8.0

Monitoring and Performance for IBM MQ

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 355.

This edition applies to version 8 release 0 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Monitoring and performance

Use the monitoring information and guidance in this section, and the specific tuning tips, to help improve the performance of your queue manager network.

Depending on the size and complexity of your queue manager network, you can obtain a range of information from monitoring the network. You can use that information, along with the information provided in specific tuning tips, to help you tune your network performance.

Entities for MQTT clients

MQTT names for client pack, clients, and client sample apps

machine-to-machine (M2M)
Mobile Messaging and M2M
Mobile Messaging and M2M Client Pack
IBM Messaging Telemetry Clients
IBM Messaging Telemetry Clients SupportPac
MQTT client for Java
MQTT messaging client for JavaScript
MQTT client for C
MQTT client sample Java app
MQTT client sample Java app for Android
MQTT messaging client sample JavaScript pages
MQTT client sample C app

MQTT phrases, xrefs, filepaths

The examples are `MQTTV3ASample.c` and `MQTTV3ASSample.c` in *sdkroot*`\SDK\clients\c\samples`.
MQTT build options for different platforms
Install a C development environment on the platform on which you are building.
System requirements for IBM Messaging Telemetry Clients SupportPac
The client identifier must be unique across all clients that connect to the server, and must not be the same as the queue manager name on the server.
The configuration gives everyone permission to publish and subscribe to any topic. The configuration of security and access control is minimal and is intended only for a queue manager that is on a secure network with restricted access. To run IBM WebSphere® MQ and MQTT in an insecure environment, you must configure security. To configure security for IBM WebSphere MQ and MQTT, see the related links at the end of this task.
If you use the default `MqttConnectOptions`, or set `MqttConnectOptions.cleanSession` to `true` before connecting the client, any old subscriptions for the client are removed when the client connects. Any new subscriptions the client makes during the session are removed when it disconnects.
If you set `MqttConnectOptions.cleanSession` to `false` before connecting, any subscriptions the client creates are added to all the subscriptions that existed for the client before it connected. All the subscriptions remain active when the client disconnects.
Another way of understanding the way the `cleanSession` attribute affects subscriptions is to think of it as a modal attribute. In its default mode, `cleanSession=true`, the client creates subscriptions and receives publications only within the scope of the session. In the alternative mode, `cleanSession=false`, subscriptions are durable. The client can connect and disconnect and its subscriptions remain active. When the client reconnects, it receives any undelivered publications. While it is connected, it can modify the set of subscriptions that are active on its behalf.
You must set the `cleanSession` mode before connecting; the mode lasts for the whole session. To change its setting, you must disconnect and reconnect the client. If you change modes from using `cleanSession=false` to `cleanSession=true`, all previous subscriptions for the client, and any

publications that have not been received, are discarded.
Example scripts to configure SSL certificates for Windows
Change *sdkroot* to suit your environment.
Install a Java development kit (JDK) at Version 7 or later.
Version 7 is required to run the **keytool** command to certify certificates. If you are not going to certify certificates, you do not require the Version 7 JDK.

```
# Build output is produced in the current directory.
# MQTTCLIENT_DIR must point to the base directory containing the MQTT client source code.
# Default MQTTCLIENT_DIR is the current directory
# Default TOOL_DIR is /Applications/Xcode.app/Contents/Developer/Platforms
# Default OPENSSL_DIR is sdkroot/openssl, relative to sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR must point to the base directory containing the OpenSSL build.
# Example:  make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
ifndef TOOL_DIR
  TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}
MQTTLIB = mqttv3c
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB}.a
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S}.a
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c,
${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A}.a
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS}.a
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
CCFLAGS = -Os -Wall -fomit-frame-pointer
CCFLAGS_SO_ARM =  ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include  -L$
{SDK_ARM}/usr/lib/system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include -L$
{SDK_i386}/usr/lib/system
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}
${MQTTLIB_DARWIN}: ${SOURCE_FILES}
   -mkdir darwin_x86_64
   ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
   libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
   rm *.o
   ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
   libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
   rm *.o
   ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
   libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
   rm *.o
   lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
```

```
	-mkdir darwin_x86_64
	${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
	rm *.o
	${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
	rm *.o
	${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
	rm *.o
	lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
	-mkdir darwin_x86_64
	${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
	rm *.o
	${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s *.o
	rm *.o
	${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
	rm *.o
	lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
	-mkdir darwin_x86_64
	${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
	rm *.o
	${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s *.o
	rm *.o
	${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
	libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
	rm *.o
	lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
.PHONY : clean
clean:
	-rm -f *.obj
	-rm -f -r darwin_x86_64
```

Download and install the iOS development tools.

For links to client API documentation for the MQTT client libraries, see MQTT client programming reference.

You can run an MQTT client for Java app on any platform with JSE 1.5 or above that is "Java Compatible"

Log on with a user ID that has administrative authority to IBM WebSphere MQ.

The server is now ready for you to test your MQTT V3.1 app.

You must have access to an MQTT Version 3.1 server that supports the MQTT protocol over SSL.

If there is a firewall between your client and the server, check that it does not block MQTT traffic.

You can run an MQTT client for Java app on any platform with JSE 1.5 or above that is "Java Compatible". See System requirements for IBM Messaging Telemetry Clients SupportPac.

The SSL channels must be started.
Choose an MQTT server to which you can connect the client app.
The statement `rm *.o` deletes all the object files that are created for each library. `lipo` concatenates all three libraries into one file.
Download the IBM Messaging Telemetry Clients SupportPac.
Download the IBM Messaging Telemetry Clients SupportPac and install the MQTT SDK.
either IBM MessageSight or IBM WebSphere MQ as the MQTT server
The server must support the MQTT Version 3.1 protocol.
The server must support the MQTT Version 3.1 protocol over SSL.
The MQTT Paho sample applications in the SDK are set up to connect to the Eclipse M2M server by default. See MQTT Sandbox Server. You do not have to configure a server to try the Paho sample applications out.
Create and run the scripts to generate key-pairs and certificates, and configure IBM WebSphere MQ as the MQTT server.
Check the SSL channels are running and are set up as you expect.
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.
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.
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.
By providing the client with a private signed digital certificate, you can authenticate the MQTT client to WebSphere MQ. The WebSphere MQ Administrator can force MQTT clients to authenticate themselves to the queue manager using SSL. You can only request client authentication as part of mutual authentication.
Client authentication using SSL 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.
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.
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. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.
The client JVM must use the standard socket factory from JSSE. If you are using Java ME, you must ensure that the JSSE package is loaded. If you are using Java SE, JSSE has been included with the JRE since Java version 1.4.1.
The SSL connection requires a number of SSL properties to be set before connecting. You can set the properties either by passing them to the JVM using the `-D` switch, or you can set the properties using the `MqttConnectionOptions.setSSLProperties` method.
If you load a non-standard socket factory, by calling the method `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, then the way SSL settings are passed to the network socket is application defined.
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 `InstallPath\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 SSL, configure the server, and the client.
When `SSLCIPH` is used with a telemetry channel, it means "SSL Cipher Suite".
The SSL cipher suite is the one supported by the JVM that is running the telemetry (MQXR) service.
If the `SSLCIPH` parameter is blank, no attempt is made to use SSL on the channel.
`SSLCIPH` specifies the CipherSpec that is used on the channel. The maximum length is 32 characters. This parameter is valid on all channel types which use transport type TRPTYPE`(TCP)`.
Specify the name of the CipherSpec you are using. The CipherSpecs that can be used with IBM MQ SSL support are shown in the following table. If a specific named CipherSpec is being used, the **`SSLCIPH`** values at the two ends of a channel must specify the same named CipherSpec.
On z/OS IBM MQ for z/OS®; IBM i IBM i; you can also specify the two digit hexadecimal code of a CipherSpec, whether or not it appears in the table. On IBM i, installation of AC3 is a prerequisite for the use of SSL.
If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.
The value for this parameter is also used to set the value of SECPROT
Download and redistribution of the OpenSSL package is subject to stringent import and export regulation, and open source licensing conditions. Take careful note of the restrictions and warnings before you decide whether to download the package.
For the C samples, the store is a Privacy-Enhanced Mail (PEM) file. For the Java samples it is a Java keystore (JKS).
Create a script in the client samples directory to compile and run `Sample` on your chosen platform.
Install a Java development kit (JDK) Version 6 or later.
Because you are developing a Java app for Android, the JDK must come from Oracle. You can get the JDK from Java SE Downloads.
Each successive line that defines the implementation of a target must start with a tab character.
The default file persistence class in the Java SE MQTT client supplied with IBM MQ Telemetry creates a folder with the name: *clientIdentifier*-tcp*hostNameport* or *clientIdentifier*-ssl*hostNameport* in the client

working directory. The folder name tells you the `hostName` and `port` used in the connection attempt
Open the MQTT client sample Java app for Android.
Connect to an MQTT server.

```
# Build output is produced in the current directory.
# MQTTCLIENT_DIR must point to the base directory containing the MQTT client source code.
# Default MQTTCLIENT_DIR is the current directory
# Default OPENSSL_DIR is sdkroot\openSSL, relative to sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR must point to the base directory containing the OpenSSL build.
# Example:  make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Set the build environment, for example:
#    %comspec% /k ""C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
#    set path=%path%;C:\Program Files\GnuWin32\bin;C:\cygwin\bin
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
MQTTLIB = mqttv3c
MQTTDLL = windows_ia32/${MQTTLIB}.dll
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource:${MQTTDLL}\;2
MQTTLIB_S = mqttv3cs
MQTTDLL_S  = windows_ia32/${MQTTLIB_S}.dll
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource:${MQTTDLL_S}\;2
MQTTLIB_A = mqttv3a
MQTTDLL_A  = windows_ia32/${MQTTLIB_A}.dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c,
${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource:${MQTTDLL_S}\;2
MQTTLIB_AS = mqttv3as
MQTTDLL_AS  = windows_ia32/${MQTTLIB_AS}.dll
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource:${MQTTDLL_S}\;2
CC = cl
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/
LD = link
LINKFLAGS = /nologo /machine:x86 /manifest /dll
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
        advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
        odbc32.lib odbccp32.lib ws2_32.lib
IMP = /implib:${@:.dll=.lib}
LIBPDB = /pdb:${@:.dll=.pdb}
LIBMAP = /map:${@:.dll=.map}
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib
${MQTTDLL}: ${SOURCE_FILES}
   -mkdir windows_ia32
   -rm ${CURDIR}/MQTTAsync.obj
   ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
```

${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL}
${MANIFEST}
${MQTTDLL_A}: ${SOURCE_FILES_A}
-mkdir windows_ia32
-rm ${CURDIR}/MQTTClient.obj
${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL_A}
${MANIFEST_A}
${MQTTDLL_S}: ${SOURCE_FILES_S}
-mkdir windows_ia32
-rm ${CURDIR}/MQTTAsync.obj
${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
{MQTTDLL_S}
${MANIFEST_S}
${MQTTDLL_AS}: ${SOURCE_FILES_AS}
-rm ${CURDIR}/MQTTClient.obj
${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
(MQTTDLL_AS}
$(MANIFEST_AS}
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}
.PHONY : clean
clean:
-rm -f *.obj
-rm -f -r windows_ia32

This step is optional on Windows because you can administer IBM WebSphere MQ as a Windows administrator.

Set the location of the MQTT source code.

Run the makefile in the same directory as the MQTT source files, or set the `MQTTCLIENT_DIR` command-line parameter:

make -f makefile MQTTCLIENT_DIR=*sdkroot*/SDK/clients/c/mqttv3c/src

Add the following lines to the makefile:

The example sets VPATH to the directory where **make** searches for source files that are not explicitly identified; for example all the header files that are required in the build.

Set the location of the OpenSSL libraries.

This step is required to build the SSL versions of the MQTT client for C libraries.

Set the default path to the OpenSSL libraries to same directory as you expanded the MQTT SDK. Otherwise, set `OPENSSL_DIR` as a command-line parameter.

OpenSSL is the OpenSSL directory that contains all the OpenSSL subdirectories. You might have to move the directory tree from where you expanded it, because it contains unnecessary empty parent directories.

Select all the source files that are required to build each MQTT library. Also, set the name and location of the MQTT library to build.

The source files depend on whether you are building a synchronous or asynchronous library, and whether the library includes SSL or not.

Add the following line to the makefile to list all the MQTT source files:

Set the keystore location and attributes with MQ Explorer, or with the **`DEFINE CHANNEL`** command; see DEFINE CHANNEL (MQTT). Multiple channels can share a keystore.

MQTT non-phrases

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

`SampleAsyncCallBack`

`SampleAsyncCallBack` is in the `org.eclipse.paho.client.mqttv3` package. It calls the asynchronous MQTT API. The asynchronous API does not wait for MQTT to complete processing a call; it returns to the application. The application carries on with other tasks, then waits for the next event to arrive for it to process. MQTT posts an event notification back to the application when it completes processing. The event driven MQTT interface is suited to the service and activity programming model of Android and other event driven operating systems.

As an example, look at how the `mqttExerciser` sample integrates MQTT into Android using the service and activity programming model.

SampleAsyncWait
`SampleAsyncWait` is in the `org.eclipse.paho.client.mqttv3` package. It uses the asynchronous MQTT API; it waits on a different thread until an action completes. The main thread can do other work until it synchronizes on the thread that is waiting for the MQTT action to complete.

The example command files create the certificates and certificate stores as described in the steps in the task. In addition, the example sets up the MQTT client queue manager to use the server certificate store. The example deletes and re-creates the queue manager by calling the `SampleMQM.bat` script that is provided with IBM WebSphere MQ.

initcert.bat
`initcert.bat` sets the names and paths to certificates and other parameters that are required by the **keytool** and **openSSL** commands. The settings are described in comments in the script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
```

cleancert.bat

The commands in the `cleancert.bat` script delete the MQTT client queue manager to ensure that the server certificate store is not locked, and then delete all the keystores and certificates that are created by the sample security scripts.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

Figure 2. `cleancert.bat`

genkeys.bat

The commands in the `genkeys.bat` script create key-pairs for your private certificate authority, the server, and a client.

```
@rem
@echo _________________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

Figure 3. `genkeys.bat`

sscerts.bat

The commands in the `sscerts.bat` script export the client and server self-signed certificates from their keystores, and import the server certificate into the client truststore, and the client certificate into the server keystore. The server does not have a truststore. The commands create a client truststore in PEM format from the client JKS truststore.

```
@rem
@echo ________________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ________________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ________________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

Figure 4. `sscerts.bat`

`cacerts.bat`

The script imports the certificate authority root certificate into the private keystores. The CA root certificate is needed to create the keychain between the root certificate and the signed certificate. The `cacerts.bat` script exports the client and server certificate requests from their keystores. The script signs the certificate requests with the key of the private certificate authority in the `cajkskeystore.jks` keystore, then imports the signed certificates back into the same keystores from which the requests came. The import creates the certificate chain with the CA root certificate. The script creates a client truststore in PEM format from the client JKS truststore.

```
@rem
@echo ________________________________________________________________________________
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ________________________________________________________________________________
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
```

mqcerts.bat

The script lists the keystores and certificates in the certificate directory. It then creates the MQTT sample queue manager and configures the secure telemetry channels.

```
@echo ________________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Figure 6. mqcerts.bat

Get a copy of the IBM WebSphere MQ installation materials and a license in one of the following ways:

1. Ask your IBM WebSphere MQ administrator for the installation materials, and to confirm you can accept the license agreement.
2. Get a 90-day evaluation copy of IBM WebSphere MQ. See Evaluate: IBM WebSphere MQ.
3. Buy IBM WebSphere MQ. See IBM WebSphere MQ product page.

Secure the SSL channel with either certificate authority signed keys, or self-signed keys.

There is no installation program, you just expand the downloaded file.

1. Download the IBM Messaging Telemetry Clients SupportPac.
2. Create a folder where you are going to install the SDK.

   You might want to name the folder MQTT. The path to this folder is referred to here as *sdkroot*.
3. Expand the compressed IBM Messaging Telemetry Clients SupportPac file contents into *sdkroot*. The expansion creates a directory tree that starts at *sdkroot*\SDK.

There is a similar limitation for the MQTT client for Java. If the client code is running on a Java 1.6 JRE from IBM, the required SHA-2 cipher suites must be explicitly enabled. In order to use these suites, the client must also set the SSL context to a value that supports Version 1.2 of the Transport Layer Security (TLS) protocol. For example:

```
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
java.util.Properties sslClientProps = new java.util.Properties();
sslClientProps.setProperty("com.ibm.ssl.keyStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.keyStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.trustStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.trustStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
sslClientProps.setProperty("com.ibm.ssl.enabledCipherSuites",
"SSL_RSA_WITH_AES_256_CBC_SHA256" );
mqttConnectOptions.setSSLProperties(sslClientProps);
```

On IBM WebSphere MQ, type the following command into a command window:

- Linux

```
echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
```

• Windows

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- On z/OS z/OS, Windows, UNIX and Linux® systems, when a CipherSpec name includes _EXPORT, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- On Windows, UNIX and Linux systems, when a CipherSpec name includes _EXPORT1024, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

Monitoring your IBM MQ network

A number of monitoring techniques are available in IBM MQ to obtain statistics and other specific information about how your queue manager network is running. Use the monitoring information and guidance in this section to help improve the performance of your queue manager network.

The following list provides examples of reasons for monitoring your queue manager network:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.
- Generate messages when certain events occur.
- Record message activity.
- Determine the last known location of a message.
- Check various statistics of a queue manager network in real time.
- Generate an audit trail.
- Account for application resource usage.
- Capacity planning.

Event monitoring

Event monitoring is the process of detecting occurrences of *instrumentation events* in a queue manager network. An instrumentation event is a logical combination of events that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an *event message*, on an event queue.

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. Use these events to monitor the operation of the queue managers in your queue manager network to achieve the following goals:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.

- Generate an audit trail.
- React to queue manager state changes

Related reference

“Event types” on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information

Event message reference
Event message format

Instrumentation events

An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can use these events to monitor the operation of queue managers (with other methods such as Tivoli® NetView® for z/OS).

Figure 7 on page 20 illustrates the concept of instrumentation events.

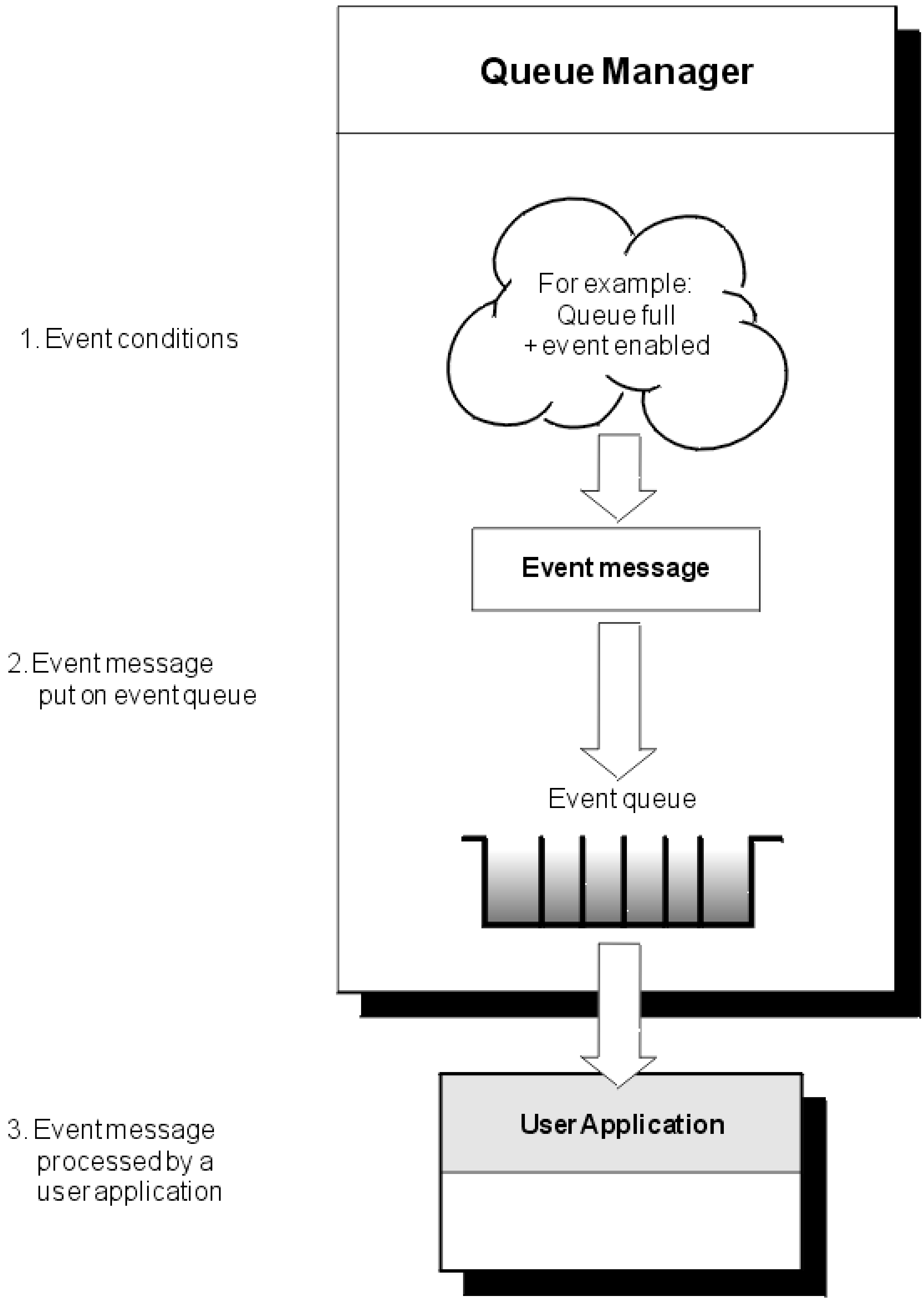


Figure 7. Understanding instrumentation events

Event monitoring applications

Applications that use events to monitor queue managers must include the following provisions:

1. Set up channels between the queue managers in your network.

2. Implement the required data conversions. The normal rules of data conversion apply. For example, if you are monitoring events on a UNIX system queue manager from a z/OS queue manager, ensure that you convert EBCDIC to ASCII.

Event notification through event queues

When an event occurs, the queue manager puts an event message on the appropriate event queue, if defined. The event message contains information about the event that you can retrieve by writing a suitable MQI application program that performs the following steps:

- Get the message from the queue.
- Process the message to extract the event data.

The related information describes the format of event messages.

Conditions that cause events

The following list gives examples of conditions that can cause instrumentation events:

- A threshold limit for the number of messages on a queue is reached.
- A channel instance is started or stopped.
- A queue manager becomes active, or is requested to stop.
- An application tries to open a queue specifying a user ID that is not authorized on IBM MQ for IBM i, Windows, UNIX and Linux systems.
- Objects are created, deleted, changed, or refreshed.
- An MQSC or PCF command runs successfully.
- A queue manager starts writing to a new log extent.
- Putting a message on the dead-letter queue, if the event conditions are met.

Related concepts

"Performance events" on page 33
Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

"Sample program to monitor instrumentation events" on page 64
V 8.0.0.4 **amqsevt** formats the instrumentation events that a queue manager can create, and is supplied with IBM MQ. The program reads messages from event queues, and formats them into readable strings.

Event types

Use this page to view the types of instrumentation event that a queue manager or channel instance can report

IBM MQ instrumentation events have the following types:

- Queue manager events
- Channel and bridge events
- Performance events
- Configuration events
- Command events
- Logger events
- Local events

For each queue manager, each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

This event queue:	Contains messages from:
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CONFIG.EVENT	Configuration events
SYSTEM.ADMIN.COMMAND.EVENT	Command events
SYSTEM.ADMIN.LOGGER.EVENT	Logger events
SYSTEM.ADMIN.PUBSUB.EVENT	Gets events related to Publish/Subscribe. Only used with Multicast. For more information see, Multicast application monitoring.

By incorporating instrumentation events into your own system management application, you can monitor the activities across many queue managers, across many different nodes, and for multiple IBM MQ applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support IBM MQ events) as shown inFigure 8 on page 22.

Instrumentation events can be reported through a user-written reporting mechanism to an administration application that can present the events to an operator.

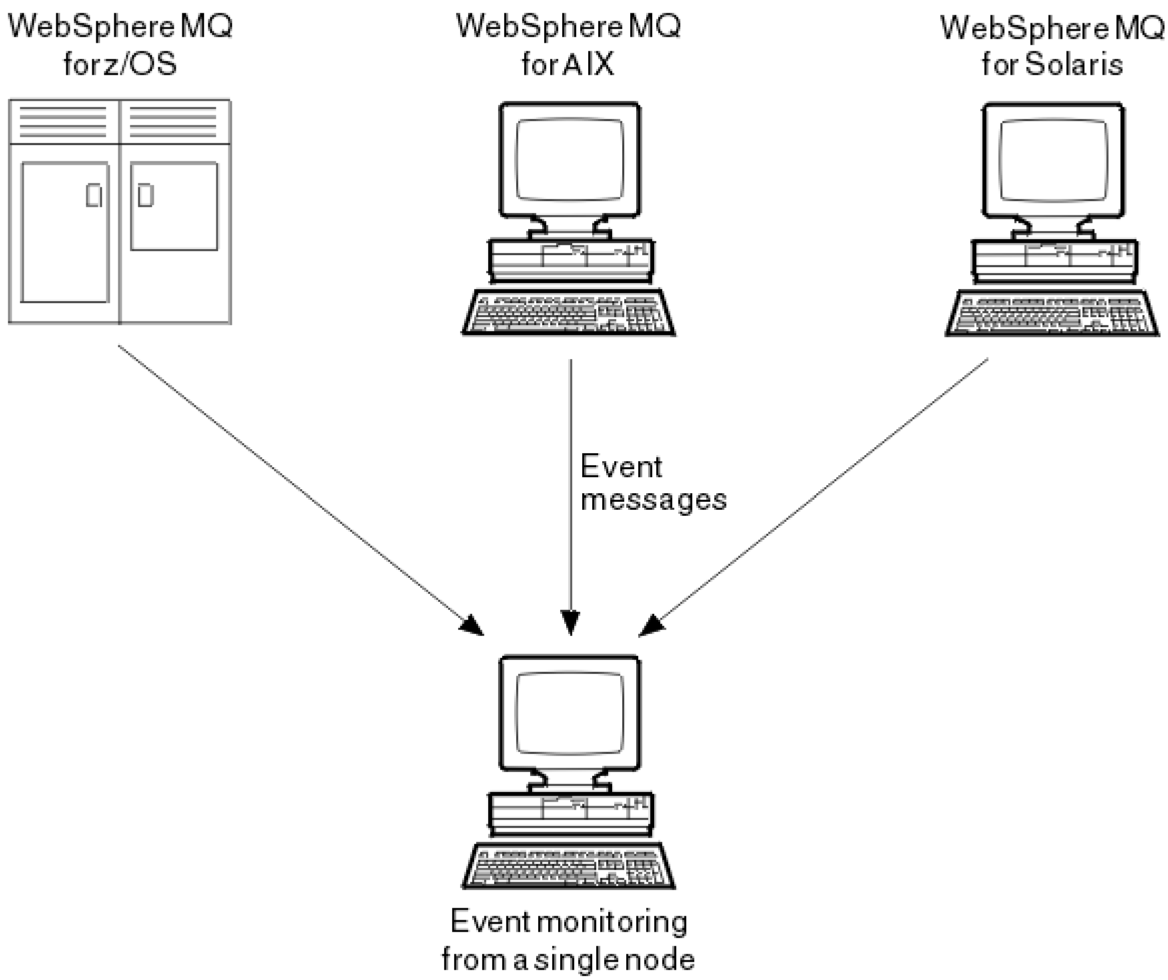


Figure 8. Monitoring queue managers across different platforms, on a single node

Instrumentation events also enable applications acting as agents for other administration networks, for example Tivoli NetView for z/OS, to monitor reports and create the appropriate alerts.

