the name `mqipt1` that is running locally under the same user ID as the **mqiptAdmin** command:

```
mqiptAdmin -stop -n ipt1
```

The **mqiptAdmin** command connects to the active instance of MQIPT to administer using one of the following methods:

- by connecting to a local instance of MQIPT without using the command port.
- by making a network connection to a command port.

Remote shutdown must be enabled by setting the **RemoteShutDown** property to `true` before the **mqiptAdmin** command can be used to stop MQIPT by sending a command to a command port.

For more information about administering MQIPT using the **mqiptAdmin** command, see “Administering MQIPT by using the command line” on page 561.

Specifying the password encryption key

If the MQIPT configuration contains passwords that are encrypted using an encryption key other than the default key, you must provide the password encryption key in a file that MQIPT can read when it starts.

The password encryption key file

Passwords that are encrypted to be stored and used by MQIPT can be encrypted using an encryption key that you provide. If you do not provide an encryption key, the default encryption key is used. You do not have to specify a password encryption key, however it is more secure to do so.

CAUTION: The default encryption key is the same for all MQIPT installations. To protect passwords securely, supply an encryption key that is unique to your installation.

If you provide a password encryption key, it must be stored in a file that can be accessed by the **mqiptPW** command used to encrypt passwords and MQIPT. The only restrictions on the contents of the file are that it must contain at least one character, and only one line of text.

Note: You must ensure that appropriate file permissions are set on the password encryption key file to prevent any unauthorized users from reading the encryption key. Only the user that runs the **mqiptPW** command and the user under which MQIPT runs need authority to read the password encryption key.

The same password encryption key is used to encrypt and decrypt all stored passwords for an instance of MQIPT. Therefore, you need only a single password encryption key file for each MQIPT installation.

If the password encryption key for an MQIPT installation is changed, all encrypted passwords must be re-encrypted using the new encryption key.

Starting MQIPT

The default name of the password encryption key file is *MQIPT_HOME_DIR*/mqipt_cred.key, where *MQIPT_HOME_DIR* is the directory where the mqipt.conf configuration file is stored. If you are planning to run MQIPT as a service that is automatically started, you must create the password encryption key file with the default name.

If the password encryption key file is created with a name other than the default name, the name of the file must be provided to MQIPT when it is started. The name of the password encryption key file can be specified using any of the following methods, in order of preference:

1. the **-sf** parameter on the **mqipt** command used to start MQIPT.
2. the MQS_MQIPTCRED_KEYFILE environment variable.
3. the com.ibm.mq.ipt.cred.keyfile Java property.

If no password encryption key file name is provided, the default file name will be used, if the file exists. If the default password encryption key file does not exist, the default password encryption key is used.

Administering MQIPT by using the command line

You can use the **mqiptAdmin** command on the command line to administer MQIPT.

You can use the **mqiptAdmin** command to perform the following administrative functions:

- List active local instances of MQIPT.
- Refresh an instance of MQIPT after you make changes to the configuration file.
- Stop an instance of MQIPT.

The mqiptAdmin command is located in the bin subdirectory of the MQIPT installation directory.

The **mqiptAdmin** command connects to the active instance of MQIPT to administer using one of the following methods:

- by making a network connection to a command port.
- by connecting to a local instance of MQIPT without using the command port.

The **mqiptAdmin** command is compatible with previous versions of MQIPT, but you cannot use the command to administer versions of MQIPT that are a higher version than the version of the **mqiptAdmin** command. In an environment that includes different versions of MQIPT, you must use the latest version of the **mqiptAdmin** command.

For more information about the syntax of the **mqiptAdmin** command, see mqiptAdmin (administer MQIPT).

Local administration without a command port

Local instances of MQIPT can be administered without using a command port. Local administration allows you to administer MQIPT by using the **mqiptAdmin** command only when it is run on the same system as the MQIPT instance that you want to administer.

In order for **mqiptAdmin** to be authorized to administer a local instance of MQIPT without using the command port, the MQIPT instance must be running on the same system and under the same user ID as **mqiptAdmin**. Alternatively, on AIX and Linux, **mqiptAdmin** can be run as `root`.

Local administration is enabled by default. To disable local administration, use the **LocalAdmin** configuration property. For more information about the **LocalAdmin** property, see LocalAdmin.