Queue manager events
Queue manager events are related to the use of resources within queue managers. For example, a queue manager event is generated if an application tries to put a message on a queue that does not exist.

The following examples are conditions that can cause a queue manager event:

- An application issues an MQI call that fails. The reason code from the call is the same as the reason code in the event message.

  A similar condition can occur during the internal operation of a queue manager; for example, when generating a report message. The reason code in an event message might match an MQI reason code, even though it is not associated with any application. Do not assume that, because an event message reason code looks like an MQI reason code, the event was necessarily caused by an unsuccessful MQI call from an application.
- A command is issued to a queue manager and processing this command causes an event. For example:
  - A queue manager is stopped or started.
  - A command is issued where the associated user ID is not authorized for that command.

IBM MQ puts messages for queue manager events on the SYSTEM.ADMIN.QMGR.EVENT queue, and supports the following queue manager event types:

Authority (on Windows, and UNIX systems only)
Authority events report an authorization, such as an application trying to open a queue for which it does not have the required authority, or a command being issued from a user ID that does not have the required authority. The authority event message can contain the following event data:

- Not Authorized (type 1)
- Not Authorized (type 2)
- Not Authorized (type 3)
- Not Authorized (type 4)
- Not Authorized (type 5)
- Not Authorized (type 6)

All authority events are valid on Windows, and UNIX systems only.

Inhibit
Inhibit events indicate that an MQPUT or MQGET operation has been attempted against a queue where the queue is inhibited for puts or gets, or against a topic where the topic is inhibited for publishes. The inhibit event message can contain the following event data:

- Get Inhibited
- Put Inhibited

Local
Local events indicate that an application (or the queue manager) has not been able to access a local queue or other local object. For example, an application might try to access an object that has not been defined. The local event message can contain the following event data:

- Alias Base Queue Type Error
- Unknown Alias Base Queue
- Unknown Object Name

Remote
Remote events indicate that an application or the queue manager cannot access a remote queue on another queue manager. For example, the transmission queue to be used might not be correctly defined. The remote event message can contain the following event data:

- Default Transmission Queue Type Error
- Default Transmission Queue Usage Error
- Queue Type Error

- Remote Queue Name Error
- Transmission Queue Type Error
- Transmission Queue Usage Error
- Unknown Default Transmission Queue
- Unknown Remote Queue Manager
- Unknown Transmission Queue

Start and stop

Start and stop events indicate that a queue manager has been started or has been requested to stop or quiesce.

z/OS supports only start events.

Stop events are not recorded unless the default message-persistence of the SYSTEM.ADMIN.QMGR.EVENT queue is defined as persistent. The start and stop event message can contain the following event data:

- Queue Manager Active
- Queue Manager Not Active

For each event type in this list, you can set a queue manager attribute to enable or disable the event type.

Channel and bridge events

Channels report these events as a result of conditions detected during their operation. For example, when a channel instance is stopped.

Channel events are generated in the following circumstances:

- When a command starts or stops a channel.
- When a channel instance starts or stops.
- When a channel receives a conversion error warning when getting a message.
- When an attempt is made to create a channel automatically; the event is generated whether the attempt succeeds or fails.

Note: Client connections do not cause Channel Started or Channel Stopped events.

When a command is used to start a channel, an event is generated. Another event is generated when the channel instance starts. However, starting a channel by a listener, the **runmqchl** command, or a queue manager trigger message does not generate an event. In these cases, an event is generated only when the channel instance starts.

A successful start or stop channel command generates at least two events. These events are generated for both queue managers connected by the channel (providing they support events).

If a channel event is put on an event queue, an error condition causes the queue manager to create an event.

The event messages for channel and bridge events are put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The channel event messages can contain the following event data:

- Channel Activated
- Channel Auto-definition Error
- Channel Auto-definition OK
- Channel Conversion Error
- Channel Not Activated
- Channel Started
- Channel Stopped

- Channel Stopped By User
- Channel Blocked

IMS bridge events (z/OS only)

z/OS

These events are reported when an IMS bridge starts or stops.

The IMS bridge event messages can contain the following event data:

- Bridge Started
- Bridge Stopped

SSL events

The only Secure Sockets Layer (SSL or TLS) event is the Channel SSL Error event. This event is reported when a channel using SSL or TLS fails to establish an SSL connection.

The SSL event messages can contain the following event data:

- Channel SSL Error
- Channel SSL Warning

Performance events
Performance events are notifications that a resource has reached a threshold condition. For example, a queue depth limit has been reached.

Performance events relate to conditions that can affect the performance of applications that use a specified queue. They are not generated for the event queues themselves.

The event type is returned in the command identifier field in the message data.

If a queue manager tries to put a queue manager event or performance event message on an event queue and an error that would typically create an event is detected, another event is not created and no action is taken.

MQGET and MQPUT calls within a unit of work can generate performance events regardless of whether the unit of work is committed or backed out.

The event messages for performance events are put on the SYSTEM.ADMIN.PERFM.EVENT queue.

There are two types of performance event:

Queue depth events
Queue depth events relate to the number of messages on a queue; that is, how full or empty the queue is. These events are supported for shared queues. The queue depth event messages can contain the following event data:

- Queue Depth High
- Queue Depth Low
- Queue Full

Queue service interval events
Queue service interval events relate to whether messages are processed within a user-specified time interval. These events are not supported for shared queues.

z/OS IBM MQ for z/OS supports queue depth events for QSGDISP (SHARED) queues, but not service interval events. Queue manager and channel events remain unaffected by shared queues. The queue service event messages can contain the following event data:

- Queue Service Interval High
- Queue Service Interval OK

Configuration events
Configuration events are generated when a configuration event is requested explicitly, or automatically when an object is created, modified, or deleted.

A configuration event message contains information about the attributes of an object. For example, a configuration event message is generated if a namelist object is created, and contains information about the attributes of the namelist object.

The event messages for configuration events are put on the SYSTEM.ADMIN.CONFIG.EVENT queue.

There are four types of configuration event:

Create object events
Create object events are generated when an object is created. The event message contains the following event data: Create object.

Change object events
Change object events are generated when an object is changed. The event message contains the following event data: Change object.

Delete object events
Delete object events are generated when an object is deleted. The event message contains the following event data: Delete object.

Refresh object events
Refresh object events are generated by an explicit request to refresh. The event message contains the following event data: Refresh object.

Command events
Command events are reported when an MQSC or PCF command runs successfully.

A command event message contains information about the origin, context, and content of a command. For example, a command event message is generated with such information if the MQSC command, ALTER QLOCAL, runs successfully.

The event messages for command events are put on the SYSTEM.ADMIN.COMMAND.EVENT queue.

Command events contain the following event data: Command.

Logger events
Logger events are reported when a queue manager that uses linear logging starts writing log records to a new log extent IBM i or, on IBM i, to a new journal receiver. z/OS Logger events are not available with IBM MQ for z/OS.

A logger event message contains information specifying the log extents required by the queue manager to restart the queue manager, or for media recovery.

The event messages for logger events are put on the SYSTEM.ADMIN.LOGGER.EVENT queue.

The logger event message contains the following event data: Logger.

Event message data summary
Use this summary to obtain information about the event data that each type of event message can contain.

Event type	See these topics
Authority events	Not Authorized (type 1)
	Not Authorized (type 2)
	Not Authorized (type 3)
	Not Authorized (type 4)
	Not Authorized (type 5)
	Not Authorized (type 6)
Channel events	Channel Activated
	Channel Auto-definition Error
	Channel Auto-definition OK
	Channel Blocked
	Channel Conversion Error
	Channel Not Activated
	Channel Started
	Channel Stopped
	Channel Stopped By User
Command events	Command
Configuration events	Create object
	Change object
	Delete object
	Refresh object
IMS bridge events	Bridge Started
	Bridge Stopped
Inhibit events	Get Inhibited
	Put Inhibited
Local events	Alias Base Queue Type Error
	Unknown Alias Base Queue
	Unknown Object Name
Logger events	Logger
Performance events	Queue Depth High
	Queue Depth Low
	Queue Full
	Queue Service Interval High
	Queue Service Interval OK

Event type	See these topics
Remote events	Default Transmission Queue Type Error
	Default Transmission Queue Usage Error
	Queue Type Error
	Remote Queue Name Error
	Transmission Queue Type Error
	Transmission Queue Usage Error
	Unknown Default Transmission Queue
	Unknown Remote Queue Manager
	Unknown Transmission Queue
SSL events	Channel SSL Error
Start and stop events	Queue Manager Active
	Queue Manager Not Active

Controlling events

You enable and disable events by specifying the appropriate values for queue manager, queue attributes, or both, depending on the type of event.

You must enable each instrumentation event that you want to be generated. For example, the conditions causing a Queue Full event are:

- Queue Full events are enabled for a specified queue, and
- An application issues an MQPUT request to put a message on that queue, but the request fails because the queue is full.

Enable and disable events by using any of the following techniques:

- IBM MQ script commands (MQSC).
- The corresponding IBM MQ PCF commands.
- z/OS The operations and control panels for queue managers on z/OS.
- The IBM MQ Explorer.

Note: You can set attributes related to events for both queues and queue managers only by command. The MQI call MQSET does not support attributes related to events.

Related concepts

"Instrumentation events" on page 19
An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

Related reference

"Event types" on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information

The MQSC commands
Automating administration tasks
Using Programmable Command Formats
Introducing the operations and control panels

Controlling queue manager events

You control queue manager events by using queue manager attributes. To enable queue manager events, set the appropriate queue manager attribute to ENABLED. To disable queue manager events, set the appropriate queue manager attribute to DISABLED.

To enable or disable queue manager events, use the MQSC command ALTER QMGR, specifying the appropriate queue manager attribute. Table 1 on page 29 summarizes how to enable queue manager events. To disable a queue manager event, set the appropriate parameter to DISABLED.

Table 1. Enabling queue manager events using MQSC commands

Event	ALTER QMGR parameter
Authority Inhibit Local Remote Start and Stop	AUTHOREV (ENABLED) INHIBTEV (ENABLED) LOCALEV (ENABLED) REMOTEEV (ENABLED) STRSTPEV (ENABLED)

Controlling channel and bridge events

You control channel events by using queue manager attributes. To enable channel events, set the appropriate queue manager attribute to ENABLED. To disable channel events, set the appropriate queue manager attribute to DISABLED.

To enable or disable channels events use the MQSC command ALTER QMGR, specifying the appropriate queue manager attribute. Table 2 on page 29 summarizes how you enable channel and bridge events. To disable a queue manager event, set the appropriate parameter to DISABLED.

Restriction: z/OS Channel auto-definition events are not available on IBM MQ for z/OS.

Table 2. Enabling channel and bridge events using MQSC commands

Event	ALTER QMGR parameter
Channel Related to channel errors only IMS bridge SSL Channel auto-definition	CHLEV (ENABLED) CHLEV (EXCEPTION) BRIDGEEV (ENABLED) SSLEV (ENABLED) CHADEV(ENABLED)

With CHLEV set to exception, the following return codes, and corresponding reason qualifiers are generated:

- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED
  - with the following ReasonQualifiers:
    - MQRQ_CHANNEL_STOPPED_ERROR
    - MQRQ_CHANNEL_STOPPED_RETRY
    - MQRQ_CHANNEL_STOPPED_DISABLED
- MQRC_CHANNEL_STOPPED_BY_USER
- MQRC_CHANNEL_BLOCKED
  - with the following ReasonQualifiers:
    - MQRQ_CHANNEL_BLOCKED_NOACCESS

- MQRQ_CHANNEL_BLOCKED_USERID
- MQRQ_CHANNEL_BLOCKED_ADDRESS

Controlling performance events
You control performance events using the PERFMEV queue manager attribute. To enable performance events, set PERFMEV to ENABLED. To disable performance events, set the PERFMEV queue manager attribute to DISABLED.

To set the PERFMEV queue manager attribute to ENABLED, use the following MQSC command:

```
ALTER QMGR PERFMEV (ENABLED)
```

To enable specific performance events, set the appropriate queue attribute. Also, specify the conditions that cause the event.

Queue depth events
By default, all queue depth events are disabled. To configure a queue for any of the queue depth events:

1. Enable performance events on the queue manager.
2. Enable the event on the required queue.
3. Set the limits, if required, to the appropriate levels, expressed as a percentage of the maximum queue depth.

Queue service interval events
To configure a queue for queue service interval events you must:

1. Enable performance events on the queue manager.
2. Set the control attribute for a Queue Service Interval High or OK event on the queue as required.
3. Specify the service interval time by setting the QSVCINT attribute for the queue to the appropriate length of time.

Note: When enabled, a queue service interval event can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if an MQI call is used on a queue to put or remove a message, any applicable performance event is generated at that time. The event is *not* generated when the elapsed time becomes equal to the service interval time.

Controlling configuration, command, and logger events
You control configuration, command, and logger events by using the queue manager attributes CONFIGEV, CMDEV, and LOGGEREV. To enable these events, set the appropriate queue manager attribute to ENABLED. To disable these events, set the appropriate queue manager attribute to DISABLED.

Configuration events
To enable configuration events, set CONFIGEV to ENABLED. To disable configuration events, set CONFIGEV to DISABLED. For example, you can enable configuration events by using the following MQSC command:

```
ALTER QMGR CONFIGEV (ENABLED)
```

Command events
To enable command events, set CMDEV to ENABLED. To enable command events for commands except DISPLAY MQSC commands and Inquire PCF commands, set the CMDEV to NODISPLAY. To disable command events, set CMDEV to DISABLED. For example, you can enable command events by using the following MQSC command:

```
ALTER QMGR CMDEV (ENABLED)
```

Logger events

To enable logger events, set LOGGEREV to ENABLED. To disable logger events, set LOGGEREV to DISABLED. For example, you can enable logger events by using the following MQSC command:

```
ALTER QMGR LOGGEREV(ENABLED)
```

Event queues

When an event occurs, the queue manager puts an event message on the defined event queue. The event message contains information about the event.

You can define event queues, either as:

- Local queues
- Alias queues
- Local definitions of remote queues, or as
- Remote cluster queues

If you define all your event queues as local definitions of the same remote queue on one queue manager, you can centralize your monitoring activities.

You must not define event queues as transmission queues, because event messages have formats that are incompatible with the message format that is required for transmission queues.

Shared event queues are local queues defined with the QSGDISP(SHARED) value.

For more information about defining shared queues on z/OS, see Application programming with shared queues.

When an event queue is unavailable

If an event occurs when the event queue is not available, the event message is lost. For example, if you do not define an event queue for a category of event, all event messages for that category are lost. The event messages are not, for example, saved on the dead-letter (undelivered-message) queue.

However, you can define the event queue as a remote queue. Then, if there is a problem on the remote system putting messages to the resolved queue, the event message arrives on the dead-letter queue of the remote system.

An event queue might be unavailable for many different reasons including:

- The queue has not been defined.
- The queue has been deleted.
- The queue is full.
- The queue has been put-inhibited.

The absence of an event queue does not prevent the event from occurring. For example, after a performance event, the queue manager changes the queue attributes and resets the queue statistics. This change happens whether the event message is put on the performance event queue or not. The same is true in the case of configuration and command events.

Using triggered event queues

You can set up the event queues with triggers so that when an event is generated, the event message being put onto the event queue starts a user-written monitoring application. This application can process the event messages and take appropriate action. For example, certain events might require an operator to be informed, other events might start an application that performs some administration tasks automatically.

Event queues can have trigger actions associated with them and can create trigger messages. However, if these trigger messages in turn cause conditions that would normally generate an event, no event is generated. not generating an event in this instance ensures that looping does not occur.

Related concepts

“Controlling events” on page 28
You enable and disable events by specifying the appropriate values for queue manager, queue attributes, or both, depending on the type of event.

“Format of event messages” on page 32
Event messages contain information about an event and its cause. Like other IBM MQ messages, an event message has two parts: a message descriptor and the message data.

Related information

Application programming with shared queues
QSGDisp (MQLONG)
Conditions for a trigger event

Format of event messages

Event messages contain information about an event and its cause. Like other IBM MQ messages, an event message has two parts: a message descriptor and the message data.

- The message descriptor is based on the MQMD structure.
- The message data consists of an *event header* and the *event data*. The event header contains the reason code that identifies the event type. Putting the event message, and any subsequent action, does not affect the reason code returned by the MQI call that caused the event. The event data provides further information about the event.

Typically, you process event messages with a system management application tailored to meet the requirements of the enterprise at which it runs.

When the queue managers in a queue sharing group detect the conditions for generating an event message, several queue managers can generate an event message for the shared queue, resulting in several event messages. To ensure that a system can correlate multiple event messages from different queue managers, these event messages have a unique correlation identifier *(CorrelId)* set in the message descriptor (MQMD).

Related reference

“Activity report MQMD (message descriptor)” on page 108
Use this page to view the values contained by the MQMD structure for an activity report

“Activity report MQEPH (Embedded PCF header)” on page 112
Use this page to view the values contained by the MQEPH structure for an activity report

“Activity report MQCFH (PCF header)” on page 113
Use this page to view the PCF values contained by the MQCFH structure for an activity report

Related information

Event message reference
Event message format
Event message MQMD (message descriptor)
Event message MQCFH (PCF header)
Event message descriptions

Performance events

Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

Performance event messages can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if you use an MQI call on a queue to put or remove a message, any appropriate performance events are generated at that time.

Every performance event message that is generated is placed on the queue, SYSTEM.ADMIN.PERFM.EVENT.

The event data contains a reason code that identifies the cause of the event, a set of performance event statistics, and other data. The types of event data that can be returned in performance event messages are described in the following list:

- Queue Depth High
- Queue Depth Low
- Queue Full
- Queue Service Interval High
- Queue Service Interval OK

Examples that illustrate the use of performance events assume that you set queue attributes by using the appropriate IBM MQ commands (MQSC). On z/OS, you can also set queue attributes using the operations and controls panels for queue managers.

Related reference

"Event types" on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Performance event statistics

The performance event data in the event message contains statistics about the event. Use the statistics to analyze the behavior of a specified queue.

The event data in the event message contains information about the event for system management programs. For all performance events, the event data contains the names of the queue manager and the queue associated with the event. The event data also contains statistics related to the event. Table 3 on page 33 summarizes the event statistics that you can use to analyze the behavior of a queue. All the statistics refer to what has happened since the last time the statistics were reset.

Table 3. Performance event statistics

Parameter	Description
TimeSinceReset	The elapsed time since the statistics were last reset.
HighQDepth	The maximum number of messages on the queue since the statistics were last reset.
MsgEnqCount	The number of messages enqueued (the number of MQPUT calls to the queue), since the statistics were last reset.
MsgDeqCount	The number of messages dequeued (the number of MQGET calls to the queue), since the statistics were last reset.

Performance event statistics are reset when any of the following changes occur:

- A performance event occurs (statistics are reset on all active queue managers).
- A queue manager stops and restarts.

- The PCF command, Reset Queue Statistics, is issued from an application program.
- z/OS On z/OS only, the RESET QSTATS command is issued at the console.

Related concepts

"Performance events" on page 33
Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

"The service timer" on page 35
Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

"Rules for queue service interval events" on page 36
Formal rules control when the service timer is set and queue service interval events are generated.

Related tasks

"Enabling queue service interval events" on page 37
To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

Related information

Queue Depth High

Reset Queue Statistics

RESET QSTATS

Queue service interval events

Queue service interval events indicate whether an operation was performed on a queue within a user-defined time interval called the *service interval*. Depending on your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

Queue service interval events are *not* supported on shared queues.

The following types of queue service interval events can occur, where the term *get operation* refers to an **MQGET** call or an activity that removes a messages from a queue, such as using the **CLEAR QLOCAL** command:

Queue Service Interval OK

Indicates that after one of the following operations:

- An MQPUT call
- A get operation that leaves a non-empty queue

a get operation was performed within a user-defined time period, known as the *service interval*.

Only a get operation can cause the Queue Service Interval OK event message. Queue Service Interval OK events are sometimes described as OK events.

Queue Service Interval High

Indicates that after one of the following operations:

- An MQPUT call
- A get operation that leaves a non-empty queue

a get operation was **not** performed within a user-defined service interval.

Either a get operation or an MQPUT call can cause the Queue Service Interval High event message. Queue Service Interval High events are sometimes described as High events.

To enable both Queue Service Interval OK and Queue Service Interval High events, set the `QServiceIntervalEvent` control attribute to High. Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated. You do not need to enable Queue Service Interval OK events independently.

OK and High events are mutually exclusive, so if one is enabled the other is disabled. However, both events can be simultaneously disabled.

Figure 9 on page 35 shows a graph of queue depth against time. At time P1, an application issues an MQPUT, to put a message on the queue. At time G1, another application issues an MQGET to remove the message from the queue.

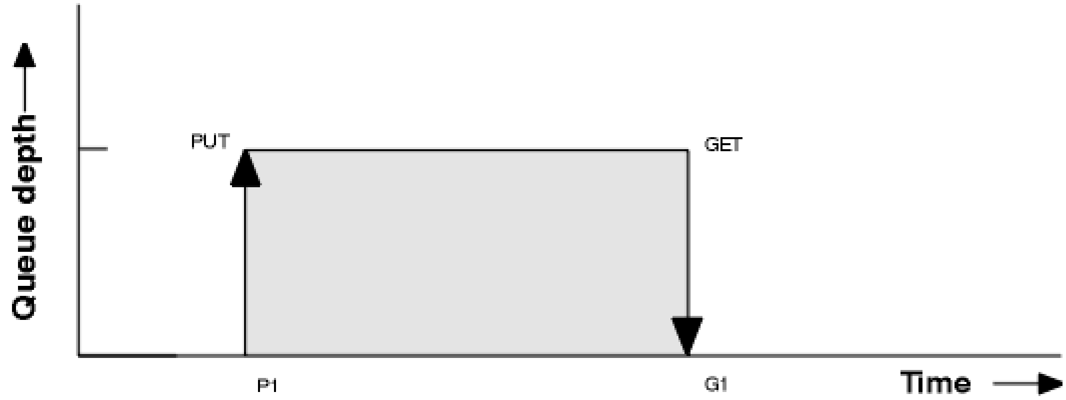


Figure 9. Understanding queue service interval events

The possible outcomes of queue service interval events are as follows:

- If the elapsed time between the put and the get is less than or equal to the service interval:
  - A *Queue Service Interval OK* event is generated at time G1, if queue service interval events are enabled
- If the elapsed time between the put and get is greater than the service interval:
  - A *Queue Service Interval High* event is generated at time G1, if queue service interval events are enabled.

The algorithm for starting the service timer and generating events is described in "Rules for queue service interval events" on page 36.

Related information

Queue Service Interval OK

Queue Service Interval High

QServiceIntervalEvent (MQLONG)

QServiceIntervalEvent (10-digit signed integer)

ServiceIntervalEvent property

The service timer

Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

What precisely does the service timer measure?

The service timer measures the elapsed time between an MQPUT call to an empty queue or a get operation, and the next put or get, provided the queue depth is nonzero between these two operations.

When is the service timer active?

The service timer is always active (running), if the queue has messages on it (depth is nonzero) and a queue service interval event is enabled. If the queue becomes empty (queue depth zero), the timer is put into an OFF state, to be restarted on the next put.

When is the service timer reset?

The service timer is always reset after a get operation . It is also reset by an MQPUT call to an empty queue. However, it is not necessarily reset on a queue service interval event.

How is the service timer used?

Following a get operation or an MQPUT call, the queue manager compares the elapsed time as measured by the service timer, with the user-defined service interval. The result of this comparison is that:

- An OK event is generated if there is a get operation and the elapsed time is less than or equal to the service interval, AND this event is enabled.
- A high event is generated if the elapsed time is greater than the service interval, AND this event is enabled.

Can applications read the service timer?

No, the service timer is an internal timer that is not available to applications.

What about the *TimeSinceReset* parameter?

The *TimeSinceReset* parameter is returned as part of the event statistics in the event data. It specifies the time between successive queue service interval events, unless the event statistics are reset.

Rules for queue service interval events

Formal rules control when the service timer is set and queue service interval events are generated.

Rules for the service timer

The service timer is reset to zero and restarted as follows:

- After an MQPUT call to an empty queue.
- After an MQGET call, if the queue is not empty after the MQGET call.

The resetting of the timer does not depend on whether an event has been generated.

At queue manager startup the service timer is set to startup time if the queue depth is greater than zero.

If the queue is empty following a get operation, the timer is put into an OFF state.

Queue Service Interval High events

The Queue Service Interval event must be enabled (set to HIGH).

Queue Service Interval High events are automatically enabled when a Queue Service Interval OK event is generated.

If the service time is greater than the service interval, an event is generated on, or before, the next MQPUT or get operation.

Queue Service Interval OK events

Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated.

If the service time (elapsed time) is less than or equal to the service interval, an event is generated on, or before, the next get operation.

Related tasks

"Enabling queue service interval events" on page 37

To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

Enabling queue service interval events
To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

About this task

The high and OK events are mutually exclusive; that is, when one is enabled, the other is automatically disabled:

- When a high event is generated on a queue, the queue manager automatically disables high events and enables OK events for that queue.
- When an OK event is generated on a queue, the queue manager automatically disables OK events and enables high events for that queue.

Table 4. Enabling queue service interval events using MQSC

Queue service interval event	**Queue attributes**
Queue Service Interval High Queue Service Interval OK No queue service interval events	QSVCIEV (HIGH) QSVCIEV (OK) QSVCIEV (NONE)
Service interval	QSVCINT (*tt*) where *tt* is the service interval time in milliseconds.

Perform the following steps to enable queue service interval events:

Procedure

1. Set the queue manager attribute PERFMEV to ENABLED.
   Performance events are enabled on the queue manager.
2. Set the control attribute, QSVCIEV, for a Queue Service Interval High or OK event on the queue, as required.
3. Set the QSVCINT attribute for the queue to specify the appropriate service interval time.