To administer local instances of MQIPT, you must give each instance a name. You can assign a name to an instance of MQIPT by using the **-n** parameter when starting MQIPT with the **mqipt** command. If you do not specify a name when starting MQIPT, the name of the home directory is used as the name of the MQIPT instance. For example, the following command starts MQIPT and assigns the name `ipt1` to the instance:

```
mqipt /opt/mqipt1 -n ipt1
```

Once the instance has a name, you can administer that instance by specifying the name in the **mqiptAdmin** command with the **-n** parameter. For example, the following command stops the local instance of MQIPT with the name `ipt1`:

```
mqiptAdmin -stop -n ipt1
```

You can list all local active instance of MQIPT that the **mqiptAdmin** command is authorized to administer without using a command port by using the **mqiptAdmin** command with the **-list** parameter. For example, the following command lists all local active instances of MQIPT that the user that started the **mqiptAdmin** command is authorized to administer:

```
mqiptAdmin -list
```

Administration using a command port

You can configure MQIPT with one command port that is unsecured and one command port that is secured with TLS. You can use these command ports to administer MQIPT as any user that is on the same system as the MQIPT instance that you want to administer, or from a remote system.

Previous versions of MQIPT only accepted administrative commands issued to the unsecured command port.

Note: Connections to the unsecured command port are not encrypted, therefore data sent over the network to the unsecured command port, including the MQIPT access password, can be visible to other users on the network.

In order for MQIPT to listen on a command port for commands issued by the **mqiptAdmin** command, a value must specified for either the **CommandPort** or **SSLCommandPort** properties in the global section of the `mqipt.conf` configuration file.

Review the security considerations in Other security considerations before enabling either of the MQIPT command ports. Consider enabling authentication for commands received by the command ports. For

more information about command port authentication, see “Command port authentication” on page 566.

To administer an instance of MQIPT using a command port, specify the network address of the host where MQIPT is running, and the command port number, as parameters to the **mqiptAdmin** command. For example, to refresh the MQIPT instance that is running on `mqipt.example.com`, and has the unsecured command port configured to listen on port 1890, issue the following command:

```
mqiptAdmin -refresh -r mqipt.example.com:1890
```

If you do not specify the host name and port number, **mqiptAdmin** attempts to connect to `localhost`, port 1881.

For more information about administering MQIPT by using the TLS command port, see “Administering MQIPT using the TLS command port” on page 563.

Administering MQIPT using the TLS command port

MQIPT can be configured to use a TLS command port to listen for administrative commands issued by the **mqiptAdmin** command. Using the TLS command port protects sensitive data such as the MQIPT access password on the network between **mqiptAdmin** and MQIPT. Use this procedure to configure the TLS command port and administer MQIPT using the TLS command port.

About this task

The TLS command port must be configured with a server certificate stored either in a PKCS #12 keystore, or in cryptographic hardware that supports the PKCS #11 Cryptographic Token Interface. The command port server certificate is sent to the **mqiptAdmin** command during the TLS handshake. This task assumes that you request a new server certificate from a trusted Certificate Authority (CA), and that the certificate is returned to you in a file. The **mqiptAdmin** command validates the command port certificate using the CA certificate of the CA that signed the server certificate. The CA certificate must be stored in a PKCS #12 keystore that can be accessed by the **mqiptAdmin** command.

Client certificate authentication is not supported by the TLS command port. To enable authentication for administrative commands issued to a command port, see “Command port authentication” on page 566.

This procedure describes how to manage the keystores and digital certificates that are required to use the TLS command port by using the **mqiptKeytool** command. For more information about managing keystores that MQIPT uses, see Managing MQIPT keystores.

Procedure

1. Follow these steps to configure the TLS command port for the instance of MQIPT.
   a) Create a public and private key pair, and an associated TLS command port server certificate in a PKCS #12 keystore.

      To create the keystore that contains TLS command port server certificate, enter the following command:

      ```
      mqiptKeytool -genkeypair -keystore filename -storetype pkcs12 -storepass password
                   -dname distinguished_name -alias label
                   -keyalg key_algorithm -keysize key_size -sigalg sig_algorithm
      ```

      where:

      -keystore *filename*
      Specifies the keystore name.

      -storepass *password*
      Specifies the keystore password.

      -alias *label*
      Specifies the certificate label.

-keyalg *key_algorithm*
Specifies the algorithm that is used to create the key pair.

-keysize *key_size*
Specifies the key size.

-sigalg *algorithm*
Specifies the algorithm that is used to sign the certificate.

-dname *distinguished_name*
Specifies the X.500 distinguished name enclosed in double quotation marks.

b) Create a certificate request for the CA signed TLS command port server certificate.

To create a certificate request, enter the following command:

```
mqiptKeytool -certreq -keystore filename -storetype pkcs12 -storepass password
             -alias label -file certreq_filename
```

where:

-keystore *filename*
Specifies the keystore name.

-storepass *password*
Specifies the keystore password.

-alias *label*
Specifies the certificate label.