Example

To enable Queue Service Interval High events with a service interval time of 10 seconds (10 000 milliseconds) use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QSVCINT(10000) QSVCIEV(HIGH)
```

.

Queue service interval events examples

Use the examples in this section to understand the information that you can obtain from queue service interval events.

The three subtopic examples provide progressively more complex illustrations of the use of queue service interval events.

The figures accompanying the examples in each subtopic have the same structure:

- Figure 1 is a graph of queue depth against time, showing individual MQGET calls and MQPUT calls.
- The Commentary section shows a comparison of the time constraints. There are three time periods that you must consider:
  - The user-defined service interval.
  - The time measured by the service timer.
  - The time since event statistics were last reset (TimeSinceReset in the event data).
- The Event statistics summary section shows which events are enabled at any instant and what events are generated.

The examples illustrate the following aspects of queue service interval events:

- How the queue depth varies over time.
- How the elapsed time as measured by the service timer compares with the service interval.
- Which event is enabled.
- Which events are generated.

Remember: Example 1 shows a simple case where the messages are intermittent and each message is removed from the queue before the next one arrives. From the event data, you know that the maximum number of messages on the queue was one. You can, therefore, work out how long each message was on the queue.

However, in the general case, where there is more than one message on the queue and the sequence of MQGET calls and MQPUT calls is not predictable, you cannot use queue service interval events to calculate how long an individual message remains on a queue. The TimeSinceReset parameter, which is returned in the event data, can include a proportion of time when there are no messages on the queue. Therefore any results you derive from these statistics are implicitly averaged to include these times.

Related concepts

“Queue service interval events” on page 34
Queue service interval events indicate whether an operation was performed on a queue within a user-defined time interval called the *service interval*. Depending on your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

“The service timer” on page 35

Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

Queue service interval events: example 1
A basic sequence of MQGET calls and MQPUT calls, where the queue depth is always one or zero.

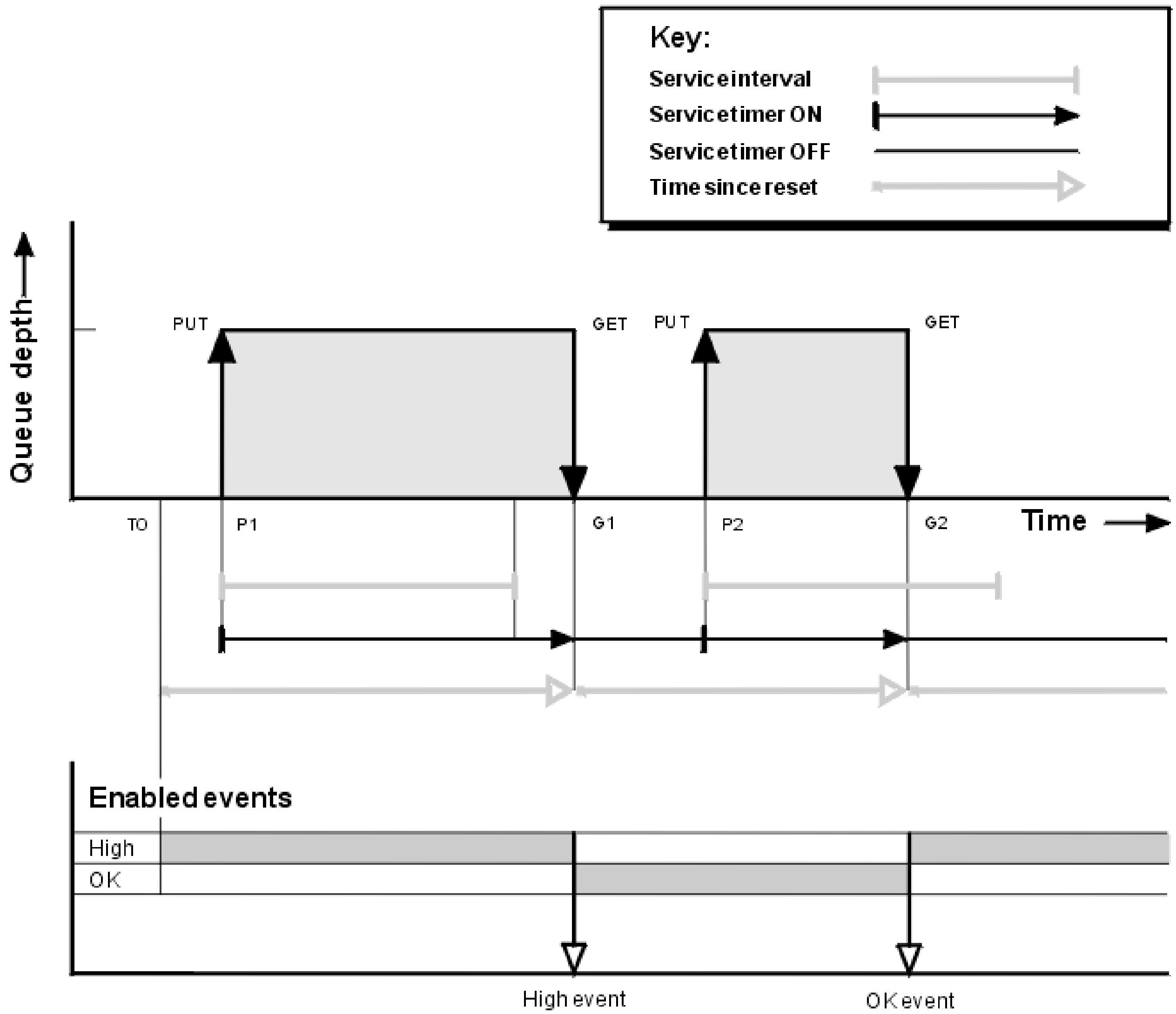


Figure 10. Queue service interval events - example 1

Commentary

1. At P1, an application puts a message onto an empty queue. This starts the service timer.

   Note that T0 might be queue manager startup time.
2. At G1, another application gets the message from the queue. Because the elapsed time between P1 and G1 is greater than the service interval, a Queue Service Interval High event is generated on the MQGET call at G1. When the high event is generated, the queue manager resets the event control attribute so that:
   a. The OK event is automatically enabled.
   b. The high event is disabled.

   Because the queue is now empty, the service timer is switched to an OFF state.
3. At P2, a second message is put onto the queue. This restarts the service timer.

4. At G2, the message is removed from the queue. However, because the elapsed time between P2 and G2 is less than the service interval, a Queue Service Interval OK event is generated on the MQGET call at G2. When the OK event is generated, the queue manager resets the control attribute so that:
   a. The high event is automatically enabled.
   b. The OK event is disabled.

   Because the queue is empty, the service timer is again switched to an OFF state.

Event statistics summary

Table 5 on page 40 summarizes the event statistics for this example.

Table 5. Event statistics summary for example 1

	Event 1	**Event 2**
Time of event	T(G1)	T(G2)
Type of event	High	OK
TimeSinceReset	T(G1) - T(0)	T(G2) - T(G1)
HighQDepth	1	1
MsgEnqCount	1	1
MsgDeqCount	1	1

The middle part of Figure 10 on page 39 shows the elapsed time as measured by the service timer compared to the service interval for that queue. To see whether a queue service interval event might occur, compare the length of the horizontal line representing the service timer (with arrow) to that of the line representing the service interval. If the service timer line is longer, and the Queue Service Interval High event is enabled, a Queue Service Interval High event occurs on the next get. If the timer line is shorter, and the Queue Service Interval OK event is enabled, a Queue Service Interval OK event occurs on the next get.

Queue service interval events: example 2
A sequence of MQPUT calls and MQGET calls, where the queue depth is not always one or zero.

This example also shows instances of the timer being reset without events being generated, for example, at time P2.

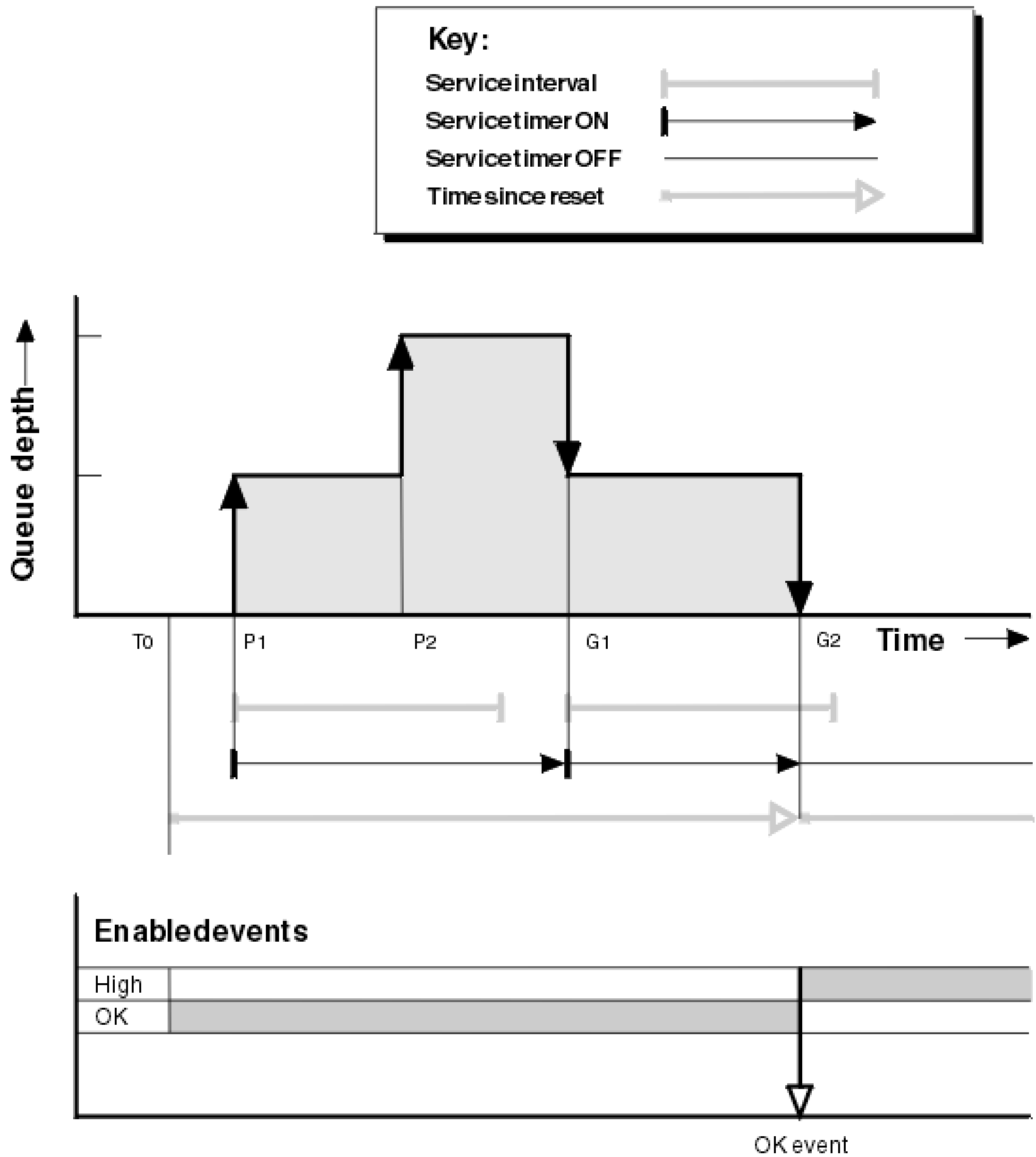


Figure 11. Queue service interval events - example 2

Commentary

In this example, OK events are enabled initially and queue statistics were reset at time T0.

1. At P1, the first put starts the service timer.
2. At P2, the second put does not generate an event because a put cannot cause an OK event.
3. At G1, the service interval has now been exceeded and therefore an OK event is not generated. However, the MQGET call causes the service timer to be reset.
4. At G2, the second get occurs within the service interval and this time an OK event is generated. The queue manager resets the event control attribute so that:

    a. The high event is automatically enabled.

    b. The OK event is disabled.

    Because the queue is now empty, the service timer is switched to an OFF state.

Event statistics summary

Table 6 on page 42 summarizes the event statistics for this example.

Table 6. Event statistics summary for example 2

	Event 2
Time of event	T(G2)
Type of event	OK
TimeSinceReset	T(G2) - T(0)
HighQDepth	2
MsgEnqCount	2
MsgDeqCount	2

Queue service interval events: example 3
A sequence of MQGET calls and MQPUT calls that is more sporadic than the previous examples.

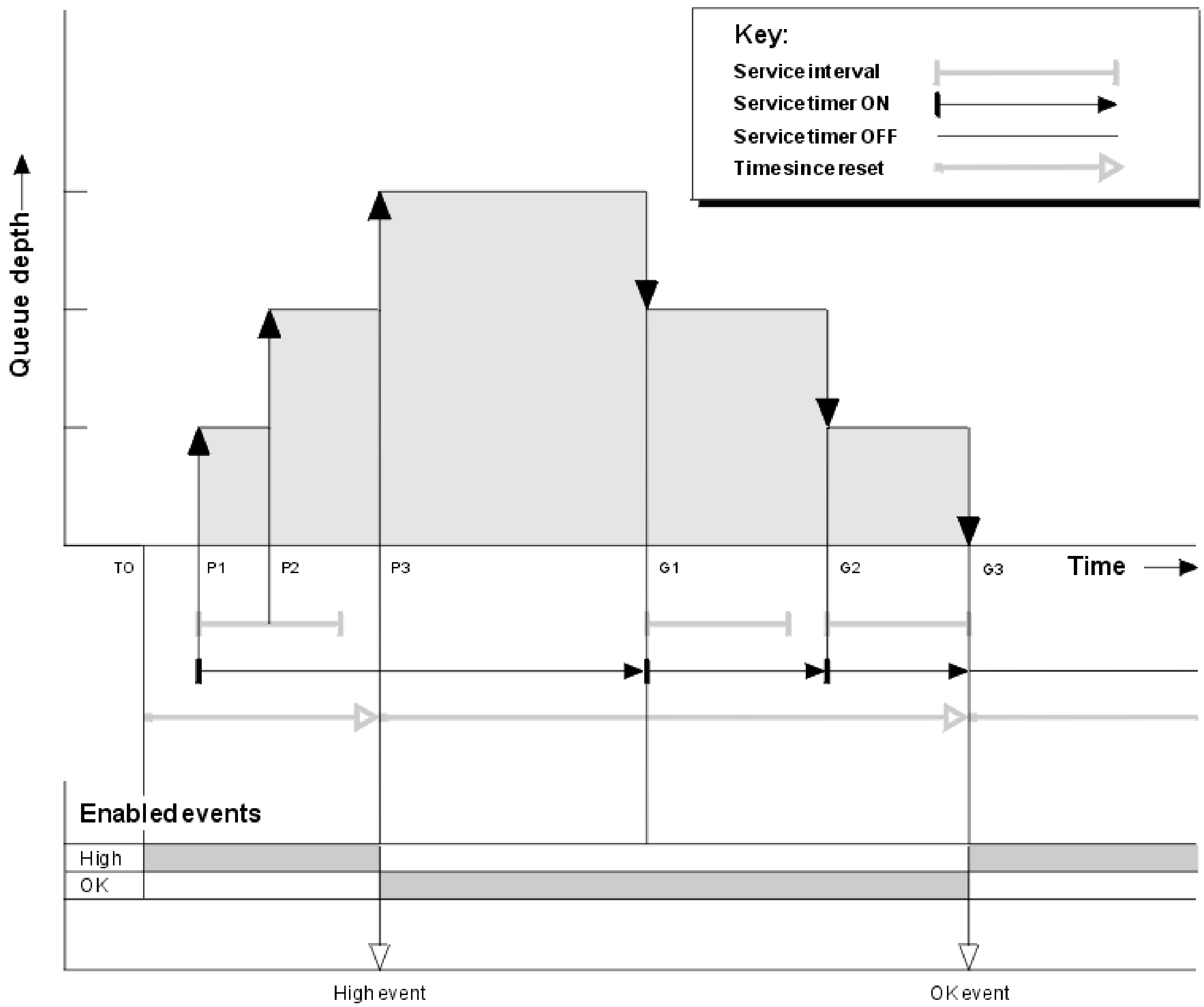


Figure 12. Queue service interval events - example 3

Commentary

1. At time T(0), the queue statistics are reset and Queue Service Interval High events are enabled.
2. At P1, the first put starts the service timer.
3. At P2, the second put increases the queue depth to two. A high event is not generated here because the service interval time has not been exceeded.
4. At P3, the third put causes a high event to be generated. (The timer has exceeded the service interval.) The timer is not reset because the queue depth was not zero before the put. However, OK events are enabled.
5. At G1, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. The MQGET call does, however, reset the service timer.
6. At G2, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. Again, the MQGET call resets the service timer.
7. At G3, the third get empties the queue and the service timer is *equal* to the service interval. Therefore an OK event is generated. The service timer is reset and high events are enabled. The MQGET call empties the queue, and this puts the timer in the OFF state.

Event statistics summary

Table 7 on page 43 summarizes the event statistics for this example.

Table 7. Event statistics summary for example 3

	Event 1	**Event 2**
Time of event	T(P3)	T(G3)
Type of event	High	OK
TimeSinceReset	T(P3) - T(0)	T(G3) - T(P3)
HighQDepth	3	3
MsgEnqCount	3	0
MsgDeqCount	0	3

Queue depth events

Queue depth events are related to the queue depth, that is, the number of messages on the queue.

In IBM MQ applications, queues must not become full. If they do, applications can no longer put messages on the queue that they specify. Although the message is not lost if this occurs, a full queue can cause considerable inconvenience. The number of messages can build up on a queue if the messages are being put onto the queue faster than the applications that process them can take them off.

The solution to this problem depends on the particular circumstances, but might involve:

- Diverting some messages to another queue.
- Starting new applications to take more messages off the queue.
- Stopping nonessential message traffic.
- Increasing the queue depth to overcome a transient maximum.

Advance warning that problems might be on their way makes it easier to take preventive action. For this purpose, IBM MQ provides the following queue depth events:

Queue Depth High events
: Indicate that the queue depth has increased to a predefined threshold called the Queue Depth High limit.

Queue Depth Low events
: Indicate that the queue depth has decreased to a predefined threshold called the Queue Depth Low limit.

Queue Full events
: Indicate that the queue has reached its maximum depth, that is, the queue is full.

A Queue Full Event is generated when an application attempts to put a message on a queue that has reached its maximum depth. Queue Depth High events give advance warning that a queue is filling up. This means that having received this event, the system administrator needs to take some preventive action. You can configure the queue manager such that, if the preventive action is successful and the queue depth drops to a safer level, the queue manager generates a Queue Depth Low event.

The first queue depth event example illustrates the effect of presumed action preventing the queue becoming full.

Related concepts

"Queue depth events examples" on page 47
Use these examples to understand the information that you can obtain from queue depth events

Related information

Queue Full
Queue Depth High
Queue Depth Low

Enabling queue depth events
To configure a queue for any of the queue depth events you set the appropriate queue manager and queue attributes.

About this task

By default, all queue depth events are disabled. When enabled, queue depth events are generated as follows:

- A Queue Depth High event is generated when a message is put on the queue, causing the queue depth to be greater than or equal to the value set for **QDepthHighLimit**.
  - A Queue Depth High event is automatically enabled by a Queue Depth Low event on the same queue.
  - A Queue Depth High event automatically enables both a Queue Depth Low and a Queue Full event on the same queue.
- distributed A Queue Depth Low event is generated when a message is removed from a queue by a GET operation, causing the queue depth to be less than or equal to the value set for **QDepthLowLimit**.

  z/OS A Queue Depth Low event is generated when a message is removed from a queue by a GET operation, or would have been removed but has since expired, causing the queue depth to be less than or equal to the value set for **QDepthLowLimit**.
  - A Queue Depth Low event is automatically enabled by a Queue Depth High event or a Queue Full event on the same queue.
  - A Queue Depth Low event automatically enables both a Queue Depth High and a Queue Full event on the same queue.
- A Queue Full event is generated when an application is unable to put a message onto a queue because the queue is full.
  - A Queue Full event is automatically enabled by a Queue Depth High or a Queue Depth Low event on the same queue.
  - A Queue Full event automatically enables a Queue Depth Low event on the same queue.

Perform the following steps to configure a queue for any of the queue depth events:

Procedure

1. Enable performance events on the queue manager, using the queue manager attribute **PERFMEV**.

   The events go to the SYSTEM.ADMIN.PERFM.EVENT queue.
2. Set one of the following attributes to enable the event on the required queue:
   - *QDepthHighEvent* (**QDPHIEV** in MQSC)
   - *QDepthLowEvent* (**QDPLOEV** in MQSC)
   - *QDepthMaxEvent* (**QDPMAXEV** in MQSC)
3. Optional: To set the limits, assign the following attributes, as a percentage of the maximum queue depth:
   - *QDepthHighLimit* (**QDEPTHHI** in MQSC)
   - *QDepthLowLimit* (**QDEPTHLO** in MQSC)

   Restriction: QDEPTHHI must not be less than **QDEPTHLO**.

   If **QDEPTHHI** equals **QDEPTHLO** an event message is generated every time the queue depth passes the value in either direction, because the high threshold is enabled when the queue depth is below the value and the low threshold is enabled when the depth is above the value.

Results

Note:

distributed A Queue Depth Low event is not generated when expired messages are removed from a queue by a GET operation causing the queue depth to be less than, or equal to, the value set for **QDepthLowLimit**. IBM MQ generates the queue depth low event message only during a successful GET operation. Therefore, when the expired messages are removed from the queue, no queue depth low event message is generated. Additionally, after the removal of these expired messages from the queue, *QDepthHighEvent* and *QDepthLowEvent* are not reset.

z/OS IBM MQ generates the queue depth low event message during either a successful destructive GET operation, or a destructive GET operation that would have succeeded had a matching message not expired. Otherwise, when expired messages are removed from a queue during general background processing, no queue depth low event message is generated. Additionally, after the removal of expired messages from a queue during general background processing, *QDepthHighEvent* and *QDepthLowEvent* are not reset. For more information on expired message processing, see Tuning your queue manager on IBM MQ for z/OS.

Example

To enable Queue Depth High events on the queue MYQUEUE with a limit set at 80%, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHHI(80) QDPHIEV(ENABLED)
```

To enable Queue Depth Low events on the queue MYQUEUE with a limit set at 20%, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHLO(20) QDPLOEV(ENABLED)
```

To enable Queue Full events on the queue MYQUEUE, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDPMAXEV(ENABLED)
```

Shared queues and queue depth events (IBM MQ for z/OS)
Event monitoring is more straightforward for an application that uses shared queues if all the queue managers in the queue-sharing group have the same setting for the PERFMEV attribute.

When a queue depth event occurs on a shared queue, and the queue manager attribute PERFMEV is set to ENABLED, the queue managers in the queue-sharing group produce an event message. If PERFMEV is set to DISABLED on some of the queue managers, event messages are not produced by those queue managers, making event monitoring from an application more difficult. For more straightforward monitoring, give each queue manager the same setting for the PERFMEV attribute.

This event message that each queue manager generates represents its individual usage of the shared queue. If a queue manager performs no activity on the shared queue, various values in the event message are null or zero. You can use null event messages as follows:

- Ensure that each active queue manager in a queue-sharing group generates one event message
- Highlight cases of no activity on a shared queue for the queue manager that produced the event message

Coordinating queue manager

When a queue manager issues a queue depth event, it updates the shared queue object definition to toggle the active performance event attributes. For example, depending on the definition of the queue attributes, a Queue Depth High event enables a Queue Depth Low and a Queue Full event. After updating the shared queue object successfully, the queue manager that detected the performance event initially becomes the *coordinating queue manager*.

If enabled for performance events, the coordinating queue manager performs the following actions:

1. Issues an event message that captures all shared queue performance data it has gathered since the last time an event message was created, or since the queue statistics were last reset. The message descriptor (MQMD) of this message contains a unique correlation identifier (*CorrelId*) created by the coordinating queue manager.
2. Broadcasts to all other *active* queue managers in the same queue-sharing group to request the production of an event message for the shared queue. The broadcast contains the correlation identifier created by the coordinating queue manager for the set of event messages.

Having received a request from the coordinating queue manager, if there is an active queue manager in the queue-sharing group that is enabled for performance events , that active queue manager issues an event message for the shared queue. The event message that is issued contains information about all the operations performed by the receiving (active) queue manager since the last time an event message was created, or since the statistics were last reset. The message descriptor (MQMD) of this event message contains the unique correlation identifier (*CorrelId*) specified by the coordinating queue manager.

When performance events occur on a shared queue, *n* event messages are produced, where *n* is a number from 1 to the number of active queue managers in the queue-sharing group. Each event message contains data that relates to the shared queue activity for the queue manager that generated the event message.

Differences between shared and nonshared queues

Enabling queue depth events on shared queues differs from enabling them on nonshared queues. A key difference is that events are switched on for shared queues even if PERFMEV is DISABLED on the queue manager. This is not the case for nonshared queues.

Consider the following example, which illustrates this difference:

- QM1 is a queue manager with `PerformanceEvent` (PERFMEV in MQSC) set to DISABLED.
- SQ1 is a shared queue with QSGDISP set to (SHARED) QLOCAL in MQSC.
- LQ1 is a nonshared queue with QSGDISP set to (QMGR) QLOCAL in MQSC.

Both queues have the following attributes set on their definitions:

- QDPHIEV (ENABLED)
- QDPLOEV (DISABLED)
- QDPMAXEV (DISABLED)

If messages are placed on both queues so that the depth meets or exceeds the QDEPTHHI threshold, the QDPHIEV value on SQ1 switches to DISABLED. Also, QDPLOEV and QDPMAXEV are switched to ENABLED. SQ1's attributes are automatically switched for each performance event at the time the event criteria are met.

In contrast the attributes for LQ1 remain unchanged until PERFMEV on the queue manager is ENABLED. This means that if the queue manager's PERFMEV attribute is ENABLED, DISABLED and then re-ENABLED for example, the performance event settings on shared queues might not be consistent with those of nonshared queues, even though they might have initially been the same.

Queue depth events examples

Use these examples to understand the information that you can obtain from queue depth events

The first example provides a basic illustration of queue depth events. The second example is more extensive, but the principles are the same as for the first example. Both examples use the same queue definition, as follows:

The queue, MYQUEUE1, has a maximum depth of 1000 messages. The high queue depth limit is 80% and the low queue depth limit is 20%. Initially, Queue Depth High events are enabled, while the other queue depth events are disabled.

The IBM MQ commands (MQSC) to configure this queue are:

```
ALTER QMGR PERFMEV(ENABLED)

DEFINE QLOCAL('MYQUEUE1') MAXDEPTH(1000) QDPMAXEV(DISABLED) QDEPTHHI(80)
QDPHIEV(ENABLED) QDEPTHLO(20) QDPLOEV(DISABLED)
```

Related concepts

“Queue depth events” on page 43
Queue depth events are related to the queue depth, that is, the number of messages on the queue.

Related tasks

“Enabling queue depth events” on page 44
To configure a queue for any of the queue depth events you set the appropriate queue manager and queue attributes.

Related information

The MQSC commands

Queue depth events: example 1
A basic sequence of queue depth events.

Figure 13 on page 48 shows the variation of queue depth over time.

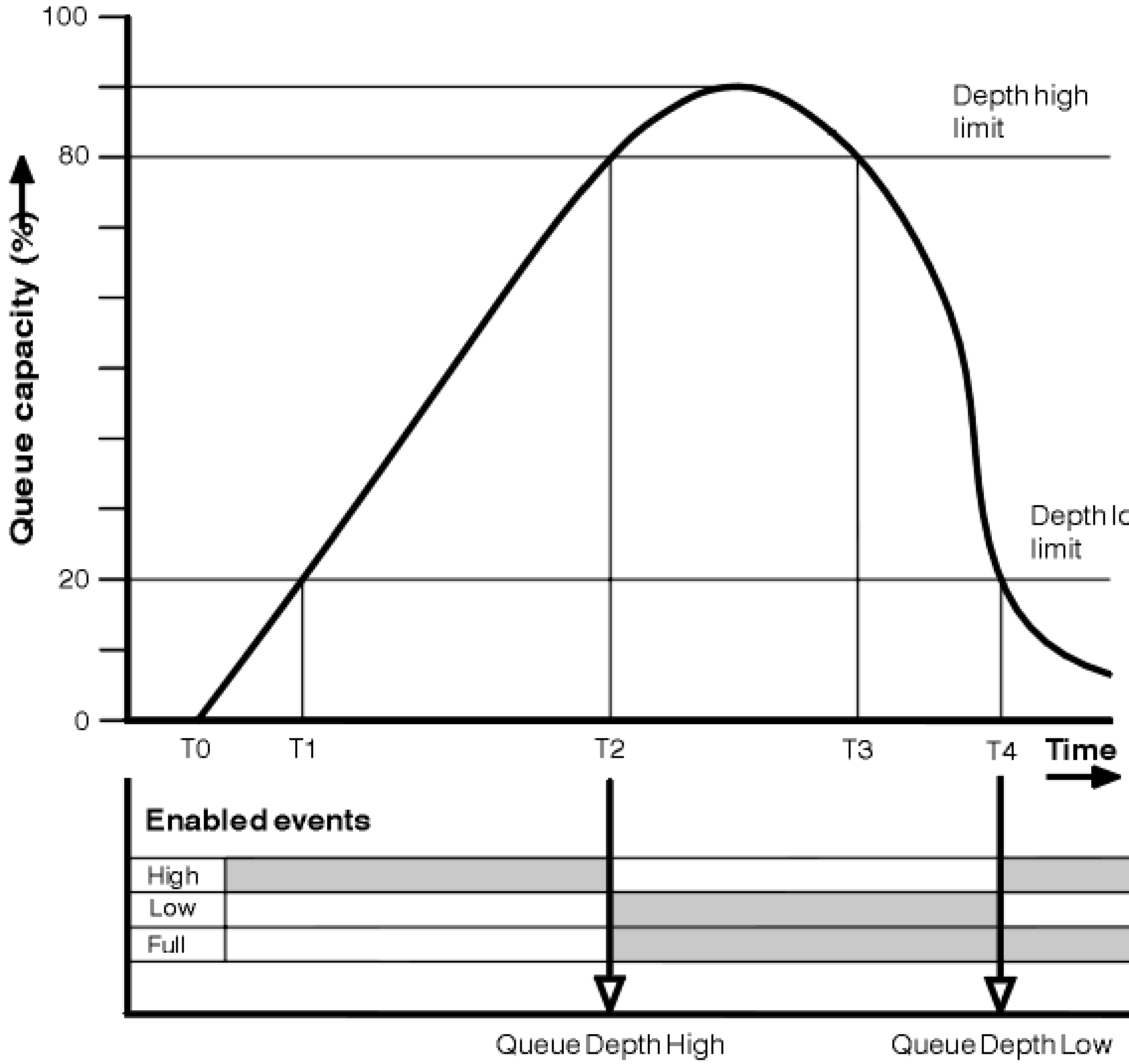


Figure 13. Queue depth events (1)

Commentary

1. At T(1), the queue depth is increasing (more MQPUT calls than MQGET calls) and crosses the Queue Depth Low limit. No event is generated at this time.
2. The queue depth continues to increase until T(2), when the depth high limit (80%) is reached and a Queue Depth High event is generated.

   This enables both Queue Full and Queue Depth Low events.
3. The (presumed) preventive actions instigated by the event prevent the queue from becoming full. By time T(3), the Queue Depth High limit has been reached again, this time from above. No event is generated at this time.
4. The queue depth continues to fall until T(4), when it reaches the depth low limit (20%) and a Queue Depth Low event is generated.

   This enables both Queue Full and Queue Depth High events.

Event statistics summary

Table 8 on page 49 summarizes the queue event statistics and Table 9 on page 49 summarizes which events are enabled.

Table 8. Event statistics summary for queue depth events (example 1)		
	Event 2	**Event 4**
Time of event	T(2)	T(4)
Type of event	Queue Depth High	Queue Depth Low
TimeSinceReset	T(2) - T(0)	T(4) - T(2)
HighQDepth (Maximum queue depth since reset)	800	900
MsgEnqCount	1157	1220
MsgDeqCount	357	1820

Table 9. Summary showing which events are enabled			
Time period	**Queue Depth High event**	**Queue Depth Low event**	**Queue Full event**
Before T(1)	ENABLED	-	-
T(1) to T(2)	ENABLED	-	-
T(2) to T(3)	-	ENABLED	ENABLED
T(3) to T(4)	-	ENABLED	ENABLED
After T(4)	ENABLED	-	ENABLED

Queue depth events: example 2
A more extensive sequence of queue depth events.

Figure 14 on page 50 shows the variation of queue depth over time.

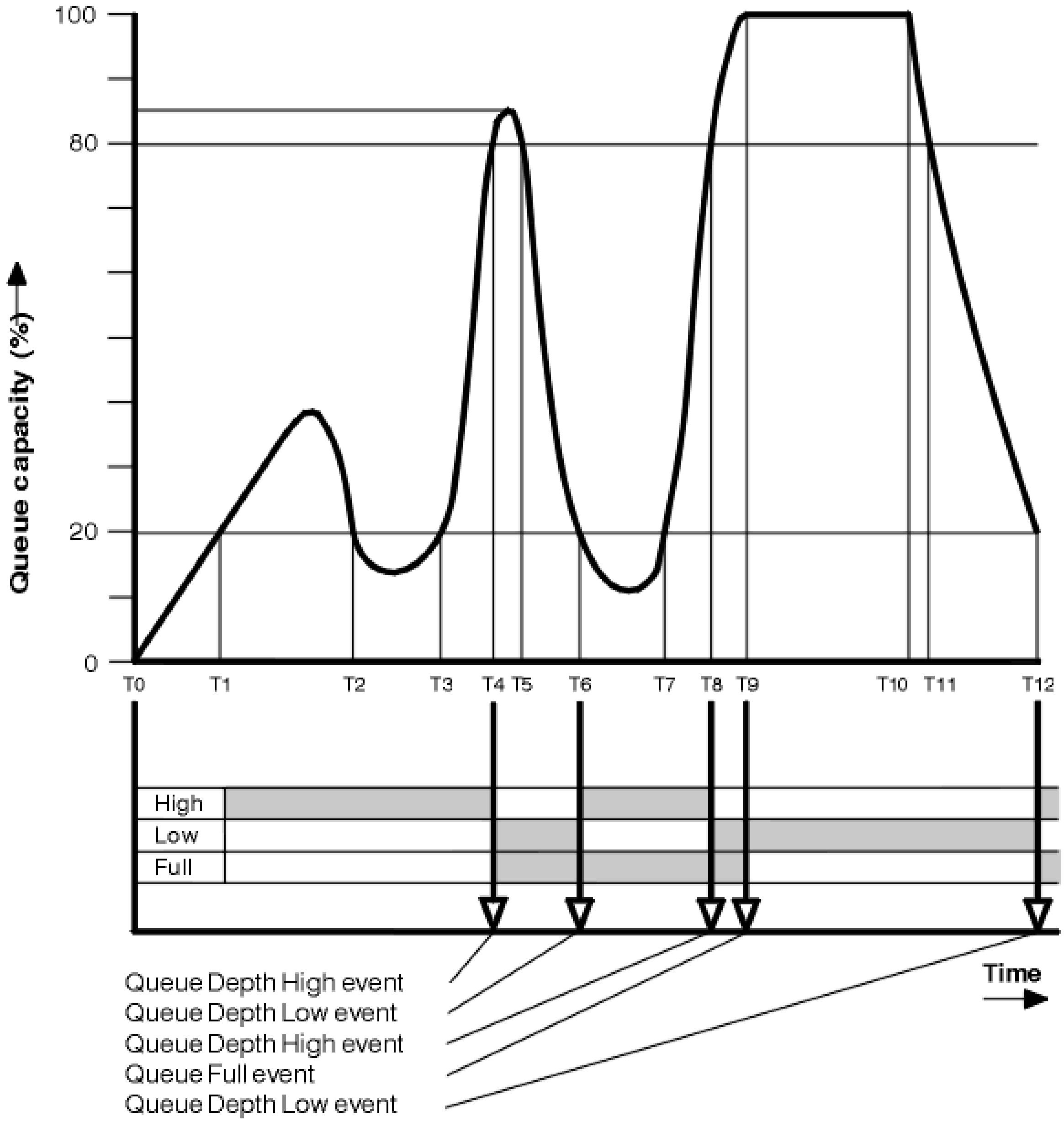


Figure 14. Queue depth events (2)

Commentary

1. No Queue Depth Low event is generated at the following times:
   - T(1) (Queue depth increasing, and not enabled)
   - T(2) (Not enabled)
   - T(3) (Queue depth increasing, and not enabled)
2. At T(4) a Queue Depth High event occurs. This enables both Queue Full and Queue Depth Low events.
3. At T(9) a Queue Full event occurs **after** the first message that cannot be put on the queue because the queue is full.
4. At T(12) a Queue Depth Low event occurs.

Event statistics summary

Table 10 on page 51 summarizes the queue event statistics and Table 11 on page 51 summarizes which events are enabled at different times for this example.

Table 10. Event statistics summary for queue depth events (example 2)

	Event 4	Event 6	Event 8	Event 9	Event 12
Time of event	T(4)	T(6)	T(8)	T(9)	T(12)
Type of event	Queue Depth High	Queue Depth Low	Queue Depth High	Queue Full	Queue Depth Low
TimeSinceReset	T(4) - T(0)	T(6) - T(4)	T(8) - T(6)	T(9) - T(8)	T(12) - T(9)
HighQDepth	800	855	800	1000	1000
MsgEnqCount	1645	311	1377	324	221
MsgDeqCount	845	911	777	124	1021

Table 11. Summary showing which events are enabled

Time period	Queue Depth High event	Queue Depth Low event	Queue Full event
T(0) to T(4)	ENABLED	-	-
T(4) to T(6)	-	ENABLED	ENABLED
T(6) to T(8)	ENABLED	-	ENABLED
T(8) to T(9)	-	ENABLED	ENABLED
T(9) to T(12)	-	ENABLED	-
After T(12)	ENABLED	-	ENABLED

Note: Events are out of syncpoint. Therefore you could have an empty queue, then fill it up causing an event, then roll back all of the messages under the control of a syncpoint manager. However, event enabling has been automatically set, so that the next time the queue fills up, no event is generated.

Configuration events

Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Configuration events notify you about changes to the attributes of an object. There are four types of configuration events:

- Create object events
- Change object events
- Delete object events
- Refresh object events

The event data contains the following information:

Origin information
: comprises the queue manager from where the change was made, the ID of the user that made the change, and how the change came about, for example by a console command.

Context information
: a replica of the context information in the message data from the command message.

Context information is included in the event data only when the command was entered as a message on the SYSTEM.COMMAND.INPUT queue.

Object identity
comprises the name, type and disposition of the object.

Object attributes
comprises the values of all the attributes in the object.

In the case of change object events, two messages are generated, one with the information before the change, the other with the information after.

Every configuration event message that is generated is placed on the queue SYSTEM.ADMIN.CONFIG.EVENT.

Related concepts

"Configuration events" on page 26
Configuration events are generated when a configuration event is requested explicitly, or automatically when an object is created, modified, or deleted.

Related reference

"Event types" on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information

Create object
Change object
Delete object
Refresh object

Configuration event generation

Use this page to view the commands that cause configuration events to be generated and to understand the circumstances in which configuration events are not generated

A configuration event message is put to the configuration event queue when the CONFIGEV queue manager attribute is ENABLED and

- any of the following commands, or their PCF equivalent, are issued:
  - DELETE AUTHINFO
  - DELETE CFSTRUCT
  - DELETE CHANNEL
  - DELETE NAMELIST
  - DELETE PROCESS
  - DELETE QMODEL/QALIAS/QREMOTE
  - DELETE STGCLASS
  - DELETE TOPIC
  - REFRESH QMGR
- any of the following commands, or their PCF equivalent, are issued even if there is no change to the object:
  - DEFINE/ALTER AUTHINFO
  - DEFINE/ALTER CFSTRUCT
  - DEFINE/ALTER CHANNEL
  - DEFINE/ALTER NAMELIST
  - DEFINE/ALTER PROCESS
  - DEFINE/ALTER QMODEL/QALIAS/QREMOTE

- DEFINE/ALTER STGCLASS
- DEFINE/ALTER TOPIC
- DEFINE MAXSMSGS
- SET CHLAUTH
- ALTER QMGR, unless the CONFIGEV attribute is DISABLED and is not changed to ENABLED

- any of the following commands, or their PCF equivalent, are issued for a local queue that is not temporary dynamic, even if there is no change to the queue.
  - DELETE QLOCAL
  - DEFINE/ALTER QLOCAL
- an MQSET call is issued, other than for a temporary dynamic queue, even if there is no change to the object.

When configuration events are not generated

Configuration events messages are not generated in the following circumstances:

- When a command or an MQSET call fails
- When a queue manager encounters an error trying to put a configuration event on the event queue, in which case the command or MQSET call completes, but no event message is generated
- For a temporary dynamic queue
- When internal changes are made to the TRIGGER queue attribute
- For the configuration event queue SYSTEM.ADMIN.CONFIG.EVENT, except by the REFRESH QMGR command
- For REFRESH/RESET CLUSTER and RESUME/SUSPEND QMGR commands that cause clustering changes
- When Creating or deleting a queue manager

Related concepts

"Configuration events" on page 51
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information

The MQSC commands
Introduction to Programmable Command Formats
MQSET - Set object attributes
MQSET - Set object attributes

Configuration event usage

Use this page to view how you can use configuration events to obtain information about your system, and to understand the factors, such as CMDSCOPE, that can affect your use of configuration events.

You can use configuration events for the following purposes:

1. To produce and maintain a central configuration repository, from which reports can be produced and information about the structure of the system can be generated.
2. To generate an audit trail. For example, if an object is changed unexpectedly, information regarding who made the alteration and when it was done can be stored.

   This can be particularly useful when command events are also enabled. If an MQSC or PCF command causes a configuration event and a command event to be generated, both event messages will share the same correlation identifier in their message descriptor.

For an MQSET call or any of the following commands:

- DEFINE object
- ALTER object

- DELETE object

if the queue manager attribute CONFIGEV is enabled, but the configuration event message cannot be put on the configuration event queue, for example the event queue has not been defined, the command or MQSET call is executed regardless.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the configuration event message or messages will be generated on the queue manager or queue managers where the command is executed, not where the command is entered. However, all the origin and context information in the event data will relate to the original command as entered, even where the command using CMDSCOPE is one that has been generated by the source queue manager.

Where a queue sharing group includes queue managers that are not at the current version, events will be generated for any command that is executed by means of CMDSCOPE on a queue manager that is at the current version, but not on those that are at a previous version. This happens even if the queue manager where the command is entered is at the previous version, although in such a case no context information is included in the event data.

Related concepts

"Configuration events" on page 51
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information

Introduction to Programmable Command Formats

MQSET - Set object attributes

MQSET - Set object attributes

Refresh Object configuration event

The Refresh Object configuration event is different from the other configuration events, because it occurs only when explicitly requested.

The create, change, and delete events are generated by an MQSET call or by a command to change an object but the refresh object event occurs only when explicitly requested by the MQSC command, REFRESH QMGR, or its PCF equivalent.

The REFRESH QMGR command is different from all the other commands that generate configuration events. All the other commands apply to a particular object and generate a single configuration event for that object. The REFRESH QMGR command can produce many configuration event messages potentially representing every object definition stored by a queue manager. One event message is generated for each object that is selected.

The REFRESH QMGR command uses a combination of three selection criteria to filter the number of objects involved:

- Object Name
- Object Type
- Refresh Interval

If you specify none of the selection criteria on the REFRESH QMGR command, the default values are used for each selection criteria and a refresh configuration event message is generated for every object definition stored by the queue manager. This might cause unacceptable processing times and event message generation. Consider specifying some selection criteria.

The REFRESH QMGR command that generates the refresh events can be used in the following situations:

- When configuration data is wanted about all or some of the objects in a system regardless of whether the objects have been recently manipulated, for example, when configuration events are first enabled.

Consider using several commands, each with a different selection of objects, but such that all are included.

- If there has been an error in the SYSTEM.ADMIN.CONFIG.EVENT queue. In this circumstance, no configuration event messages are generated for Create, Change, or Delete events. When the error on the queue has been corrected, the Refresh Queue Manager command can be used to request the generation of event messages, which were lost while there was an error in the queue. In this situation consider setting the refresh interval to the time for which the queue was unavailable.

Related concepts

"Configuration events" on page 51
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information

REFRESH QMGR

Refresh Queue Manager

Command events

Command events are notifications that an MQSC, or PCF command has run successfully.

The event data contains the following information:

Origin information
comprises the queue manager from where the command was issued, the ID of the user that issued the command, and how the command was issued, for example by a console command.

Context information
a replica of the context information in the message data from the command message. If a command is not entered using a message, context information is omitted.

Context information is included in the event data only when the command was entered as a message on the SYSTEM.COMMAND.INPUT queue.

Command information
the type of command that was issued.

Command data

- for PCF commands, a replica of the command data
- for MQSC commands, the command text

The command data format does not necessarily match the format of the original command. For example, on distributed platforms the command data format is always in PCF format, even if the original request was an MQSC command.

Every command event message that is generated is placed on the command event queue, SYSTEM.ADMIN.COMMAND.EVENT.

Related reference

"Event types" on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information

Command

Command event generation

Use this page to view the situations that cause command events to be generated and to understand the circumstances in which command events are not generated

A command event message is generated in the following situations:

- When the CMDEV queue manager attribute is specified as ENABLED and an MQSC or PCF command runs successfully.
- When the CMDEV queue manager attribute is specified as NODISPLAY and any command runs successfully, with the exception of DISPLAY commands (MQSC), and Inquire commands (PCF).
- When you run the MQSC command, ALTER QMGR, or the PCF command, Change Queue Manager, and the CMDEV queue manager attribute meets either of the following conditions:
  - CMDEV is not specified as DISABLED after the change
  - CMDEV was not specified as DISABLED before the change

If a command runs against the command event queue, SYSTEM.ADMIN.COMMAND.EVENT, a command event is generated if the queue still exists and it is not put-inhibited.

When command events are not generated

A command event message is not generated in the following circumstances:

- When a command fails
- When a queue manager encounters an error trying to put a command event on the event queue, in which case the command runs regardless, but no event message is generated
- For the MQSC command REFRESH QMGR TYPE (EARLY)
- For the MQSC command START QMGR MQSC
- For the MQSC command SUSPEND QMGR, if the parameter LOG is specified
- For the MQSC command RESUME QMGR, if the parameter LOG is specified

Related concepts
"Command events" on page 55
Command events are notifications that an MQSC, or PCF command has run successfully.

Related information
REFRESH QMGR
START QMGR
SUSPEND QMGR
RESUME QMGR
SUSPEND QMGR, RESUME QMGR and clusters

Command event usage

Use this page to view how you can use command events to generate an audit trail of the commands that have run

For example, if an object is changed unexpectedly, information regarding who made the alteration and when it was done can be stored. This can be particularly useful when configuration events are also enabled. If an MQSC or PCF command causes a command event and a configuration event to be generated, both event messages will share the same correlation identifier in their message descriptor.

If a command event message is generated, but cannot be put on the command event queue, for example if the command event queue has not been defined, the command for which the command event was generated still runs regardless.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the command event message or messages will be generated on the queue manager or queue managers where the command runs, not where the command is entered. However, all the origin and context information in the event data will relate to the original command as entered, even where the command using CMDSCOPE is one that has been generated by the source queue manager.

Related concepts

“Command events” on page 55

Command events are notifications that an MQSC, or PCF command has run successfully.

“Command event generation” on page 55

Use this page to view the situations that cause command events to be generated and to understand the circumstances in which command events are not generated

Related information

The MQSC commands

PCF commands and responses in groups

Logger events

Logger events are notifications that a queue manager has started writing to a new log extent IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

The event data contains the following information:

- The name of the current log extent.
- The name of the earliest log extent needed for restart recovery.
- The name of the earliest log extent needed for media recovery.
- The directory in which the log extents are located.

Every logger event message that is generated is placed on the logger event queue, SYSTEM.ADMIN.LOGGER.EVENT.

Related reference

“Event types” on page 21

Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information

Logger

Logger event generation

Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

A logger event message is generated in the following situations:

- When the LOGGEREV queue manager attribute is specified as ENABLED and the queue manager starts writing to a new log extent or, on IBM i, a journal receiver.
- When the LOGGEREV queue manager attribute is specified as ENABLED and the queue manager starts.
- When the LOGGEREV queue manager attribute is changed from DISABLED to ENABLED.

Tip: You can use the RESET QMGR MQSC command to request a queue manager to start writing to a new log extent.

When logger events are not generated

A logger event message is not generated in the following circumstances:

- When a queue manager is configured to use circular logging.

  In this case, the LOGGEREV queue manager attribute is set as DISABLED and cannot be altered.
- When a queue manager encounters an error trying to put a logger event on the event queue, in which case the action that caused the event completes, but no event message is generated.

Related concepts

“Logger events” on page 57

Logger events are notifications that a queue manager has started writing to a new log extent IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

Related information

LoggerEvent (MQLONG)

LoggerEvent (10-digit signed integer)

RESET QMGR

Logger event usage

Use this page to view how you can use logger events to determine the log extents that are no longer required for queue manager restart, or media recovery.

You can archive superfluous log extents to a medium such as tape for disaster recovery before removing them from the active log directory. Regular removal of superfluous log extents keeps disk space usage to a minimum.

If the LOGGEREV queue manager attribute is enabled, but a logger event message cannot be put on the logger event queue, for example because the event queue has not been defined, the action that caused the event continues regardless.

Related concepts

“Logger events” on page 57

Logger events are notifications that a queue manager has started writing to a new log extent IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

Related reference

“Logger event generation” on page 57

Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

Related information

LoggerEvent (MQLONG)

LoggerEvent (10-digit signed integer)

Sample program to monitor the logger event queue

Use this page to view a sample C program that monitors the logger event queue for new event messages, reads those messages, and puts the contents of the message to stdout.