-file *certreq_filename*
Specifies the file name for the certificate request.

c) Send the certificate request file created in step "1.b" on page 564 to your CA to be signed.

d) After the CA sends you the signed certificate, receive the signed certificate into the keystore.

To receive the signed certificate into the keystore, enter the following command:

```
mqiptKeytool -importcert -keystore cert_filename -storetype pkcs12 -storepass password
             -file cert_filename
```

where *cert_filename* is the name of the file that contains the certificate, *filename* is the name of the keystore, and *password* is the keystore password.

e) Encrypt the keystore password using the **mqiptPW** command.

Enter the following command:

```
mqiptPW -sf encryption_key_file
```

where *encryption_key_file* is the name of a file that contains the password encryption key for your MQIPT installation. You do not need to specify the **-sf** parameter if your MQIPT installation is using the default password encryption key. Type in the keystore password to encrypt when prompted.

For more information about the **mqiptPW** command, see Encrypting a key ring password.

f) Edit the `mqipt.conf` configuration file and specify the following properties to configure the TLS command port:

i) Set the value of the **SSLCommandPort** property to the TLS command port number.

ii) Set the value of the **SSLCommandPortKeyRing** property to the file name of the keystore created in step "1.a" on page 563.

iii) Set the value of the **SSLCommandPortKeyRingPW** to the string output by the **mqiptPW** command in step "1.e" on page 564.

iv) Set the value of the **SSLCommandPortSiteLabel** property to the label name of the TLS command port certificate, specified when creating the certificate request in step "1.b" on page 564.

v) If you want to restrict inbound connections to the TLS command port to those from a particular network interface, set the value of the **SSLCommandPortListenerAddress** property to a

network address belonging to one of the network interfaces on the system where MQIPT is running. For example, to restrict inbound connections to the TLS command port to those only from the local machine, set the value of the **SSLCommandPortListenerAddress** property to `localhost`.

g) Start or refresh MQIPT to enable the TLS command port.

MQIPT issues console messages such as the following to display the TLS command port configuration that is in effect:

```
MQCPI155 Listening for control commands on port 1882 on local address * using TLS
MQCPI139 ......secure socket protocols <NULL>
MQCPI031 ......cipher suites <NULL>
MQCPI032 ......key ring file c:\\iptHome\\ssl\\commandport.p12
MQCPI072 ......and certificate label mqiptadmin
```

2. On the system where the **mqiptAdmin** command is used to administer MQIPT, follow these steps to enable **mqiptAdmin** to connect to the TLS command port.

a) Import the CA certificate of the CA that signed the TLS command port certificate into a PKCS #12 keystore to be used as a trust store by the **mqiptAdmin** command.

To import the CA certificate, enter the following command:

```
mqiptKeytool -importcert -keystore filename -storetype pkcs12 -storepass password
             -file cert_filename -alias certlabel
```

where:

filename
Specifies the name of the keystore to create

password
Specifies the keystore password

certlabel
Specifies the label to be given to the CA certificate

cert_filename
Specifies the name of the file that contains the CA certificate

b) Encrypt the keystore password by using the **mqiptPW** command.

Enter the following command:

```
mqiptPW -sf encryption_key_file
```

where *encryption_key_file* is the name of the file that contains the password encryption key. The password encryption key file can be different to the one used to encrypt passwords in the MQIPT configuration. The default password encryption key is used if you do not specify an encryption key file with the **-sf** parameter. Type in the keystore password to encrypt when prompted.

For more information about the **mqiptPW** command, see Encrypting a key ring password.

c) Create a properties file to be used by the **mqiptAdmin** command and specify the following properties:

```
SSLClientCAKeyRing=key_ring_file_name
SSLClientCAKeyRingPW=key_ring_password
PasswordProtectionKeyFile=encryption_key_file
```

where:

key_ring_file_name
is the name of the keystore created in step “2.a” on page 565.

key_ring_password
is the encrypted password output by the **mqiptPW** command in step “2.b” on page 565.

encryption_key_file
is the name of the file that contains the password encryption key. You need to specify the **PasswordProtectionKeyFile** property only if an encryption key file was used to encrypt the keystore password in step "2.b" on page 565.

d) Issue the **mqiptAdmin** command to administer MQIPT, specifying the **-s** parameter to indicate that a TLS connection is required, and the **-p** parameter to specify the name of the properties file that was created in step "2.c" on page 565.

For example, enter the following command to refresh an instance of MQIPT by sending a refresh command to the TLS command port:

```
mqiptAdmin -refresh -r hostname:port -s -p properties_file
```

The **mqiptAdmin** command issues a message such as the following to confirm that the connection to MQIPT is protected with TLS:

```
MQCAI109 The connection to MQIPT is secured with TLSv1.2.
```

What to do next

To enable authentication for commands received by the TLS command port, follow the steps in "Command port authentication" on page 566.