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSLOG0.C                                                   */
/*                                                                            */
/* Description:  Sample C program to monitor the logger event queue and output*/
/*               a message to stdout when a logger event occurs               */
/* <N_OCO_COPYRIGHT>                                                          */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 2005, 2022. All Rights Reserved.                   */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/* <NOC_COPYRIGHT>                                                            */
/******************************************************************************/
/*                                                                            */
/* Function: AMQSLOG is a sample program which monitors the logger event      */
/* queue for new event messages, reads those messages, and puts the contents  */
/* of the message to stdout.                                                  */
```

```
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSLOG has 1 parameter - the queue manager name (optional, if not         */
/* specified then the default queue manager is implied)                       */
/*                                                                            */
/******************************************************************************/

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

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

/******************************************************************************/
/* Constants                                                                  */
/******************************************************************************/

#define   MAX_MESSAGE_LENGTH   8000

typedef struct _ParmTableEntry
{
  MQLONG  ConstVal;
  PMQCHAR Desc;
} ParmTableEntry;

ParmTableEntry ParmTable[] =
{
  0                             ,"",
  MQCA_Q_MGR_NAME               ,"Queue Manager Name",
  MQCMD_LOGGER_EVENT            ,"Logger Event Command",
  MQRC_LOGGER_STATUS            ,"Logger Status",
  MQCACF_CURRENT_LOG_EXTENT_NAME,"Current Log Extent",
  MQCACF_RESTART_LOG_EXTENT_NAME,"Restart Log Extent",
  MQCACF_MEDIA_LOG_EXTENT_NAME  ,"Media Log Extent",
  MQCACF_LOG_PATH               ,"Log Path"};

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                       PMQCHAR    pBuffer);

static PMQCHAR ParmToString(MQLONG Parameter);

/**********************************************************************/
/* Function: main                                                     */
/**********************************************************************/
int main(int argc, char * argv[])
{
  MQLONG    CompCode;
  MQLONG    Reason;
  MQHCONN   hConn = MQHC_UNUSABLE_HCONN;
  MQOD      ObjDesc = { MQOD_DEFAULT };
  MQCHAR    QMName[MQ_Q_MGR_NAME_LENGTH+1] = "";
  MQCHAR    LogEvQ[MQ_Q_NAME_LENGTH] = "SYSTEM.ADMIN.LOGGER.EVENT";
  MQHOBJ    hEventQueue;
  PMQCHAR   pBuffer = NULL;

  printf("\n/*************************************/\n");
  printf("/* Sample Logger Event Monitor start */\n");
  printf("/*************************************/\n");

  /********************************************************************/
  /* Parse any command line options                                   */
  /********************************************************************/

  if (argc > 1)
     strncpy(QMName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);

  pBuffer = (char *)malloc(MAX_MESSAGE_LENGTH);
  if (!pBuffer)
  {
    printf("Can't allocate %d bytes\n",MAX_MESSAGE_LENGTH);
    goto MOD_EXIT;
  }
```

```
  /****************************************************************/
  /* Connect to the specified (or default) queue manager          */
  /****************************************************************/

  MQCONN(QMName,
         &hConn,
         &CompCode,
         &Reason);

  if (Reason != MQCC_OK)
  {
    printf("Error in call to MQCONN, Reason %d, CompCode %d\n", Reason,
    CompCode);
    goto MOD_EXIT;
  }

  /* Open the logger event queue for input  */

  strncpy(ObjDesc.ObjectQMgrName,QMName, MQ_Q_MGR_NAME_LENGTH);
  strncpy(ObjDesc.ObjectName, LogEvQ, MQ_Q_NAME_LENGTH);

  MQOPEN(  hConn,
          &ObjDesc,
           MQOO_INPUT_EXCLUSIVE,
          &hEventQueue,
          &CompCode,
          &Reason);
  if (Reason)
  {
    printf("MQOPEN failed for queue manager %.48s Queue %.48s Reason: %d\n",
                                          ObjDesc.ObjectQMgrName,
                                  ObjDesc.ObjectName,
                                  Reason);
    goto MOD_EXIT;
  }
  else
  {
    ProcessPCF(hConn, hEventQueue, pBuffer);
  }

  MOD_EXIT:

  if (pBuffer != NULL) {
    free(pBuffer);
  }

  /****************************************************************/
  /* Disconnect                                                   */
  /****************************************************************/
  if (hConn != MQHC_UNUSABLE_HCONN) {
     MQDISC(&hConn, &CompCode, &Reason);
  }

  return 0;
}

/******************************************************************************/
/* Function: ProcessPCF                                                       */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to queue manager connection                      */
/*                    Handle to the opened logger event queue object          */
/*                    Pointer to a memory buffer to store the incoming PCF msg*/
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Wait for messages to appear on the logger event queue and display   */
/* their contents.                                                            */
/*                                                                            */
/******************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                       PMQCHAR    pBuffer)
{
  MQCFH   * pCfh;
  MQCFST  * pCfst;
  MQGMO     Gmo     = { MQGMO_DEFAULT };
  MQMD      Mqmd    = { MQMD_DEFAULT };
  PMQCHAR   pPCFCmd;
  MQLONG    Reason  = 0;
```

```
MQLONG    CompCode;
MQLONG    MsgLen;
PMQCHAR   Parm = NULL;
                                       /* Set timeout value           */
Gmo.Options     |= MQGMO_WAIT;
Gmo.Options |= MQGMO_CONVERT;
Gmo.WaitInterval = MQWI_UNLIMITED;
/******************************************************************/
/* Process response Queue                                         */
/******************************************************************/
while (Reason == MQCC_OK)
{
  memcpy(&Mqmd.MsgId;   , MQMI_NONE, sizeof(Mqmd.MsgId));
  memset(&Mqmd.CorrelId, 0, sizeof(Mqmd.CorrelId));

  MQGET( hConn,
         hEventQueue,
        &Mqmd,
        &Gmo,
         MAX_MESSAGE_LENGTH,
         pBuffer,
        &MsgLen,
        &CompCode,
        &Reason);
  if (Reason != MQCC_OK)
  {
    switch(Reason)
    {
      case MQRC_NO_MSG_AVAILABLE:
           printf("Timed out");
           break;

      default:
           printf("MQGET failed RC(%d)\n", Reason);
           break;
    }
    goto MOD_EXIT;
  }

  /****************************************************************/
  /* Only expect PCF event messages on this queue                 */
  /****************************************************************/
  if (memcmp(Mqmd.Format, MQFMT_EVENT, sizeof(Mqmd.Format)))
  {
   printf("Unexpected message format '%8.8s' received\n",Mqmd.Format);
   continue;
  }


  /*****************************************************************/
  /* Build the output by parsing the received PCF message, first the */
  /* header, then each of the parameters                           */
  /*****************************************************************/

  pCfh = (MQCFH *)pBuffer;

  if (pCfh -> Reason)
  {
   printf("-------------------------------------------------------------\n");
   printf("Event Message Received\n");

   Parm = ParmToString(pCfh->Command);
   if (Parm != NULL) {
     printf("Command  :%s \n",Parm);
   }
   else
   {
     printf("Command  :%d \n",pCfh->Command);
   }

   printf("CompCode :%d\n"   ,pCfh->CompCode);

   Parm = ParmToString(pCfh->Reason);
   if (Parm != NULL) {
     printf("Reason   :%s \n",Parm);
   }
   else
   {
     printf("Reason   :%d \n",pCfh->Reason);
   }
  }
```

```
    pPCFCmd  = (char *)  (pCfh+1);
    printf("-----------------------------------------------------------------\n");
    while(pCfh -> ParameterCount--)
    {
      pCfst = (MQCFST *) pPCFCmd;
      switch(pCfst -> Type)
      {
        case MQCFT_STRING:
             Parm = ParmToString(pCfst -> Parameter);
             if (Parm != NULL) {
               printf("%-32s",Parm);
             }
             else
             {
               printf("%-32d",pCfst -> Parameter);
             }

             fwrite( pCfst -> String, pCfst -> StringLength, 1, stdout);
             pPCFCmd += pCfst -> StrucLength;
             break;
        default:
             printf("Unrecoginised datatype %d returned\n",pCfst->Type);
             goto MOD_EXIT;
      }
      putchar('\n');
    }
    printf("-----------------------------------------------------------------\n");
  }
MOD_EXIT:

 return;
}

/******************************************************************************/
/* Function: ParmToString                                                     */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Parameter for which to get string description           */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Takes a parameter as input and returns a pointer to a string        */
/* description for that parameter, or NULL if the parameter does not  */
/* have an associated string description                                      */
/******************************************************************************/

static PMQCHAR ParmToString(MQLONG Parameter){
  long i;
  for (i=0 ; i< sizeof(ParmTable)/sizeof(ParmTableEntry); i++)
  {
    if (ParmTable[i].ConstVal == Parameter ParmTable[i].Desc)
      return ParmTable[i].Desc;
  }
  return NULL;
}
```

Sample output

This application produces the following form of output:

```
/************************************/
/* Sample Logger Event Monitor start */
/************************************/
-----------------------------------------------------------------
Event Message Received
Command  :Logger Event Command
CompCode :0
Reason   :Logger Status
-----------------------------------------------------------------
Queue Manager Name              CSIM

Current Log Extent              AMQA000001
Restart Log Extent              AMQA000001
Media Log Extent                AMQA000001
```

```
Log Path                         QMCSIM
-----------------------------------------------------------------
```

Related concepts

“Logger event usage” on page 58
Use this page to view how you can use logger events to determine the log extents that are no longer required for queue manager restart, or media recovery.

“Command event usage” on page 56
Use this page to view how you can use command events to generate an audit trail of the commands that have run

Related reference

“Logger event generation” on page 57
Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

V 8.0.0.4 Authority configuration events

Authority configuration events are output when a change is made from any of the security control operations through the command line, MQSC, PCF, or corresponding iSeries commands.

The event data contains the following information:

Origin information
: comprises the queue manager from where the change was made, the ID of the user that made the change, and how the change came about, for example by a console command.

Context information
: a replica of the context information in the message data from the command message.

  Context information is included in the event data when the command was entered as a message on the SYSTEM.ADMIN.COMMAND.QUEUE queue.

Authority Record identity
: comprises the profile name, and object type of the authority record.

Object attributes
: comprises the values of all the attributes in the authority record.

  In the case of change authority record events, two messages are generated, one with the information before the change, the other with the information after the change.

  Every event message that is generated is placed on the SYSTEM.ADMIN.CONFIG.EVENT queue.

Related reference

“Event types” on page 21
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

V 8.0.0.4 *Authority configuration event generation*

Use this page to view the situations that cause authority configuration events to be generated, and to understand the circumstances in which authority configuration events are not generated.

Authority configuration events notify you about changes to the attributes of an authority record. There are three types of authority configuration event:

- Change Authority Record
- Delete Authority Record
- Refresh Authority Record

An authority event message is put to the configuration event queue, when the **CONFIGEV** queue manager attribute is set to *ENABLED* and any of the following commands, or their MQSC equivalent, are issued, even if there is no actual change to the authority record:

- Delete Authority Record PCF command
- Set Authority Record PCF command
- setmqaut control command
- RVKMQMAUT CL command
- GRTMQMAUT CL command

When authority configuration events are not generated

The authority configuration event messages are not generated in the following circumstances:

- When a command fails
- When a queue manager encounters an error trying to put a message on the event queue, in which case the command completes, but no event message is generated
- When creating or deleting a queue manager
- When an object is deleted regardless of the **AUTHREC** option on the delete command. The corresponding command event shows that operation, which does not apply to the authority record for individual users.

Related concepts

"Command events" on page 55
Command events are notifications that an MQSC, or PCF command has run successfully.

Related information

REFRESH QMGR

Sample program to monitor instrumentation events

V 8.0.0.4 **amqsevt** formats the instrumentation events that a queue manager can create, and is supplied with IBM MQ. The program reads messages from event queues, and formats them into readable strings.

V 8.0.0.4

As a sample program, both source and binary are provided. The sample is provided on all the distributed platforms, including IBM i.

The single binary file `amqsevt` (or `amqsevt.exe`) is shipped in the samples fileset and is installed in the samples bin (`tools\c\samples\bin` or `bin64`) directory.

The source files `amqsevta.c` is also shipped in the samples fileset, and is installed in the samples directory, that is, `tools\c\samples` on Windows.

Note that the program can read from multiple event queues, and subscribe to multiple topics, by using MQCB to retrieve the messages.

When running as a client, the sample can connect to any queue manager including z/OS.

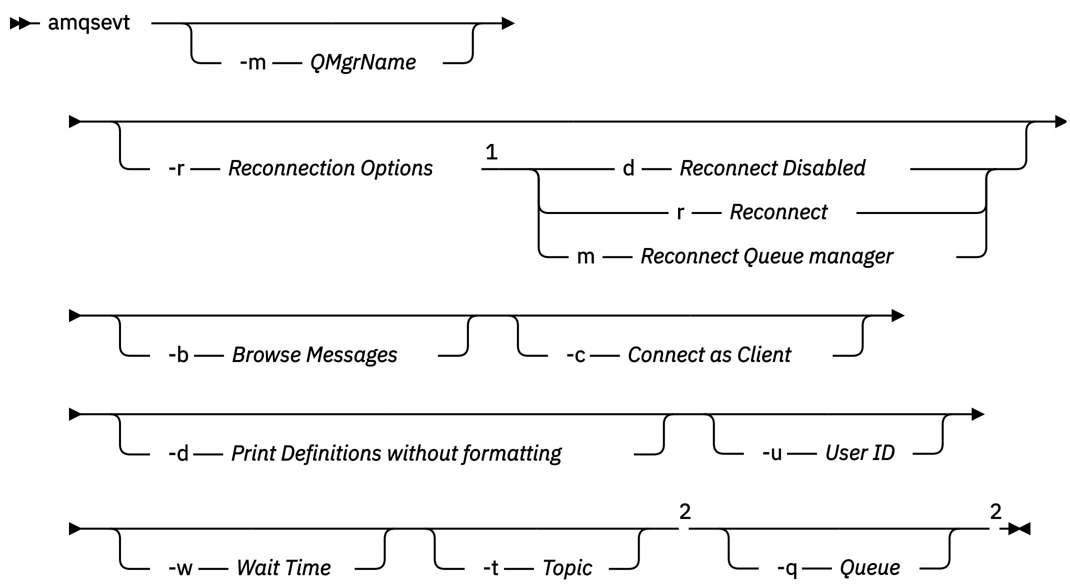
Attention: You can use the program without specifying any parameters, in which case the program attempts to connect to the default queue manager and read messages from the standard set of event queues (SYSTEM.ADMIN.*.EVENT).

In this situation, the program waits forever for messages, until you press the Enter key to end the program.

However, you are more likely to use the program with the various options described.

Syntax

V 8.0.0.4



Notes:

[1] Available only when being used as a client

[2] Queues and topics can have multiple entries

Optional parameters

V 8.0.0.4

-m *QueueManagerName*
: Specify a specific queue manager for reading events.

-r *Reconnection Options*
: Auto reconnection options when used as a client. The possible values are:

  d
  : Reconnect the client disabled

  r
  : Reconnect the client

  m
  : Reconnect the queue manager

-b
: Browse records only, rather than destructively reading the messages

-c
: Selects connection as a client.

-d
: Selects the printing mode used in the second example. The MQI constants are printed exactly as they appear in the header files.

-u *User ID*
: Specify a specific user and causes a prompt to appear requesting a password

-w *Wait*
: Causes the program to exit if no event messages have arrived within the number of seconds specified.

  Note that, if you do not specify a time, the program only ends normally when you press the Enter key.

-t *Topic* and
-q *Queue*
: Both the **-q** and **-t** options can be given multiple times on the command line.

Therefore, it is possible to read from some standard queues and also from topics (if events are being sent to them) from a single run of the program.

If no queues or topics are named on the command line, the default event queues are opened.

Note: The program detects if it has connected to a z/OS queue manager as a client, and changes the default set of event queues appropriately, as z/OS does not have the SYSTEM.ADMIN.LOGGER.EVENT queue.

When topics are used, the program uses a non-durable subscription with a managed queue so that everything gets cleaned out when it exits.

Sample output

V 8.0.0.4

The following two examples show the output from the program.

The first example uses the default formatting option where the program takes the MQI definition of a field and formats the output to make the output more readable.

```
**** Message (320 Bytes) on Queue SYSTEM.ADMIN.QMGR.EVENT ****
Event Type                       : Queue Mgr Event
Reason                           : Unknown Alias Base Queue
Event created                    : 2015/06/17 13:47:07.02 GMT
  Queue Mgr Name                 : V8003_A
  Queue Name                     : EVT.NO.BASE.QUEUE
  Base Object Name               : EVT.NOT.DEFINED
  Appl Type                      : Unix
  Appl Name                      : amqsput
  Base Type                      : Queue
```

The second example shows the alternative formatting, using the -d option, that does not try to translate MQI constants. This might be preferable for some scripting tools that look for specific MQI values.

```
**** Message (320 Bytes) on Queue SYSTEM.ADMIN.QMGR.EVENT ****
Event Type                       : MQCMD_Q_MGR_EVENT
Reason                           : MQRC_UNKNOWN_ALIAS_BASE_Q
Event created                    : 2015/06/17 13:52:48.18 GMT
  MQCA_Q_MGR_NAME                : V8003_A
  MQCA_Q_NAME                    : EVT.NO.BASE.QUEUE
  MQCA_BASE_OBJECT_NAME          : EVT.NOT.DEFINED
  MQIA_APPL_TYPE                 : MQAT_UNIX
  MQCACF_APPL_NAME               : amqsput
  MQIA_BASE_TYPE                 : MQOT_Q
```

Example usage

V 8.0.0.4

The following example shows you how to use more than one queue:

```
amqsevt -m QM1 -q SYSTEM.ADMIN.QMGR.EVENT -q SYSTEM.ADMIN.PERM.EVENT -w 1
```

Related concepts

“Event monitoring” on page 18
Event monitoring is the process of detecting occurrences of *instrumentation events* in a queue manager network. An instrumentation event is a logical combination of events that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an *event message*, on an event queue.

“Instrumentation events” on page 19

An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

Related reference

"Sample program to monitor the logger event queue" on page 58
Use this page to view a sample C program that monitors the logger event queue for new event messages, reads those messages, and puts the contents of the message to stdout.

Related information

C programming

Message monitoring

Message monitoring is the process of identifying the route a message has taken through a queue manager network. By identifying the types of activities, and the sequence of activities performed on behalf of a message, the message route can be determined.

As a message passes through a queue manager network, various processes perform activities on behalf of the message. Use one of the following techniques to determine a message route:

- The IBM MQ display route application (dspmqrte)
- Activity recording
- Trace-route messaging

These techniques all generate special messages that contain information about the activities performed on the message as it passed through a queue manager network. Use the information returned in these special messages to achieve the following objectives:

- Record message activity.
- Determine the last known location of a message.
- Detect routing problems in your queue manager network.
- Assist in determining the causes of routing problems in your queue manager network.
- Confirm that your queue manager network is running correctly.
- Familiarize yourself with the running of your queue manager network.
- Trace published messages.

Related information

Types of message

Activities and operations

Activities are discrete actions that an application performs on behalf of a message. Activities consist of operations, which are single pieces of work that an application performs.

The following actions are examples of activities:

- A message channel agent (MCA) sends a message from a transmission queue down a channel
- An MCA receives a message from a channel and puts it on its target queue
- An application getting a message from a queue, and putting a reply message in response.
- The IBM MQ publish/subscribe engine processes a message.

Activities consist of one or more *operations*. Operations are single pieces of work that an application performs. For example, the activity of an MCA sending a message from a transmission queue down a channel consists of the following operations:

1. Getting a message from a transmission queue (a *Get* operation).
2. Sending the message down a channel (a *Send* operation).

In a publish/subscribe network, the activity of the IBM MQ publish/subscribe engine processing a message can consist of the following multiple operations:

1. Putting a message to a topic string (a *Put* operation).
2. Zero or more operations for each of the subscribers that are considered for receipt of the message (a *Publish* operation, a *Discarded Publish* operation or an *Excluded Publish* operation).

Information from activities

You can identify the sequence of activities performed on a message by recording information as the message is routed through a queue manager network. You can determine the route of a message through the queue manager network from the sequence of activities performed on the message, and can obtain the following information:

The last known location of a message
If a message does not reach its intended destination, you can determine the last known location of the message from a complete or partial message route.

Configuration issues with a queue manager network
When studying the route of a message through a queue manager network, you might see that the message has not gone where expected. There are many reasons why this can occur, for example, if a channel is inactive, the message might take an alternative route.

For a publish/subscribe application, you can also determine the route of a message being published to a topic and any messages that flow in a queue manager network as a result of being published to subscribers.

In such situations, a system administrator can determine whether there are any problems in the queue manager network, and if appropriate, correct them.

Message routes

Depending on your reason for determining a message route, you can use the following general approaches:

Using activity information recorded for a trace-route message
Trace-route messages record activity information for a specific purpose. You can use them to determine configuration issues with a queue manager network, or to determine the last known location of a message. If a trace-route message is generated to determine the last known location of a message that did not reach its intended destination, it can mimic the original message. This gives the trace-route message the greatest chance of following the route taken by the original message.

The IBM MQ display route application can generate trace-route messages.

Using activity information recorded for the original message
You can enable any message for activity recording and have activity information recorded on its behalf. If a message does not reach its intended destination, you can use the recorded activity information to determine the last known location of the message. By using activity information from the original message, the most accurate possible message route can be determined, leading to the last known location. To use this approach, the original message must be enabled for activity recording.

Warning: Avoid enabling all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase to an unacceptable level.

Related concepts

"Message monitoring" on page 67
Message monitoring is the process of identifying the route a message has taken through a queue manager network. By identifying the types of activities, and the sequence of activities performed on behalf of a message, the message route can be determined.

"Message route techniques" on page 69

Activity recording and trace-route messaging are techniques that allow you to record activity information for a message as it is routed through a queue manager network.

"Trace-route messaging" on page 75
Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

Related information

Writing your own message channel agents

Message route techniques

Activity recording and trace-route messaging are techniques that allow you to record activity information for a message as it is routed through a queue manager network.

Activity recording

If a message has the appropriate report option specified, it requests that applications generate *activity reports* as it is routed through a queue manager network. When an application performs an activity on behalf of a message, an activity report can be generated, and delivered to an appropriate location. An activity report contains information about the activity that was performed on the message.

The activity information collected using activity reports must be arranged in order before a message route can be determined.

Trace-route messaging

Trace-route messaging is a technique that involves sending a *trace-route message* into a queue manager network. When an application performs an activity on behalf of the trace-route message, activity information can be accumulated in the message data of the trace-route message, or activity reports can be generated. If activity information is accumulated in the message data of the trace-route message, when it reaches its target queue a trace-route reply message containing all the information from the trace-route message can be generated and delivered to an appropriate location.

Because a trace-route message is dedicated to recording the sequence of activities performed on its behalf, there are more processing options available compared with normal messages that request activity reports.

Comparison of activity recording and trace-route messaging

Both activity recording and trace-route messaging can provide activity information to determine the route a message has taken through a queue manager network. Both methods have their own advantages.

Benefit	Activity recording	Trace-route messaging
Can determine the last known location of a message	Yes	Yes
Can determine configuration issues with a queue manager network	Yes	Yes
Can be requested by any message (is not restricted to use with trace-route messages)	Yes	No
Message data is left unmodified	Yes	No
Message processed normally	Yes	No
Activity information can be accumulated in the message data	No	Yes
Optional message delivery to target queue	No	Yes
If a message is caught in an infinite loop, it can be detected and dealt with	No	Yes
Activity information can be put in order reliably	No	Yes

Benefit	Activity recording	Trace-route messaging
Application provided to display the activity information	No	Yes

Message route completeness

In some cases it is not possible to identify the full sequence of activities performed on behalf of a message, so only a partial message route can be determined. The completeness of a message route is directly influenced by the queue manager network that the messages are routed through. The completeness of a message route depends on the level of the queue managers in the queue manager network, as follows:

Queue managers at IBM WebSphere MQ 6.0 and subsequent releases
MCAs and user-written applications connected to queue managers at IBM WebSphere MQ 6.0 or subsequent releases can record information related to the activities performed on behalf of a message. The recording of activity information is controlled by the queue manager attributes ACTIVREC and ROUTEREC. If a queue manager network consists of queue managers at IBM WebSphere MQ 6.0 or subsequent releases only, complete message routes can be determined.

IBM MQ queue managers before Version 6.0
Applications connected to IBM MQ queue managers before Version 6.0 **do not** record the activities that they have performed on behalf of a message. If a queue manager network contains any IBM MQ queue manager prior to Version 6.0, only a partial message route can be determined.

How activity information is stored

IBM MQ stores activity information in activity reports, trace-route messages, or trace-route reply messages. In each case the information is stored in a structure called the *Activity* PCF group. A trace-route message or trace-route reply message can contain many Activity PCF groups, depending on the number of activities performed on the message. Activity reports contain one Activity PCF group because a separate activity report is generated for every recorded activity.

With trace-route messaging, additional information can be recorded. This additional information is stored in a structure called the *TraceRoute* PCF group. The TraceRoute PCF group contains a number of PCF structures that are used to store additional activity information, and to specify options that determine how the trace-route message is handled as it is routed through a queue manager network.

Related concepts

"Activity recording" on page 71
Activity recording is a technique for determining the routes that messages take through a queue manager network. To determine the route that a message has taken, the activities performed on behalf of the message are recorded.

"Trace-route messaging" on page 75
Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

Related reference

"The TraceRoute PCF group" on page 81
Attributes in the *TraceRoute* PCF group control the behavior of a trace-route message. The *TraceRoute* PCF group is in the message data of every trace-route message.

"Activity report message data" on page 114

Use this page to view the parameters contained by the *Activity* PCF group in an activity report message. Some parameters are returned only when specific operations have been performed.

Activity recording

Activity recording is a technique for determining the routes that messages take through a queue manager network. To determine the route that a message has taken, the activities performed on behalf of the message are recorded.

When using activity recording, each activity performed on behalf of a message can be recorded in an activity report. An activity report is a type of report message. Each activity report contains information about the application that performed the activity on behalf of the message, when the activity took place, and information about the operations that were performed as part of the activity. Activity reports are typically delivered to a reply-to queue where they are collected together. By studying the activity reports related to a message, you can determine the route that the message took through the queue manager network.

Activity report usage

When messages are routed through a queue manager network, activity reports can be generated. You can use activity report information in the following ways:

Determine the last known location of a message
: If a message that is enabled for activity recording does not reach its intended destination, activity reports generated for the message as it was routed through a queue manager network can be studied to determine the last known location of the message.

Determine configuration issues with a queue manager network
: A number of messages enabled for activity recording can be sent into a queue manager network. By studying the activity reports related to each message it can become apparent that they have not taken the expected route. There are many reasons why this can occur, for example, a channel could have stopped, forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

Note: You can use activity recording in conjunction with trace-route messages by using the IBM MQ display route application.

Activity report format

Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message. Activity reports are standard IBM MQ report messages containing a message descriptor and message data, as follows:

The message descriptor

- An MQMD structure

Message data

- An embedded PCF header (MQEPH)
- Activity report message data

Activity report message data consists of the *Activity* PCF group, and if generated for a trace-route message, the *TraceRoute* PCF group.

Related information

MQMD - Message descriptor

MQEPH - Embedded PCF header

Controlling activity recording

Enable activity recording at the queue manager level. To enable an entire queue manager network, individually enable every queue manager in the network for activity recording. If you enable more queue managers, more activity reports are generated.

About this task

To generate activity reports for a message as it is routed through a queue manager: define the message to request activity reports; enable the queue manager for activity recording; and ensure that applications performing activities on the message are capable of generating activity reports.

If you do *not* want activity reports to be generated for a message as it is routed through a queue manager, *disable* the queue manager for activity recording.

Procedure

1. Request activity reports for a message
   a) In the message descriptor of the message, specify MQRO_ACTIVITY in the *Report* field.
   b) In the message descriptor of the message, specify the name of a reply-to queue in the *ReplyToQ* field.

   Warning: Avoid enabling all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase to an unacceptable level.
2. Enable or disable the queue manager for activity recording.

   Use the MQSC command `ALTER QMGR`, specifying the parameter `ACTIVREC`, to change the value of the queue manager attribute. The value can be:

   MSG
   The queue manager is enabled for activity recording. Any activity reports generated are delivered to the reply-to queue specified in the message descriptor of the message. This is the default value.

   QUEUE
   The queue manager is enabled for activity recording. Any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE. The system queue can also be used to forward activity reports to a common queue.

   DISABLED
   The queue manager is disabled for activity recording. No activity reports are generated while in the scope of this queue manager.

   For example, to enable a queue manager for activity recording and specify that any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE, use the following MQSC command:

   ```
   ALTER QMGR ACTIVREC(QUEUE)
   ```

   Remember: When you modify the *ACTIVREC* queue manager attribute, a running MCA does not detect the change until the channel is restarted.
3. Ensure that your application uses the same algorithm as MCAs use to determine whether to generate an activity report for a message:
   a) Verify that the message has requested activity reports to be generated
   b) Verify that the queue manager where the message currently resides is enabled for activity recording
   c) Put the activity report on the queue determined by the *ACTIVREC* queue manager attribute

Setting up a common queue for activity reports

To determine the locations of the activity reports related to a specific message when the reports are delivered to the local system queue, it is more efficient to use a common queue on a single node

Before you begin

Set the `ACTIVREC` parameter to enable the queue manager for activity recording and to specify that any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE.

About this task

If a number of queue managers in a queue manager network are set to deliver activity reports to the local system queue, it can be time consuming to determine the locations of the activity reports related to a specific message. Alternatively, use a single node, which is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver activity reports to this common queue. The benefit of using a common queue is that queue managers do not have to deliver activity reports to the reply-to queue specified in a message and, when determining the locations of the activity reports related to a message, you query one queue only.

To set up a common queue, perform the following steps:

Procedure

1. Select or define a queue manager as the single node
2. On the single node, select or define a queue for use as the common queue
3. On all queue managers where activity reports are to be delivered to the common queue, redefine the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE as a remote queue definition:
   a) Specify the name of the single node as the remote queue manager name
   b) Specify the name of the common queue as the remote queue name

Determining message route information

To determine a message route, obtain the information from the activity reports collected. Determine whether enough activity reports are on the reply-to queue to enable you to determine the required information and arrange the activity reports in order.

About this task

The order that activity reports are put on the reply-to queue does not necessarily correlate to the order in which the activities were performed. You must order activity reports manually, unless they are generated for a trace-route message, in which case you can use the IBM MQ display route application to order the activity reports.

Determine whether enough activity reports are on the reply-to queue for you to obtain the necessary information:

Procedure

1. Identify all related activity reports on the reply-to queue by comparing identifiers of the activity reports and the original message. Ensure you set the report option of the original message such that the activity reports can be correlated with the original message.
2. Order the identified activity reports from the reply-to queue.

   You can use the following parameters from the activity report:

   OperationType

      The types of operations performed might enable you to determine the activity report that was generated directly before, or after, the current activity report.

For example, an activity report details that an MCA sent a message from a transmission queue down a channel. The last operation detailed in the activity report has an *OperationType* of `send` and details that the message was sent using the channel, CH1, to the destination queue manager, QM1. This means that the next activity performed on the message will have occurred on queue manager, QM1, and that it will have begun with a `receive` operation from channel, CH1. By using this information you can identify the next activity report, providing it exists and has been acquired.

OperationDate* and *OperationTime

You can determine the general order of the activities from the dates and times of the operations in each activity report.

Warning: Unless every queue manager in the queue manager network has their system clocks synchronized, ordering by date and time does not guarantee that the activity reports are in the correct sequence. You must establish the order manually.

The order of the activity reports represents the route, or partial route, that the message took through the queue manager network.

3. Obtain the information you need from the activity information in the ordered activity reports.

   If you have insufficient information about the message, you might be able to acquire further activity reports.

Retrieving further activity reports

To determine a message route, sufficient information must be available from the activity reports collected. If you retrieve the activity reports related to a message from the reply-to queue that the message specified, but you not have the necessary information, look for further activity reports.

About this task

To determine the locations of any further activity reports, perform the following steps:

Procedure

1. For any queue managers in the queue manager network that deliver activity reports to a common queue, retrieve activity reports from the common queue that have a *CorrelId* that matches the *MsgId* of the original message.
2. For any queue managers in the queue manager network that do not deliver activity reports to a common queue, retrieve activity reports as follows:
   a) Examine the existing activity reports to identify queue managers through which the message was routed.
   b) For these queue managers, identify the queue managers that are enabled for activity recording.
   c) For these queue managers, identify any that did not return activity reports to the specified reply-to queue.
   d) For each of the queue managers that you identify, check the system queue SYSTEM.ADMIN.ACTIVITY.QUEUE and retrieve any activity reports that have a *CorrelId* that matches the *MsgId* of the original message.
   e) If you find no activity reports on the system queue, check the queue manager dead letter queue, if one exists.

      An activity report can only be delivered to a dead letter queue if the report option, MQRO_DEAD_LETTER_Q, is set.
3. Arrange all the acquired activity reports in order.

   The order of the activity reports then represents the route, or partial route, that the message took.
4. Obtain the information you need from the activity information in the ordered activity reports.

   In some circumstances, recorded activity information cannot reach the specified reply-to queue, a common queue, or a system queue.

Circumstances where activity information is not acquired

To determine the complete sequence of activities performed on behalf of a message, information related to every activity must be acquired. If the information relating to any activity has not been recorded, or has not been acquired, you can determine only a partial sequence of activities.

Activity information is not recorded in the following circumstances:

- The message is processed by an IBM MQ queue manager earlier than Version 6.0.
- The message is processed by a queue manager that is not enabled for activity recording.
- The application that expected to process the message is not running.

Recorded activity information is unable to reach the specified reply-to queue in the following circumstances:

- There is no channel defined to route activity reports to the reply-to queue.
- The channel to route activity reports to the reply-to queue is not running.
- The remote queue definition to route activity reports back to the queue manager where the reply-to queue resides (the queue manager alias), is not defined.
- The user that generated the original message does not have open, or put, authority to the queue manager alias.
- The user that generated the original message does not have open, or put, authority to the reply-to queue.
- The reply-to queue is put inhibited.

Recorded activity information is unable to reach the system queue, or a common queue, in the following circumstances:

- If a common queue is to be used and there is no channel defined to route activity reports to the common queue.
- If a common queue is to be used and the channel to route activity reports to the common queue is not running.
- If a common queue is to be used and the system queue is incorrectly defined.
- The user that generated the original message does not have open, or put, authority to the system queue.
- The system queue is put inhibited.
- If a common queue is to be used and the user that generated the original message does not have open, or put, authority to the common queue.
- If a common queue is to be used and the common queue is put inhibited.

In these circumstances, providing the activity report does not have the report option MQRO_DISCARD_MSG specified, the activity report can be retrieved from a dead letter queue if one was defined on the queue manager where the activity report was rejected. An activity report will only have this report option specified if the original message, from which the activity report was generated, had both MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG specified in the Report field of the message descriptor.

Trace-route messaging

Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

As the trace-route message is routed through the queue manager network, activity information is recorded. This activity information includes information about the applications that performed the activities, when they were performed, and the operations that were performed as part of the activities. You can use the information recorded using trace-route messaging for the following purposes:

To determine the last known location of a message

If a message does not reach its intended destination, you can use the activity information recorded for a trace-route message to determine the last known location of the message. A trace-route message is sent into a queue manager network with the same target destination as the original message, intending that it follows the same route. Activity information can be accumulated in the message data of the trace-route message, or recorded using activity reports. To increase the probability that the trace-route message follows the same route as the original message, you can modify the trace-route message to mimic the original message.

To determine configuration issues with a queue manager network
Trace-route messages are sent into a queue manager network and activity information is recorded. By studying the activity information recorded for a trace-route message, it can become apparent that the trace-route message did not follow the expected route. There are many reasons why this can occur, for example, a channel might be inactive, forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

You can use the IBM MQ display route application to configure, generate, and put trace-route messages into a queue manager network.

Warning: If you put a trace-route message to a distribution list, the results are undefined.

Related concepts

"Trace-route message reference" on page 132
Use this page to obtain an overview of the trace-route message format. The trace-route message data includes parameters that describe the activities that the trace-route message has caused

How activity information is recorded

With trace-route messaging, you can record activity information in the message data of the trace-route message, or use activity reports. Alternatively, you can use both techniques.

Accumulating activity information in the message data of the trace-route message

As a trace-route message is routed through a queue manager network, information about the activities performed on behalf of the trace-route message can be accumulated in the message data of the trace-route message. The activity information is stored in *Activity* PCF groups. For every activity performed on behalf of the trace-route message, an *Activity* PCF group is written to the end of the PCF block in the message data of the trace-route message.

Additional activity information is recorded in trace-route messaging, in a PCF group called the *TraceRoute* PCF group. The additional activity information is stored in this PCF group, and can be used to help determine the sequence of recorded activities. This technique is controlled by the *Accumulate* parameter in the *TraceRoute* PCF group.

Recording activity information using activity reports

As a trace-route message is routed through a queue manager network, an activity report can be generated for every activity that was performed on behalf of the trace-route message. The activity information is stored in the *Activity* PCF group. For every activity performed on behalf of a trace-route message, an activity report is generated containing an *Activity* PCF group. Activity recording for trace-route messages works in the same way as for any other message.

Activity reports generated for trace-route messages contain additional activity information compared to the those generated for any other message. The additional information is returned in a *TraceRoute* PCF group. The information contained in the *TraceRoute* PCF group is accurate only from the time the activity report was generated. You can use the additional information to help determine the sequence of activities performed on behalf of the trace-route message.

Acquiring recorded activity information

When a trace-route message has reached its intended destination, or is discarded, the method that you use to acquire the activity information depends on how that information was recorded.

Before you begin

If you are unfamiliar with activity information, refer to "How activity information is recorded" on page 76.

About this task

Use the following methods to acquire the activity information after the trace-route message has reached its intended destination, or is discarded:

Procedure

- Retrieve the trace-route message.

  The *Deliver* parameter, in the *TraceRoute* PCF group, controls whether a trace-route message is placed on the target queue on arrival, or whether it is discarded. If the trace-route message is delivered to the target queue, you can retrieve the trace-route message from this queue. Then, you can use the IBM MQ display route application to display the activity information.

  To request that activity information is accumulated in the message data of a trace-route message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_IN_MSG`.
- Use a trace-route reply message.

  When a trace-route message reaches its intended destination, or the trace-route message cannot be routed any further in a queue manager network, a trace-route reply message can be generated. A trace-route reply message contains a duplicate of all the activity information from the trace-route message, and is either delivered to a specified reply-to queue, or the system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE. You can use the IBM MQ display route application to display the activity information.

  To request a trace-route reply message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_AND_REPLY`.
- Use activity reports.

  If activity reports are generated for a trace-route message, you must locate the activity reports before you can acquire the activity information. Then, to determine the sequence of activities, you must order the activity reports.

Controlling trace-route messaging

Enable trace-route messaging at the queue manager level, so that applications in the scope of that queue manager can write activity information to a trace-route message. To enable an entire queue manager network, individually enable every queue manager in the network for trace-route messaging. If you enable more queue managers, more activity reports are generated.

Before you begin

If you are using activity reports to record activity information for a trace-route message, refer to "Controlling activity recording" on page 72.

About this task

To record activity information for a trace-route message as it is routed through a queue manager, perform the following steps:

Procedure

- Define how activity information is to be recorded for the trace-route message.

  Refer to "Generating and configuring a trace-route message" on page 80
- If you want to accumulate activity information in the trace-route message, ensure that the queue manager is enabled for trace-route messaging
- If you want to accumulate activity information in the trace-route message, ensure that applications performing activities on the trace-route message are capable of writing activity information to the message data of the trace-route message

Related concepts

"Generating and configuring a trace-route message" on page 80

A trace-route message comprises specific message descriptor and message data parts. To generate a trace-route message, either create the message manually or use the IBM MQ display route application.

Related tasks

"Controlling activity recording" on page 72
Enable activity recording at the queue manager level. To enable an entire queue manager network, individually enable every queue manager in the network for activity recording. If you enable more queue managers, more activity reports are generated.

Enabling queue managers for trace-route messaging
To control whether queue managers are enabled or disabled for trace-route messaging use the queue manager attribute ROUTEREC.

Use the MQSC command `ALTER QMGR`, specifying the parameter `ROUTEREC` to change the value of the queue manager attribute. The value can be any of the following values:

MSG
The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)
- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

a trace-route reply message is generated, and delivered to the reply-to queue specified in the message descriptor of the trace-route message.

QUEUE
The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)
- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

a trace-route reply message is generated, and delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

DISABLED
The queue manager is disabled for trace-route messaging. Activity information is not accumulated in the the trace-route message, however the *TraceRoute* PCF group can be updated while in the scope of this queue manager.

For example, to disable a queue manager for trace-route messaging, use the following MQSC command:

```
ALTER QMGR ROUTEREC(DISABLED)
```

Remember: When you modify the *ROUTEREC* queue manager attribute, a running MCA does not detect the change until the channel is restarted.

Enabling applications for trace-route messaging
To enable trace-route messaging for a user application, base your algorithm on the algorithm used by message channel agents (MCAs)

Before you begin

If you are not familiar with the format of a trace-route message, see "Trace-route message reference" on page 132.

About this task

Message channel agents (MCAs) are enabled for trace-route messaging. To enable a user application for trace-route messaging, use the following steps from the algorithm that MCAs use:

Procedure

1. Determine whether the message being processed is a trace-route message.

   If the message does not conform to the format of a trace-route message, the message is not processed as a trace-route message.
2. Determine whether activity information is to be recorded.

   If the detail level of the performed activity is not less than the level of detail specified by the *Detail* parameter, activity information is recorded under specific circumstances. This information is only recorded if the trace-route message requests accumulation, and the queue manager is enabled for trace-route messaging, or if the trace-route message requests an activity report and the queue manager is enabled for activity recording.

   - If activity information is to be recorded, increment the *RecordedActivities* parameter.
   - If activity information is not to be recorded, increment the *UnrecordedActivities* parameter.
3. Determine whether the total number of activities performed on the trace-route message exceeds the value of the *MaxActivities* parameter.

   The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*.

   If the total number of activities exceeds *MaxActivities*, reject the message with feedback MQFB_MAX_ACTIVITIES.
4. If value of *Accumulate* is set as MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, and the queue manager is enabled for trace-route messaging, write an Activity PCF group to the end of the PCF block in the message data of a trace-route message.
5. Deliver the trace-route message to a local queue.

   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_NO, reject the trace-route message with feedback MQFB_NOT_DELIVERED.
   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_YES, deliver the trace-route message to the local queue.
6. Generate a trace-route reply message if all the following conditions are true:

   - The trace-route message was delivered to a local queue or rejected
   - The value of the parameter, *Accumulate*, is MQROUTE_ACCUMULATE_AND_REPLY
   - The queue manager is enabled for trace-route messaging

   The trace-route reply message is put on the queue determined by the ROUTEREC queue manager attribute.
7. If the trace-route message requested an activity report and the queue manager is enabled for activity recording, generate an activity report.

   The activity report is put on the queue determined by the ACTIVREC queue manager attribute.

Generating and configuring a trace-route message

A trace-route message comprises specific message descriptor and message data parts. To generate a trace-route message, either create the message manually or use the IBM MQ display route application.

A trace-route message consists of the following parts:

Message descriptor
: An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
: One of the following combinations:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF

The trace-route message data consists of the *TraceRoute* PCF group and one or more *Activity* PCF groups.

Manual generation

When generating a trace-route message manually, an *Activity* PCF group is not required. *Activity* PCF groups are written to the message data of the trace-route message when an MCA or user-written application performs an activity on its behalf.

The IBM MQ display route application

Use the IBM MQ display route application, `dspmqrte` , to configure, generate and put a trace-route message into a queue manager network. Set the *Format* parameter in the message descriptor to MQFMT_ADMIN. You cannot add user data to the trace-route message generated by the IBM MQ display route application.

Restriction: `dspmqrte` cannot be issued on queue managers before IBM WebSphere MQ 6.0 or on IBM MQ for z/OS queue managers. If you want the first queue manager the trace-route message is routed through to be a queue manager of this type, connect to the queue manager as a IBM WebSphere MQ 6.0 or later client using the optional parameter `-c`.

Mimicking the original message

When using a trace-route message to determine the route another message has taken through a queue manager network, the more closely a trace-route message mimics the original message, the greater the chance that the trace-route message will follow the same route as the original message.

The following message characteristics can affect where a message is forwarded to within a queue manager network:

Priority
: The priority can be specified in the message descriptor of the message.

Persistence
: The persistence can be specified in the message descriptor of the message.

Expiration
: The expiration can be specified in the message descriptor of the message.

Report options
: Report options can be specified in the message descriptor of the message.

Message size
: To mimic the size of a message, additional data can be written to the message data of the message. For this purpose, additional message data can be meaningless.

Tip: The IBM MQ display route application cannot specify message size.

Message data
: Some queue manager networks use content based routing to determine where messages are forwarded. In these cases the message data of the trace-route message needs to be written to mimic the message data of the original message.

Tip: The IBM MQ display route application cannot specify message data.

The TraceRoute PCF group
Attributes in the *TraceRoute* PCF group control the behavior of a trace-route message. The *TraceRoute* PCF group is in the message data of every trace-route message.

The following table lists the parameters in the *TraceRoute* group that an MCA recognizes. Further parameters can be added if user-written applications are written to recognize them, as described in "Additional activity information" on page 85.

Table 12. TraceRoute PCF group

Parameter	Type
TraceRoute	MQCFGR
Detail	MQCFIN
RecordedActivities	MQCFIN
UnrecordedActivities	MQCFIN
DiscontinuityCount	MQCFIN
MaxActivities	MQCFIN
Accumulate	MQCFIN
Forward	MQCFIN
Deliver	MQCFIN

Descriptions of each parameter in the *TraceRoute* PCF group follows:

Detail
Specifies the detail level of activity information that is to be recorded. The value can be any of the following values:

MQROUTE_DETAIL_LOW
Only activities performed by user application are recorded.

MQROUTE_DETAIL_MEDIUM
Activities specified in MQROUTE_DETAIL_LOW should be recorded. Additionally, activities performed by MCAs are recorded.

MQROUTE_DETAIL_HIGH
Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM should be recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user applications that are to record further activity information. For example, if a user application determines the route a message takes by considering certain message characteristics, the information about the routing logic could be included with this level of detail.

RecordedActivities
Specifies the number of recorded activities performed on behalf of the trace-route message. An activity is considered to be recorded if information about it has been written to the trace-route message, or if an activity report has been generated. For every recorded activity, *RecordedActivities* increments by one.

UnrecordedActivities
Specifies the number of unrecorded activities performed on behalf of the trace-route message. An activity is considered to be unrecorded if an application that is enabled for trace-route messaging neither accumulates, nor writes the related activity information to an activity report.

An activity performed on behalf of a trace-route message is unrecorded in the following circumstances:

- The detail level of the performed activity is less than the level of detail specified by the parameter *Detail*.
- The trace-route message requests an activity report but not accumulation, and the queue manager is not enabled for activity recording.

- The trace-route message requests accumulation but not an activity report, and the queue manager is not enabled for trace-route messaging.
- The trace-route message requests both accumulation and an activity report, and the queue manager is not enabled for activity recording and trace route messaging.
- The trace-route message requests neither accumulation nor an activity report.

For every unrecorded activity the parameter, *UnrecordedActivities*, increments by one.

DiscontinuityCount

Specifies the number of times the trace-route message has been routed through a queue manager with applications that were not enabled for trace-route messaging. This value is incremented by the queue manager. If this value is greater than 0, only a partial message route can be determined.

MaxActivities

Specifies the maximum number of activities that can be performed on behalf of the trace-route message.

The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*. The total number of activities must not exceed the value of *MaxActivities*.

The value of *MaxActivities* can be:

A positive integer

The maximum number of activities.

If the maximum number of activities is exceeded, the trace-route message is rejected with feedback MQFB_MAX_ACTIVITIES. This can prevent the trace-route message from being forwarded indefinitely if caught in an infinite loop.

MQROUTE_UNLIMITED_ACTIVITIES

An unlimited number of activities can be performed on behalf of the trace-route message.

Accumulate

Specifies the method used to accumulate activity information. The value can be any of the following values:

MQROUTE_ACCUMULATE_IN_MSG

If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_AND_REPLY

If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message, and a trace-route reply message is generated if any of the following occur:

- The trace-route message is discarded by an IBM MQ Version 6 (or later) queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by an IBM MQ Version 6 (or later) queue manager.
- The number of activities performed on the trace-route message exceeds the value of *MaxActivities*.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_NONE

Activity information is not accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.

Forward
Specifies where a trace-route message can be forwarded to. The value can be:

MQROUTE_FORWARD_IF_SUPPORTED
The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Queue managers use the following algorithm when determining whether to forward a trace-route message to a remote queue manager:

1. Determine whether the remote queue manager is capable of supporting trace-route messaging.
   - If the remote queue manager is capable of supporting trace-route messaging, the algorithm continues to step "4" on page 83.
   - If the remote queue manager is not capable of supporting trace-route messaging, the algorithm continues to step "2" on page 83
2. Determine whether the *Deliver* parameter from the *TraceRoute* group contains any unrecognized delivery options in the MQROUTE_DELIVER_REJ_UNSUP_MASK bit mask.
   - If any unrecognized delivery options are found, the trace-route message is rejected with feedback MQFB_UNSUPPORTED_DELIVERY.
   - If no unrecognized delivery options are found, the algorithm continues to step "3" on page 83.
3. Determine the value of the parameter *Deliver* from the *TraceRoute* PCF group in the trace-route message.
   - If *Deliver* is specified as MQROUTE_DELIVER_YES, the trace-route message is forwarded to the remote queue manager.
   - If *Deliver* is specified as MQROUTE_DELIVER_NO, the algorithm continues to step "4" on page 83.
4. Determine whether the *Forward* parameter from the *TraceRoute* group contains any unrecognized forwarding options in the MQROUTE_FORWARDING_REJ_UNSUP_MASK bit mask.
   - If any unrecognized forwarding options are found, the trace-route message is rejected with feedback MQFB_UNSUPPORTED_FORWARDING.
   - If no unrecognized forwarding options are found, the algorithm continues to step "5" on page 83.
5. Determine the value of the parameter *Forward* from the *TraceRoute* PCF group in the trace-route message.
   - If *Forward* is specified as MQROUTE_FORWARD_IF_SUPPORTED, the trace-route message is rejected with feedback MQFB_NOT_FORWARDED.
   - If *Forward* is specified as MQROUTE_FORWARD_ALL, trace-route message can be forwarded to the remote queue manager.

Deliver
Specifies the action to be taken if the trace-route message reaches its intended destination. User-written applications must check this attribute before placing a trace-route message on its target queue. The value can be any of the following values:

MQROUTE_DELIVER_YES
On arrival, the trace-route message is put on the target queue. Any application performing a get operation on the target queue can retrieve the trace-route message.

MQROUTE_DELIVER_NO
On arrival, the trace-route message is not delivered to the target queue. The message is processed according to its report options.

Setting up a common queue for trace-route reply messages

To determine the locations of the trace-route reply messages related to a specific message when the reports are delivered to the local system queue, it is more efficient to use a common queue on a single node

Before you begin

Set the ROUTEREC parameter to enable the queue manager for trace-route messaging and to specify that any trace-route reply messages generated are delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

About this task

If a number of queue managers in a queue manager network are set to deliver trace-route reply messages to the local system queue, it can be time consuming to determine the locations of the trace-route reply messages related to a specific message. Alternatively, use a single node, which is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver trace-route reply messages to this common queue. The benefit of using a common queue is that queue managers do not have to deliver trace-route reply messages to the reply-to queue specified in a message and, when determining the locations of the trace-route reply messages related to a message, you query one queue only.

To set up a common queue, perform the following steps:

Procedure

1. Select or define a queue manager as the single node
2. On the single node, select or define a queue for use as the common queue
3. On all queue managers that forward trace-route reply messages to the common queue, redefine the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE as a remote queue definition
   a) Specify the name of the single node as the remote queue manager name
   b) Specify the name of the common queue as the remote queue name

Acquiring and using recorded information

Use any of the following techniques to acquire recorded activity information for a trace-route message

Note that the circumstances in which activity information is not acquired apply also to trace-route reply messages.

Activity information is not recorded when a trace-route message is processed by a queue manager that is disabled for both activity recording and trace-route messaging.

Acquiring information from trace-route reply messages

To acquire activity information you locate the trace-route reply message. Then you retrieve the message and analyze the activity information.

About this task

You can acquire activity information from a trace-route reply message only if you know the location of the trace-route reply message. Locate the message and process the activity information as follows:

Procedure

1. Check the reply-to queue that was specified in the message descriptor of the trace-route message. If the trace-route reply message is not on the reply-to queue, check the following locations:
   - The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on the target queue manager of the trace-route message
   - The common queue, if you have set up a common queue for trace-route reply messages

- The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on any other queue manager in the queue manager network, which can occur if the trace-route message has been put to a dead-letter queue, or the maximum number of activities was exceeded

2. Retrieve the trace-route reply message
3. Use the IBM MQ display route application to display the recorded activity information
4. Study the activity information and obtain the information that you need

Acquiring information from trace-route messages
To acquire activity information you locate the trace-route message, which must have the appropriate parameters in the *TraceRoute* PCF group. Then you retrieve the message and analyze the activity information.

About this task

You can acquire activity information from a trace-route message only if you know the location of the trace-route message and it has the parameter *Accumulate* in the *TraceRoute* PCF group specified as either `MQROUTE_ACCUMULATE_IN_MSG` or `MQROUTE_ACCUMULATE_AND_REPLY`.

For the trace-route message to be delivered to the target queue the *Deliver* parameter in the *TraceRoute* PCF group must be specified as `MQROUTE_DELIVER_YES`.

Procedure

1. Check the target queue. If the trace-route message is not on the target queue, you can try to locate the trace-route message using a trace-route message enabled for activity recording. With the generated activity reports try to determine the last known location of the trace-route message.
2. Retrieve the trace-route message
3. Use the IBM MQ display route application to display the recorded activity information
4. Study the activity information and obtain the information that you need

Acquiring information from activity reports
To acquire activity information you locate the activity report, which must have the report option specified in the message descriptor. Then you retrieve the activity report and analyze the activity information.

About this task

You can acquire activity information from an activity report only if you know the location of the activity report and the report option MQRO_ACTIVITY was specified in the message descriptor of the trace-route message.

Procedure

1. Locate and order the activity reports generated for a trace-route message.

   When you have located the activity reports, you can order them manually or use the IBM MQ display route application to order and display the activity information automatically.
2. Study the activity information and obtain the information that you need

Additional activity information

As a trace-route message is routed through a queue manager network, user applications can record additional information by including one or more additional PCF parameters when writing the *Activity* group to the message data of the trace-route message or activity report.

Additional activity information can help system administrators to identify the route taken by a trace-route message took, or why that route was taken.

If you use the IBM MQ display route application to display the recorded information for a trace-route message, any additional PCF parameters can only be displayed with a numeric identifier, unless the parameter identifier of each parameter is recognized by the IBM MQ display route application. To recognize a parameter identifier, additional information must be recorded using the following PCF parameters. Include these PCF parameters in an appropriate place in the *Activity* PCF group.

GroupName

Table 13. Group name	
Description	**Grouped parameters specifying the additional information.**
Identifier	MQGACF_VALUE_NAMING.
Data type	MQCFGR
Parameters in group	*ParameterName* *ParameterValue*

ParameterName

Table 14. Parameter name	
Description	**Contains the name to be displayed by the IBM MQ display route application, which puts the value of *ParameterValue* into context.**
Identifier	MQCA_VALUE_NAME.
Data type	MQCFST
Included in PCF group:	*GroupName*.
Value:	The name to be displayed.

ParameterValue

Table 15. Parameter value	
Description	**Contains the value to be displayed by the IBM MQ display route application.**
Identifier:	The PCF structure identifier for the additional information.
Data type:	The PCF structure data type for the additional information.
Included in PCF group:	*GroupName*.
Value:	The value to be displayed.

Examples of recording additional activity information

The following examples illustrate how a user application can record additional information when performing an activity on behalf of a trace-route message. In both examples, the IBM MQ display route application is used to generate a trace-route message, and display the activity information returned to it.

Example 1

Additional activity information is recorded by a user application in a format where the parameter identifier *is not* recognized by the IBM MQ display route application.

1. The IBM MQ display route application is used to generate and put a trace-route message into a queue manager network. The necessary options are set to request the following:
   - Activity information is accumulated in the message data of the trace-route message.
   - On arrival at the target queue the trace-route message is discarded, and a trace-route reply message is generated and delivered to a specified reply-to queue.
   - On receipt of the trace-route reply message, the IBM MQ display route application displays the accumulated activity information.

The trace-route message is put into the queue manager network.

2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameter to the end of the Activity group:

   ColorValue

   Identifier
   65536

   Data type
   MQCFST

   Value
   'Red'

   This additional PCF parameter gives further information about the activity that was performed, however it is written in a format where the parameter identifier *is not* recognized by the IBM MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the IBM MQ display route application. The additional activity information is displayed as follows:

```
65536: 'Red'
```

   The IBM MQ display route application does not recognize the parameter identifier of the PCF parameter and displays it as a numeric value. The context of the additional information is not clear.

   For an example of when the IBM MQ display route application does recognize the parameter identifier of the PCF parameter, see “Example 2” on page 87.

Example 2
Additional activity information is recorded by a user application in a format where the parameter identifier *is* recognized by the IBM MQ display route application.

1. The IBM MQ display route application is used to generate and put a trace-route message into a queue manager network in the same fashion as in “Example 1” on page 86.
2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameters to the end of the Activity group:

   ColorInfo

Table 16. Color information

Description	Grouped parameters specifying information about a color.
Identifier:	MQGACF_VALUE_NAMING.
Data type:	MQCFGR.
Parameters in group:	*ColorName* *ColorValue*

   ColorName

Table 17. Color name

Description	Contains the name to be displayed by the IBM MQ display route application which puts the value of *ColorValue* into context.
Identifier:	MQCA_VALUE_NAME.

Table 17. Color name (continued)

Description	Contains the name to be displayed by the IBM MQ display route application which puts the value of *ColorValue* into context.
Data type:	MQCFST.
Included in PCF group:	*ColorInfo*.
Value:	'Color'

ColorValue

Table 18. Color value

Description	Contains the value to be displayed by the IBM MQ display route application.
Identifier:	65536.
Data type:	MQCFST.
Included in PCF group:	*ColorInfo* .
Value:	'Red'

These additional PCF parameters gives further information about the activity that was performed. These PCF parameters are written in a format where the parameter identifier *is* recognized by the IBM MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the IBM MQ display route application. The additional activity information is displayed as follows:

```
Color: 'Red'
```

The IBM MQ display route application recognizes that the parameter identifier of the PCF structure containing the value of the additional activity information has a corresponding name. The corresponding name is displayed instead of the numeric value.

IBM MQ display route application

Use the IBM MQ display route application (**dspmqrte**) to work with trace-route messages and activity information related to a trace-route message, using a command-line interface. z/OS The IBM MQ display route application is not available for IBM MQ for z/OS queue managers.

You can use the IBM MQ display route application for the following purposes:

- To configure, generate, and put a trace-route message into a queue manager network.

  By putting a trace-route message into a queue manager network, activity information can be collected and used to determine the route that the trace-route message took. You can specify the characteristics of the trace-route messages as follows:

  - The destination of the trace-route message.
  - How the trace-route message mimics another message.
  - How the trace-route message should be handled as it is routed through a queue manager network.
  - Whether activity recording or trace-route messaging are used to record activity information.
- To order and display activity information related to a trace-route message.

If the IBM MQ display route application has put a trace-route message into a queue manager network, after the related activity information has been returned, the information can be ordered and displayed immediately. Alternatively, the IBM MQ display route application can be used to order, and display, activity information related to a trace-route message that was previously generated.

Related information

dspmqrte

Parameters for trace-route messages

Use this page to obtain an overview of the parameters provided by the IBM MQ display route application, **`dspmqrte`**, to determine the characteristics of a trace-route message, including how it is treated as it is routed through a queue manager network.

Related information

dspmqrte

Queue manager connection

Use this page to specify the queue manager that the IBM MQ display route application connects to

-c

Specifies that the IBM MQ display route application connects as a client application.

If you do not specify this parameter, the IBM MQ display route application does not connect as a client application.

-m *`QMgrName`*

The name of the queue manager to which the IBM MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

The target destination

Use this page to specify the target destination of a trace-route message

-q *`TargetQName`*

If the IBM MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

-ts *`TargetTopicString`*

Specifies the topic string.

-qm *`TargetQMgr`*

Qualifies the target destination; normal queue manager name resolution will then apply. The target destination is specified with *`-q TargetQName`* or *`-ts TargetTopicString`* .

If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the target queue manager.

-o

Specifies that the target destination is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target destination is opened with option MQOO_BIND_NOT_FIXED.

If you do not specify this parameter, the target destination is bound to a specific destination.

The publication topic

For publish/subscribe applications, use this page to specify the topic string of a trace-route message for the IBM MQ display route application to publish

-ts *`TopicName`*

Specifies a topic string to which the IBM MQ display route application is to publish a trace-route message, and puts this application into topic mode. In this mode, the application traces all of the messages that result from the publish request.

You can also use the IBM MQ display route application to display the results from an activity report that was generated for publish messages.

Message mimicking
Use this page to configure a trace-route message to mimic a message, for example when the original message did not reach its intended destination

One use of trace-route messaging is to help determine the last known location of a message that did not reach its intended destination. The IBM MQ display route application provides parameters that can help configure a trace-route message to mimic the original message. When mimicking a message, you can use the following parameters:

-l *Persistence*
Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

yes
The generated trace-route message is persistent. (MQPER_PERSISTENT).

no
The generated trace-route message is **not** persistent. (MQPER_NOT_PERSISTENT).

q
The generated trace-route message inherits its persistence value from the destination specified by *-q TargetQName* or *-ts TargetTopicString*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned will share the same persistence value as the original trace-route message.

If *Persistence* is specified as **yes**, you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is **not** persistent.

-p *Priority*
Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF. MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

If you do not specify this parameter, the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

-xs *Expiry*
Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-ro none | *ReportOption*

none
Specifies no report options are set.

ReportOption
Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are:

activity
The report option MQRO_ACTIVITY is set.

coa
The report option MQRO_COA_WITH_FULL_DATA is set.

cod
The report option MQRO_COD_WITH_FULL_DATA is set.

exception
The report option MQRO_EXCEPTION_WITH_FULL_DATA is set.

expiration
The report option MQRO_EXPIRATION_WITH_FULL_DATA is set.

discard
The report option MQRO_DISCARD_MSG is set.

If neither `-ro ReportOption` nor `-ro none` are specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

The IBM MQ display route application does not allow you to add user data to the trace-route message. If you require user data to be added to the trace-route message you must generate the trace-route message manually.

Recorded activity information
Use this page to specify the method used to return recorded activity information, which you can then use to determine the route that a trace-route message has taken

Recorded activity information can be returned as follows:

- In activity reports
- In a trace-route reply message
- In the trace-route message itself (having been put on the target queue)

When using **dspmqrte**, the method used to return recorded activity information is determined using the following parameters:

-ro *activity*
Specifies that activity information is returned using activity reports. By default activity recording is enabled.

-ac -ar
Specifies that activity information is accumulated in the trace-route message, and that a trace-route reply message is to be generated.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

-ar
Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:

- The trace-route message is discarded by an IBM MQ queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by an IBM MQ queue manager.
- The number of activities performed on the trace-route message exceeds the value of specified in *-s Activities*.

-ac -d yes
Specifies that activity information is accumulated in the trace-route message, and that on arrival, the trace-route message will be put on the target queue.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

-d yes
On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

The trace-route message can then be retrieved from the target queue, and the recorded activity information acquired.

You can combine these methods as required.

Additionally, the detail level of the recorded activity information can be specified using the following parameter:

-t *Detail*
Specifies the activities that are recorded. The possible values for *Detail* are:

low
Activities performed by user-defined application are recorded only.

medium

Activities specified in **low** are recorded. Additionally, publish activities and activities performed by MCAs are recorded.

high

Activities specified in **low**, and **medium** are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic could be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

By default the IBM MQ display route application uses a temporary dynamic queue to store the returned messages. When the IBM MQ display route application ends, the temporary dynamic queue is closed, and any messages are purged. If the returned messages are required beyond the current execution of the IBM MQ display route application ends, then a permanent queue must be specified using the following parameters:

-rq *ReplyToQ*
Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the *-n* parameter is specified, a reply-to queue must be specified that is **not** a temporary dynamic queue.

If you do not specify this parameter then a dynamic reply-to queue is created using the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE.

-rqm *ReplyToQMgr*
Specifies the name of the queue manager where the reply-to queue resides. The name can contain up to 48 characters.

If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the reply-to queue manager.

How the trace-route message is handled
Use this page to control how a trace-route message is handled as it is routed through a queue manager network.

The following parameters can restrict where the trace-route message can be routed in the queue manager network:

-d *Deliver*
Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

yes	On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.
no	On arrival, the trace-route message is **not** put to the target queue.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

-f *Forward*

Specifies the type of queue manager that the trace-route message can be forwarded to. For details of the algorithm that queue managers use to determine whether to forward a message to a remote queue manager, refer to “The TraceRoute PCF group” on page 81. The possible values for *Forward* are:

all

The trace-route message is forwarded to any queue manager.

Warning: If forwarded to an IBM MQ queue manager earlier than Version 6.0, the trace-route message will not be recognized and can be delivered to a local queue despite the value of the *-d Deliver* parameter.

supported

The trace-route message is only forwarded to a queue manager that will honor the *Deliver* parameter from the *TraceRoute* PCF group

If you do not specify this parameter, the trace-route message will only be forwarded to a queue manager that will honor the *Deliver* parameter.

The following parameters can prevent a trace-route message from remaining in a queue manager network indefinitely:

-s *Activities*

Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-xs *Expiry*

Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-xp *PassExpiry*

Specifies whether the expiry time from the trace-route message is passed on to a trace-route reply message. Possible values for *PassExpiry* are:

yes

The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message.

If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD report option (if specified), and the remaining expiry time are passed on.

This is the default value.

no

The report option MQRO_PASS_DISCARD_AND_EXPIRY is not specified.

If a trace-route reply message is generated for the trace-route message, the discard option and expiry time from the trace-route message are **not** passed on.

If you do not specify this parameter, MQRO_PASS_DISCARD_AND_EXPIRY is not specified.

-ro *discard*

Specifies the MQRO_DISCARD_MSG report option. This can prevent the trace-route message remaining in the queue manager network indefinitely.

Display of activity information

The IBM MQ display route application can display activity information for a trace-route message that it has just put into a queue manager network, or it can display activity information for a previously generated trace-route message. It can also display additional information recorded by user-written applications.

To specify whether activity information returned for a trace-route message is displayed, specify the following parameter:

-n
: Specifies that activity information returned for the trace-route message is not to be displayed.

  If this parameter is accompanied by a request for a trace-route reply message, (`-ar`), or any of the report generating options from (`-ro ReportOption`), then a specific (non-model) reply-to queue must be specified using `-rq ReplyToQ` . By default, only activity report messages are requested.

  After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is displayed containing the message identifier of the trace-route message. The message identifier can be used by the IBM MQ display route application to display the activity information for the trace-route message at a later time, using the *-i CorrelId* parameter.

  If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the *-v* parameter.

When displaying activity information for a trace-route message that has just been put into a queue manager network, the following parameter can be specified:

-w *WaitTime*
: Specifies the time, in seconds, that the IBM MQ display route application will wait for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

  If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

When displaying previously accumulated activity information the following parameters must be set:

-q *TargetQName*
: If the IBM MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

-i *CorrelId*
: This parameter is used when the IBM MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply messages on the queue specified by *-q TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

  The format of *CorrelId* is a 48 character hexadecimal string.

The following parameters can be used when displaying previously accumulated activity information, or when displaying current activity information for a trace-route message:

-b
: Specifies that the IBM MQ display route application will only browse activity reports or a trace-route reply message related to a message. This allows activity information to be displayed again at a later time.

  If you do not specify this parameter, the IBM MQ display route application will destructively get activity reports or a trace-route reply message related to a message.

-v summary | all | none | outline *DisplayOption*

  summary
  : The queues that the trace-route message was routed through are displayed.

  all
  : All available information is displayed.

  none
  : No information is displayed.

outline *DisplayOption*

Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator.

If no values are supplied the following is displayed:

- The application name
- The type of each operation
- Any operation specific parameters

Possible values for *DisplayOption* are:

activity

All non-PCF group parameters in *Activity* PCF groups are displayed.

identifiers

Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*.

message

All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*.

msgdelta

All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*.

operation

All non-PCF group parameters in *Operation* PCF groups are displayed.

traceroute

All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

Display of additional information

As a trace-route message is routed through a queue manager network, user-written applications can record additional information by writing one or more additional PCF parameters to the message data of the trace-route message or to the message data of an activity report. For the IBM MQ display route application to display additional information in a readable form it must be recorded in a specific format, as described in “Additional activity information” on page 85.

IBM MQ display route application examples

The following examples show how you can use the IBM MQ display route application. In each example, two queue managers (QM1 and QM2) are inter-connected by two channels (QM2.TO.QM1 and QM1.TO.QM2).

Example 1 - Requesting activity reports

Display activity information from a trace-route message delivered to the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary report option is specified so that activity reports are requested as the trace-route reply message is routed. On arrival at the target queue the trace-route message is discarded. Activity information returned to the IBM MQ display route application using activity reports is put in order and displayed.

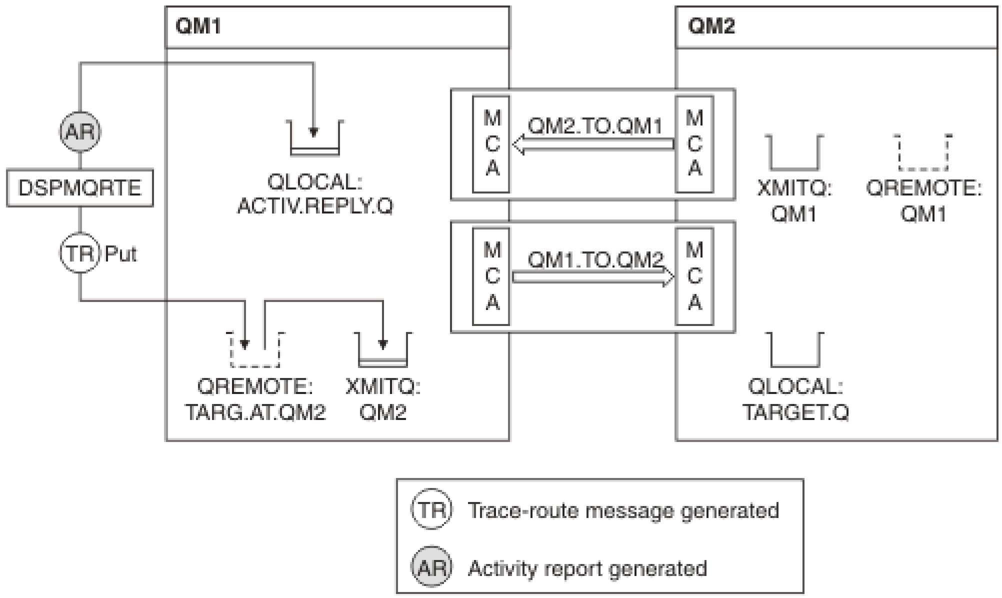


Figure 15. Requesting activity reports, Diagram 1

- The ACTIVREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

  ```
  dspmqrte -m QM1 -q TARG.AT.QM2 -rq ACTIV.REPLY.Q
  ```

  QM1 is the name of the queue manager to which the IBM MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent.

  Default values are assumed for all options that are not specified, but note in particular the -f option (the trace-route message is forwarded only to a queue manager that honors the Deliver parameter of the TraceRoute PCF group), the -d option (on arrival, the trace-route message is not put on the target queue), the -ro option (MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified), and the -t option (medium detail level activity is recorded).
- DSPMQRTE generates the trace-route message and puts it on the remote queue TARG.AT.QM2.
- DSPMQRTE then looks at the value of the ACTIVREC attribute of queue manager QM1. The value is MSG, therefore DSPMQRTE generates an activity report and puts it on the reply queue ACTIV.REPLY.Q.

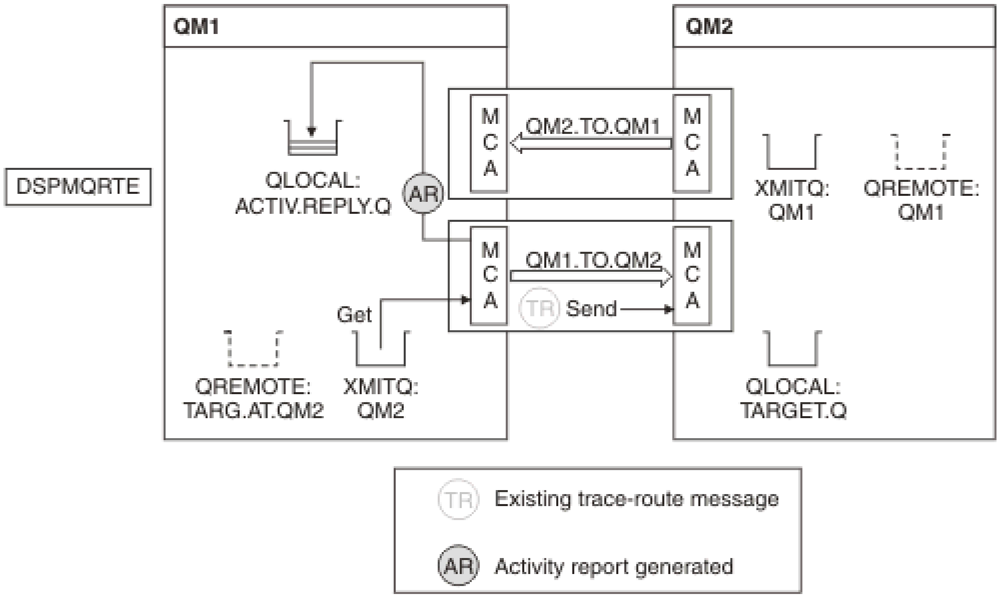


Figure 16. Requesting activity reports, Diagram 2

- The sending message channel agent (MCA) gets the trace-route message from the transmission queue. The message is a trace-route message, therefore the MCA begins to record the activity information.
- The ACTIVREC attribute of the queue manager (QM1) is MSG, and the MQRO_ACTIVITY option is specified in the Report field of the message descriptor, therefore the MCA will later generate an activity report. The RecordedActivities parameter value in the TraceRoute PCF group is incremented by 1.
- The MCA checks that the MaxActivities value in the TraceRoute PCF group has not been exceeded.
- Before the message is forwarded to QM2 the MCA follows the algorithm that is described in Forwarding (steps “1” on page 83, “4” on page 83, and “5” on page 83) and the MCA chooses to send the message.
- The MCA then generates an activity report and puts it on the reply queue (ACTIV.REPLY.Q).

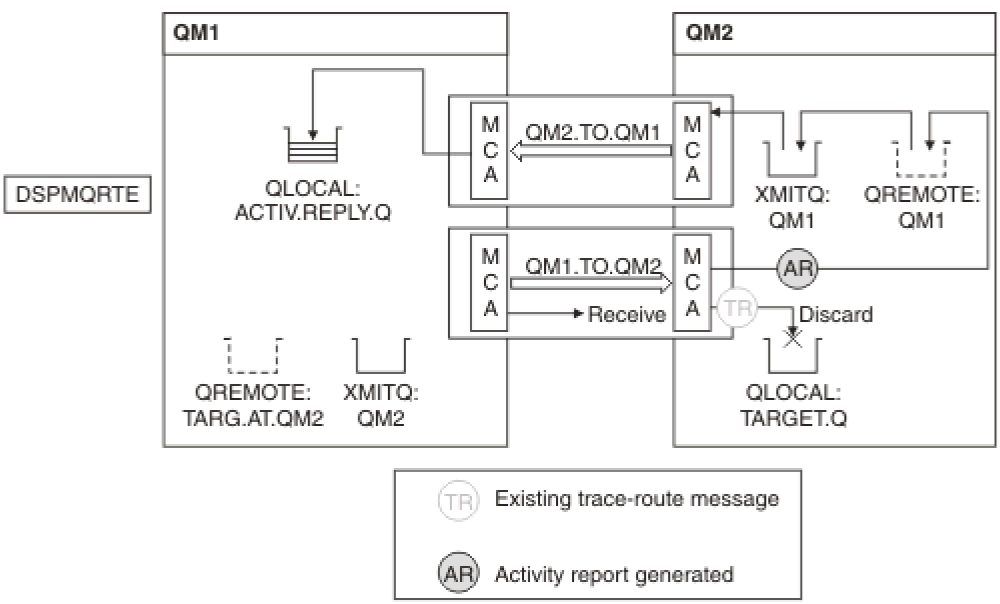


Figure 17. Requesting activity reports, Diagram 3

- The receiving MCA receives the trace-route message from the channel. The message is a trace-route message, therefore the MCA begins to record the information about the activity.

- If the queue manager that the trace-route message has come from is Version 5.3.1 or earlier, the MCA increments the DiscontinuityCount parameter of the TraceRoute PCF by 1. This is not the case here.
- The ACTIVREC attribute of the queue manager (QM2) is MSG, and the MQRO_ACTIVITY option is specified, therefore the MCA will generate an activity report. The RecordedActivities parameter value is incremented by 1.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.
- The MCA then generates the final activity report and puts it on the reply queue. This resolves to the transmission queue that is associated with queue manager QM1 and the activity report is returned to queue manager QM1 (ACTIV.REPLY.Q).

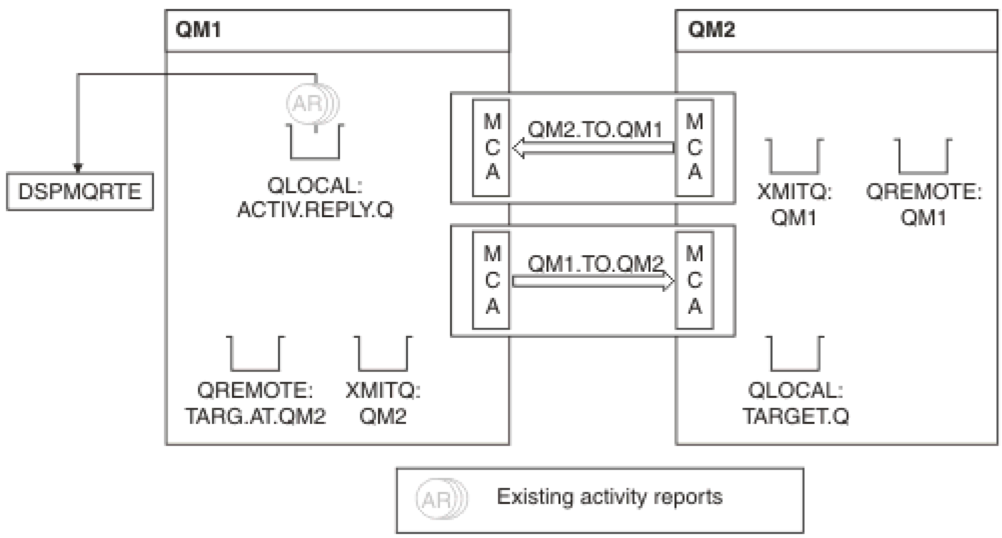


Figure 18. Requesting activity reports, Diagram 4

- Meanwhile, DSPMQRTE has been continually performing MQGETs on the reply queue (ACTIV.REPLY.Q), waiting for activity reports. It will wait for up to 120 seconds (60 seconds longer than the expiry time of the trace-route message) since -w was not specified when DSPMQRTE was started.
- DSPMQRTE gets the 3 activity reports off the reply queue.
- The activity reports are ordered using the RecordedActivities, UnrecordedActivities, and DiscontinuityCount parameters in the TraceRoute PCF group for each of the activities. The only value that is non-zero in this example is RecordedActivities, therefore this is the only parameter that is actually used.
- The program ends as soon as the discard operation is displayed. Even though the final operation was a discard, it is treated as though a put took place because the feedback is MQFB_NOT_DELIVERED.

  The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
 -rq ACTIV.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
 queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 2 - Requesting a trace-route reply message
Generate and deliver a trace-route message to the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary option is specified so that activity information is accumulated in the trace-route message. On arrival at the target queue a trace-route reply message is requested, and the trace-route message is discarded.

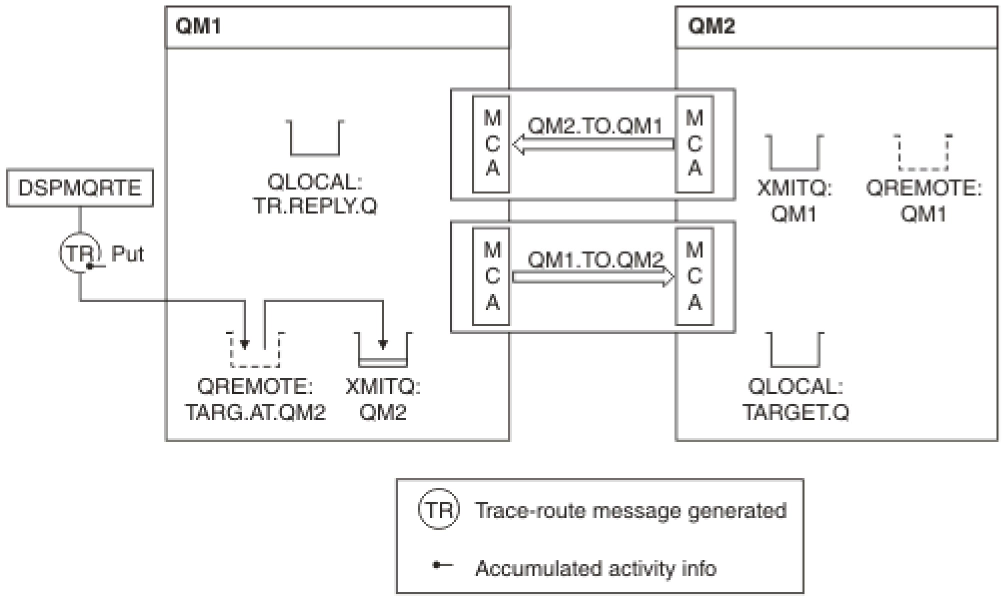


Figure 19. Requesting a trace-route reply message, Diagram 1

- The ROUTEREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

```
dspmqrte -m QM1 -q TARG.AT.QM2 -rq TR.REPLY.Q -ac -ar -ro discard
```

  QM1 is the name of the queue manager to which the IBM MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent. The -ac option specifies that activity information is accumulated in the trace-route message, the -ar option specifies that all accumulated activity is sent to the reply-to queue that is specified by the -rq option (that is, TR.REPLY.Q). The -ro option specifies that report option MQRO_DISCARD_MSG is set which means that activity reports are not generated in this example.
- DSPMQRTE accumulates activity information in the trace-route message before the message is put on the target route. The queue manager attribute ROUTEREC must not be DISABLED for this to happen.

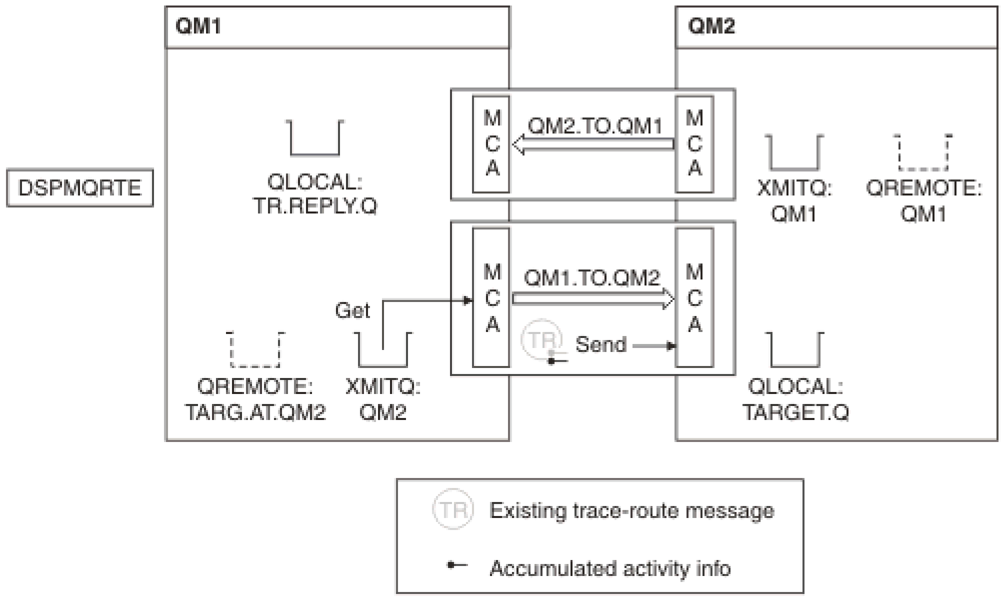


Figure 20. Requesting a trace-route reply message, Diagram 2

- The message is a trace-route message, therefore the sending MCA begins to record information about the activity.
- The queue manager attribute ROUTEREC on QM1 is not DISABLED, therefore the MCA accumulates the activity information within the message, before the message is forwarded to queue manager QM2.

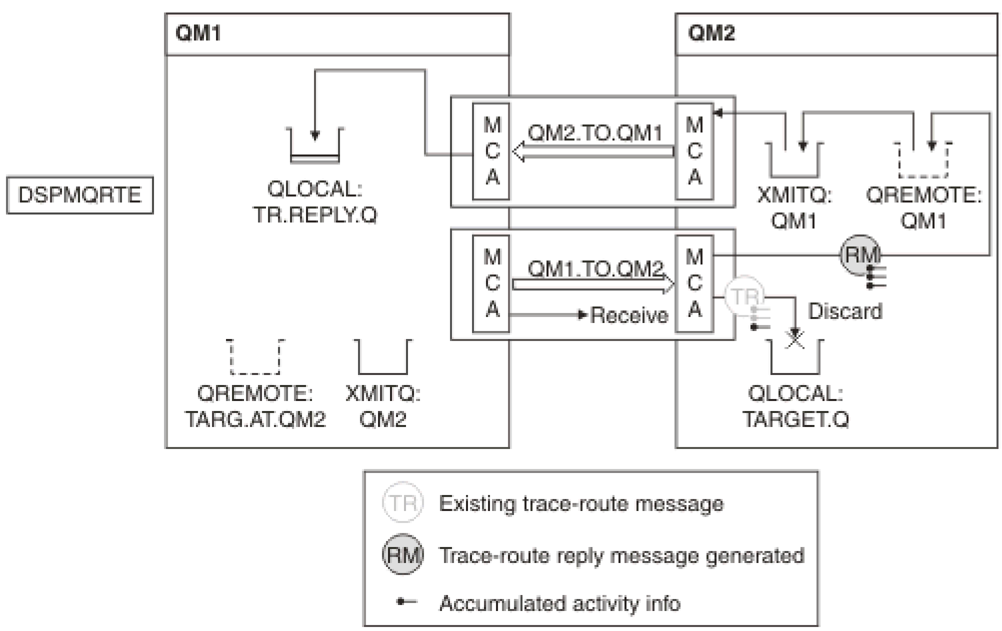


Figure 21. Requesting a trace-route reply message, Diagram 3

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.
- The queue manager attribute ROUTEREC on QM2 is not DISABLED, therefore the MCA accumulates the information within the message.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.

- This is the last activity that will take place on the message, and because the queue manager attribute ROUTEREC on QM1 is not DISABLED, the MCA generates a trace-route reply message in accordance with the Accumulate value. The value of ROUTEREC is MSG, therefore the reply message is put on the reply queue. The reply message contains all the accumulated activity information from the trace-route message.

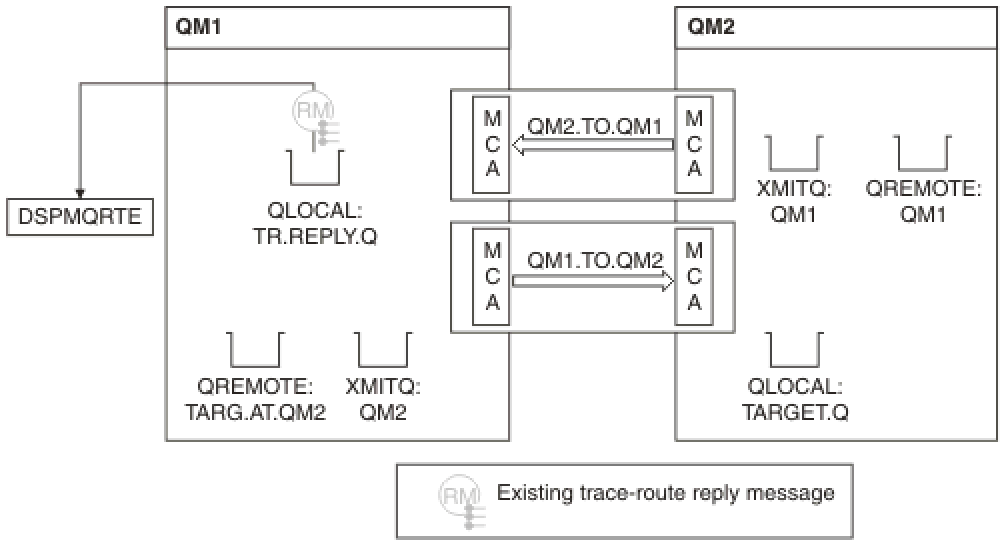


Figure 22. Requesting a trace-route reply message, Diagram 4

- Meanwhile DSPMQRTE is waiting for the trace-route reply message to return to the reply queue. When it returns, DSPMQRTE parses each activity that it contains and prints it out. The final operation is a discard operation. DSPMQRTE ends after it has been printed.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
 TR.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
 manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 3 - Delivering activity reports to the system queue
Detect when activity reports are delivered to queues other than the reply-to queue and use the IBM MQ display route application to read activity reports from the other queue.

This example is the same as “Example 1 - Requesting activity reports” on page 95, except that QM2 now has the value of the ACTIVREC queue manage attribute set to QUEUE. Channel QM1.TO.QM2 must have been restarted for this to take effect.

This example demonstrates how to detect when activity reports are delivered to queues other than the reply-to queue. Once detected, the IBM MQ display route application is used to read activity reports from another queue.

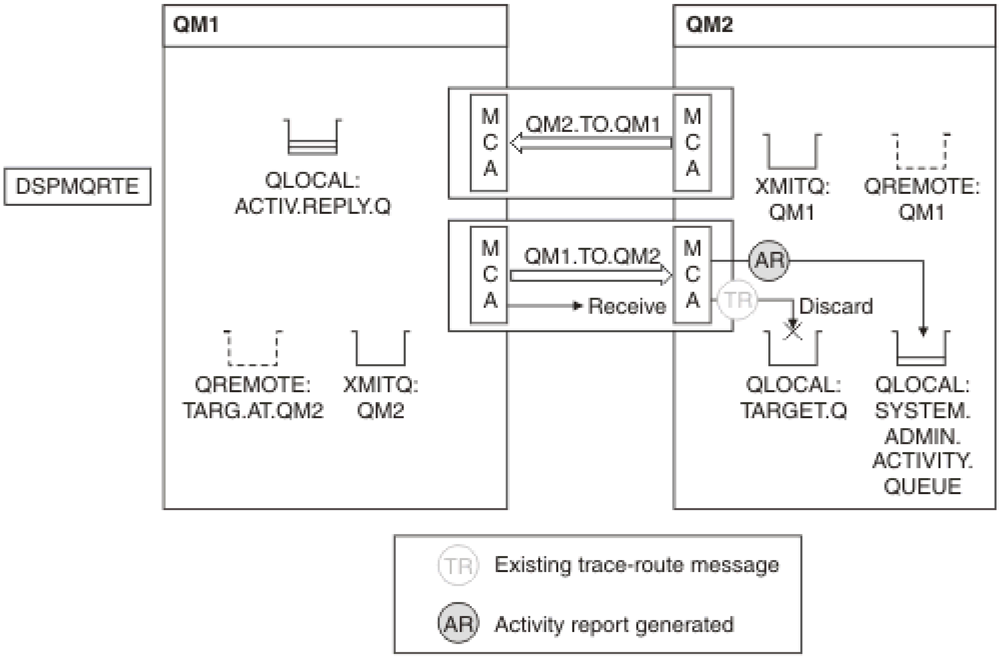


Figure 23. Delivering activity reports to the system queue, Diagram 1

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.
- The value of the ACTIVREC queue manager attribute on QM2 is now QUEUE, therefore the MCA generates an activity report, but puts it on the system queue (SYSTEM.ADMIN.ACTIVITY.QUEUE) and not on the reply queue (ACTIV.REPLY.Q).

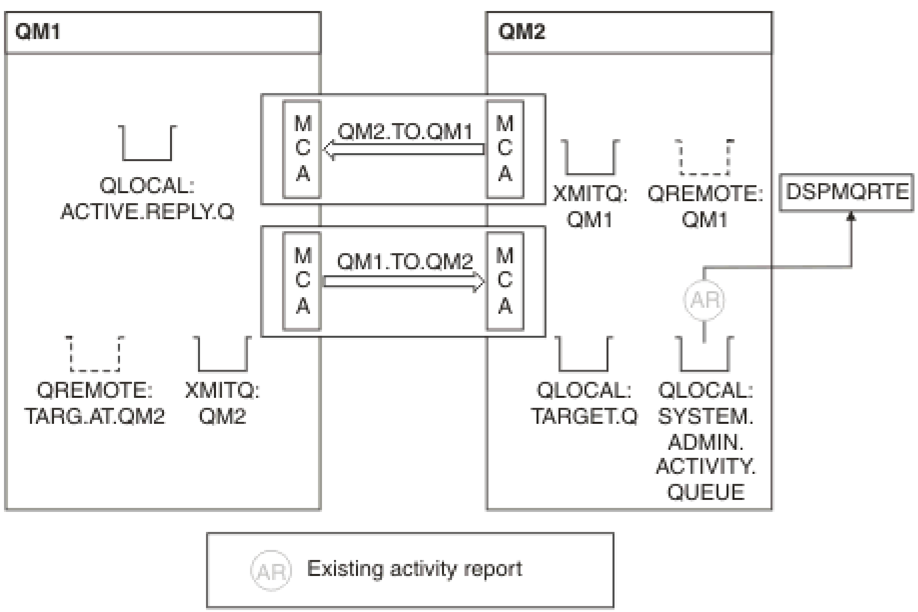


Figure 24. Delivering activity reports to the system queue, Diagram 2

- Meanwhile DSPMQRTE has been waiting for activity reports to arrive on ACTIV.REPLY.Q. Only two arrive. DSPMQRTE continues waiting for 120 seconds because it seems that the route is not yet complete.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
          ACTIV.REPLY.Q -v outline identifiers'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
          manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put

  Message:

   MQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  QName: 'TARG.AT.QM2                                     '
  ResolvedQName: 'QM2                                             '
  RemoteQName: 'TARGET.Q                                        '
  RemoteQMgrName: 'QM2                                             '
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\runmqchl.EXE'

 Operation:
  OperationType: Get

  Message:

   MQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001505'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001502'

   EmbeddedMQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  QName: 'QM2                                             '
  ResolvedQName: 'QM2                                             '

 Operation:
  OperationType: Send

  Message:

   MQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  RemoteQMgrName: 'QM2                                             '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Sender
  XmitQName: 'QM2                                             '
--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- The last operation that DSPMQRTE observed was a Send, therefore the channel is running. Now we must work out why we did not receive any more activity reports from queue manager QM2 (as identified in RemoteQMgrName).
- To check whether there is any activity information on the system queue, start DSPMQRTE on QM2 to try and collect more activity reports. Use the following command to start DSPMQRTE:

  ```
  dspmqrte -m QM2 -q SYSTEM.ADMIN.ACTIVITY.QUEUE
           -i 414D51204C4152474551202020202020A3C9154220001502 -v outline
  ```

  where 414D51204C4152474551202020202020A3C9154220001502 is the MsgId of the trace-route message that was put.
- DSPMQRTE then performs a sequence of MQGETs again, waiting for responses on the system activity queue related to the trace-route message with the specified identifier.

- DSPMQRTE gets one more activity report, which it displays. DSPMQRTE determines that the preceding activity reports are missing, and displays a message saying this. We already know about this part of the route, however.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM2
         -q SYSTEM.ADMIN.ACTIVITY.QUEUE
         -i 414D51204C4152474551202020202020A3C915420001502 -v outline'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------

Activity:
 Activity information unavailable.

--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\AMQRMPPA.EXE'

 Operation:
  OperationType: Receive
  QMgrName: 'QM2                                             '
  RemoteQMgrName: 'QM1                                             '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Receiver

 Operation:
  OperationType: Discard
  QMgrName: 'QM2                                             '
  QName: 'TARGET.Q                                        '
  Feedback: NotDelivered

--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- This activity report indicates that the route information is now complete. No problem occurred.
- Just because route information is unavailable, or because DSPMQRTE cannot display all of the route, this does not mean that the message was not delivered. For example, the queue manager attributes of different queue managers might be different, or a reply queue might not be defined to get the response back.

Example 4 - Diagnosing a channel problem
Diagnose a problem in which the trace-route message does not reach the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, generates a trace-route message, then attempts to deliver it to the target queue, TARGET.Q, on remote queue manager, QM2. In this example the trace-route message does not reach the target queue. The available activity report is used to diagnose the problem.

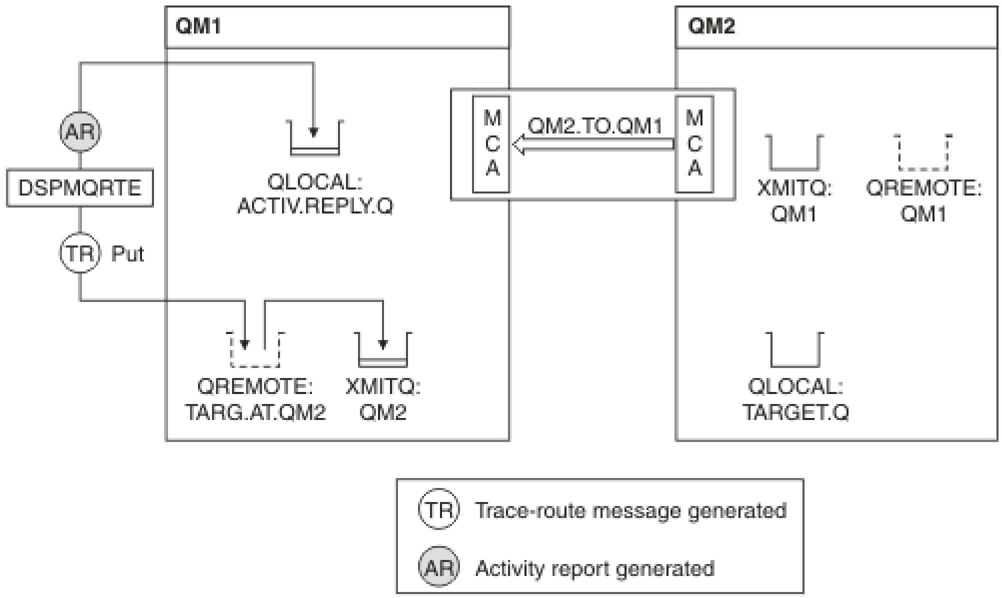


Figure 25. Diagnosing a channel problem

- In this example, the channel QM1.TO.QM2 is not running.
- DSPMQRTE puts a trace-route message (as in example 1) to the target queue and generates an activity report.
- There is no MCA to get the message from the transmission queue (QM2), therefore this is the only activity report that DSPMQRTE gets back from the reply queue. This time the fact that the route is not complete does indicate a problem. The administrator can use the transmission queue found in ResolvedQName to investigate why the transmission queue is not being serviced.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
         -rq ACTIV.REPLY.Q -v outline'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
         queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put
  QMgrName: 'QM1                                            '
  QName: 'TARG.AT.QM2                                    '
  ResolvedQName: 'QM2                                            '
  RemoteQName: 'TARGET.Q                                      '
  RemoteQMgrName: 'QM2                                            '

--------------------------------------------------------------------------------
 AMQ8652: DSPMQRTE command has finished.
```

Activity report reference

Use this page to obtain an overview of the activity report message format. The activity report message data contains the parameters that describe the activity.

Activity report format

Activity reports are standard IBM MQ report messages containing a message descriptor and message data. Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message as it has been routed through a queue manager network.

Activity reports contain the following information:

A message descriptor
An MQMD structure

Message data
Consists of the following:

- An embedded PCF header (MQEPH).
- Activity report message data.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group.

Table 19 on page 107 shows the structure of these reports, including parameters that are returned only under certain conditions.

Table 19. Activity report format

MQMD structure	Embedded PCF header MQEPH structure	Activity report message data
Structure identifier	Structure identifier	Activity
Structure version	Structure version	Activity application name
Report options	Structure length	Activity application type
Message type	Encoding	Activity description
Expiration time	Coded character set ID	Operation
Feedback	Message format	Operation type
Encoding	Flags	Operation date
Coded character set ID	PCF header (MQCFH)	Operation time
Message format	Structure type	Message
Priority	Structure length	Message length
Persistence	Structure version	MQMD [8]
Message identifier	Command identifier	EmbeddedMQMD
Correlation identifier	Message sequence number	Queue manager name
Backout count	Control options	Queue sharing group name
Reply-to queue	Completion code	Queue name [1] [2] [3] [7]
Reply-to queue manager	Reason code	Resolved queue name [1] [3] [7]
User identifier	Parameter count	Remote queue name [3] [7]
Accounting token		Remote queue manager name [2] [3] [4] [5] [7]
Application identity data		Subscription level [9]
Application type		Subscription identifier [9]
Application name		Feedback [2] [10]
Put date		Channel name [4] [5]
Put time		Channel type [4] [5]
Application origin data		Transmission queue name [5]
Group identifier		TraceRoute [6]
Message sequence number		Detail
Offset		Recorded activities
Message flags		Unrecorded activities
Original length		Discontinuity count
		Max activities
		Accumulate
		Deliver

Notes:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.

5. Returned for Send operations.
6. Returned for trace-route messages.
7. Not returned for Put operations to a topic, contained within Publish activities.
8. Not returned for Excluded Publish operations. For Publish and Discarded Publish operations, returned containing a subset of parameters.
9. Returned for Publish, Discarded Publish, and Excluded Publish operations.
10. Returned for Discarded Publish and Excluded Publish operations.

Activity report MQMD (message descriptor)

Use this page to view the values contained by the MQMD structure for an activity report

StrucId
Structure identifier:

Data type
MQCHAR4

Value
MQMD_STRUC_ID.

Version
Structure version number

Data type
MQLONG

Values
Copied from the original message descriptor. Possible values are:

MQMD_VERSION_1
Version-1 message descriptor structure, supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure, supported on AIX®, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, and all IBM MQ MQI clients connected to these systems.

Report
Options for further report messages

Data type
MQLONG

Value
If MQRO_PASS_DISCARD_AND_EXPIRY or MQRO_DISCARD_MSG were specified in the *Report* field of the original message descriptor:

MQRO_DISCARD
The report is discarded if it cannot be delivered to the destination queue.

Otherwise:

MQRO_NONE
No reports required.

MsgType
Indicates type of message

Data type
MQLONG

Value
MQMT_REPORT

Expiry
Report message lifetime

Data type
MQLONG

Value
If the *Report* field in the original message descriptor is specified as MQRO_PASS_DISCARD_AND_EXPIRY, the remaining expiry time from the original message is used.

Otherwise:

MQEI_UNLIMITED
The report does not have an expiry time.

Feedback

Description:	Feedback or reason code.
Data type:	MQLONG.
Value:	**MQFB_ACTIVITY** Activity report.

Encoding

Description:	Numeric encoding of report message data.
Data type:	MQLONG.
Value:	MQENC_NATIVE.

CodedCharSetId

Description:	Character set identifier of report message data.
Data type:	MQLONG.
Value:	Set as appropriate.

Format

Description:	Format name of report message data
Data type:	MQCHAR8.
Value:	**MQFMT_EMBEDDED_PCF** Embedded PCF message.

Priority

Description:	Report message priority.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

Persistence

Description:	Report message persistence.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

MsgId

Description:	Message identifier.

Data type:	MQBYTE24.
Values:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_MSG_ID, the message identifier from the original message is used. Otherwise, a unique value will be generated by the queue manager.

CorrelId

Description:	Correlation identifier.
Data type:	MQBYTE24.
Value:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_CORREL_ID, the correlation identifier from the original message is used. Otherwise, the message identifier is copied from the original message.

BackoutCount

Description:	Backout counter.
Data type:	MQLONG.
Value:	0.

ReplyToQ

Description:	Name of reply queue.
Data type:	MQCHAR48.
Values:	Blank.

ReplyToQMgr

Description:	Name of reply queue manager.
Data type:	MQCHAR48.
Value:	The queue manager name that generated the report message.

UserIdentifier

Description:	The user identifier of the application that generated the report message.
Data type:	MQCHAR12.
Value:	Copied from the original message descriptor.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Data type:	MQBYTE32.
Value:	Copied from the original message descriptor.

ApplIdentityData

Description:	Application data relating to identity.
Data type:	MQCHAR32.
Values:	Copied from the original message descriptor.

PutApplType

Description: Type of application that put the report message.

Data type: MQLONG.

Value: **MQAT_QMGR**
Queue manager generated message.

PutApplName

Description: Name of application that put the report message.

Data type: MQCHAR28.

Value: Either the first 28 bytes of the queue manager name, or the name of the MCA that generated the report message.

PutDate

Description: Date when message was put.

Data type: MQCHAR8.

Value: As generated by the queue manager.

PutTime

Description: Time when message was put.

Data type: MQCHAR8.

Value: As generated by the queue manager.

ApplOriginData

Description: Application data relating to origin.

Data type: MQCHAR4.

Value: Blank.

If *Version* is MQMD_VERSION_2, the following additional fields are present:

GroupId

Description: Identifies to which message group or logical message the physical message belongs.

Data type: MQBYTE24.

Value: Copied from the original message descriptor.

MsgSeqNumber

Description: Sequence number of logical message within group.

Data type: MQLONG.

Value: Copied from the original message descriptor.

Offset

Description: Offset of data in physical message from start of logical message.

Data type: MQLONG.

Value: Copied from the original message descriptor.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

OriginalLength

Description:	Length of original message.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

Activity report MQEPH (Embedded PCF header)

Use this page to view the values contained by the MQEPH structure for an activity report

The MQEPH structure contains a description of both the PCF information that accompanies the message data of an activity report, and the application message data that follows it.

For an activity report, the MQEPH structure contains the following values:

StrucId

Description:	Structure identifier.
Data type:	MQCHAR4.
Value:	MQEPH_STRUC_ID.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *Encoding* field of the original message descriptor. Otherwise, 0.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.

Data type: MQLONG.

Value: If any data from the original application message data is included in the report message, the value will be copied from the *CodedCharSetId* field of the original message descriptor.

Otherwise, MQCCSI_UNDEFINED.

Format

Description: Format name of message data that follows the last PCF parameter structure.

Data type: MQCHAR8.

Value: If any data from the original application message data is included in the report message, the value will be copied from the *Format* field of the original message descriptor.

Otherwise, MQFMT_NONE.

Flags

Description: Flags that specify attributes of the structure or control its processing.

Data type: MQLONG.

Value: **MQEPH_CCSID_EMBEDDED**
Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description: Programmable Command Format Header

Data type: MQCFH.

Value: See "Activity report MQCFH (PCF header)" on page 113.

Activity report MQCFH (PCF header)

Use this page to view the PCF values contained by the MQCFH structure for an activity report

For an activity report, the MQCFH structure contains the following values:

Type

Description: Structure type that identifies the content of the report message.

Data type: MQLONG.

Value: **MQCFT_REPORT**
Message is a report.

StrucLength

Description: Structure length.

Data type: MQLONG.

Value: **MQCFH_STRUC_LENGTH**
Length in bytes of MQCFH structure.

Version

Description: Structure version number.

Data type: MQLONG.

Values: MQCFH_VERSION_3

Command

Description: Command identifier. This identifies the category of the message.

Data type: MQLONG.

Values: **MQCMD_ACTIVITY_MSG**
Message activity.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a group of related messages.

Data type: MQLONG.

Values: 1.

Control

Description: Control options.

Data type: MQLONG.

Values: MQCFC_LAST.

CompCode

Description: Completion code.

Data type: MQLONG.

Values: MQCC_OK.

Reason

Description: Reason code qualifying completion code.

Data type: MQLONG.

Values: MQRC_NONE.

ParameterCount

Description: Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.

Data type: MQLONG.

Values: 1 or greater.

Activity report message data

Use this page to view the parameters contained by the *Activity* PCF group in an activity report message. Some parameters are returned only when specific operations have been performed.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group. The *Activity* PCF group is detailed in this topic.

Some parameters, which are described as Operation-specific activity report message data, are returned only when specific operations have been performed.

For an activity report, the activity report message data contains the following parameters:

Activity

Description:	Grouped parameters describing the activity.
Identifier:	MQGACF_ACTIVITY.
Data type:	MQCFGR.
Included in PCF group:	None.
Parameters in PCF group:	*ActivityApplName* *ActivityApplType* *ActivityDescription* *Operation* *TraceRoute*
Returned:	Always.

ActivityApplName

Description:	Name of application that performed the activity.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ActivityApplType

Description:	Type of application that performed the activity.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Activity*.
Returned:	Always.

ActivityDescription

Description:	Description of activity performed by the application.
Identifier:	MQCACF_ACTIVITY_DESCRIPTION.
Data type:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	64
Returned:	Always.

Operation

Description:	Grouped parameters describing an operation of the activity.

Identifier: MQGACF_OPERATION.

Data type: MQCFGR.

Included in PCF group: *Activity*.

Parameters in PCF group:

OperationType
OperationDate
OperationTime
Message
QMgrName
QSGName

Note: Additional parameters are returned in this group depending on the operation type. These additional parameters are described as Operation-specific activity report message data.

Returned: One *Operation* PCF group per operation in the activity.

OperationType

Description: Type of operation performed.

Identifier: MQIACF_OPERATION_TYPE.

Data type: MQCFIN.

Included in PCF group: *Operation*.

Values: MQOPER_*.

Returned: Always.

OperationDate

Description: Date when the operation was performed