Command port authentication

MQIPT can be configured to authenticate commands received by the unsecured command port and TLS command port using a password. Use this procedure to enable command port authentication.

About this task

The **mqiptAdmin** command prompts users to enter a password when the command connects to the command port of an instance of MQIPT that has command port authentication enabled. MQIPT validates the password entered in the **mqiptAdmin** command against the access password specified in the MQIPT configuration.

The properties that you set for command port authentication apply to both the TLS command port and the unsecured command port.

Procedure

1. Encrypt the MQIPT access password using the **mqiptPW** command.

   Enter the following command:

   ```
   mqiptPW -sf encryption_key_file
   ```

   where *encryption_key_file* is the name of the file that contains the password encryption key for your MQIPT installation. You do not need to specify the **-sf** parameter if your MQIPT installation is using the default password encryption key. Type in the access password to encrypt when prompted.

   For more information about encrypting passwords in the MQIPT configuration, see Encrypting stored passwords.

2. Edit the `mqipt.conf` configuration file and specify the following properties:

   ```
   AccessPW=encrypted_password
   RemoteCommandAuthentication=auth_setting
   ```

   where:

   encrypted_password
   is the encrypted password output by the **mqiptPW** command in step "1" on page 566.

auth_setting
is the authentication requirement. Command port authentication is enabled if this property is set to one of the following values:

optional
A password is not required, but if a password is provided it must be valid. This option might be useful during migration, for example.

required
A valid password must be provided with each command received by a command port.

For more information about these properties, see MQIPT global properties.

3. Start or refresh MQIPT to bring the changes into effect.

   MQIPT issues a message indicating whether command port authentication is enabled. For example, if MQIPT is configured to require a valid password to be entered each time the **mqiptAdmin** command is run, then the following message is issued:

```
MQCPI021 Password checking has been enabled on the command port
```

Making backups

There are a number of MQIPT files that you should back up as part of your regular backup procedures.

Back up the following files on a regular basis:

- The configuration file, `mqipt.conf`
- The SSL/TLS key ring files specified by the following properties in `mqipt.conf`:
  - **SSLClientKeyRing**
  - **SSLClientCAKeyRing**
  - **SSLServerKeyRing**
  - **SSLServerCAKeyRing**
  - **SSLCommandPortKeyRing**
- The SSL/TLS key ring password files specified by the following properties in `mqipt.conf`:
  - **SSLClientKeyRingPW**
  - **SSLClientCAKeyRingPW**
  - **SSLServerKeyRingPW**
  - **SSLServerCAKeyRingPW**
- The password encryption key file, if the MQIPT configuration contains passwords that are encrypted with an encryption key other than the default key.
- The security exit files and certificate exit files specified by the following properties in `mqipt.conf`:
  - **SecurityExitName**
  - **SSLExitName**
- Connection log files in the `log` subdirectory of the MQIPT home directory, if these are needed for audit purposes.

Performance tuning

You can tune the relative performance of each MQIPT route by using a combination of a thread pool and an idle timeout specification.

Connection threads

Each MQIPT route is assigned a working pool of concurrently running threads that handle incoming communication requests. At initialization, a pool of threads is created (of the size specified in the route's `MinConnectionThreads` attribute), and a thread is assigned to handle the first incoming request. When this request arrives, another thread is assigned, ready for the next incoming request. When all threads are assigned for work, a new thread is created, added to the working pool, and assigned for work.

In this way, the pool grows until the maximum number of threads (specified in **`MaxConnectionThreads`**) is reached. Threads are released back to the pool when a conversation ends, or the specified idle timeout period has elapsed. When the maximum number of working threads is reached, the next incoming request waits until a thread is released back to the working pool.

You can reduce the time that requests might have to wait by increasing the number of available threads. However, you must balance this increase with the system resources that are available.

Idle timeout

By default, working threads are not terminated because of inactivity. When a thread has been assigned to a conversation, it remains assigned to that conversation until it is closed normally, the route is deactivated, or MQIPT is shut down. Optionally, you can specify an idle timeout interval (in minutes) in the **`IdleTimeout`** property so that threads that have been inactive for the specified period of time are recycled. Threads are recycled for use by placing them back into the working pool.

If IBM MQ activity is intermittent, set its heartbeat interval to a value less than that of the MQIPT timeout so that threads are not constantly recycled.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document does not grant 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

© Copyright IBM Corp. 2007, 2026

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 Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (https://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Part Number:

(1P) P/N: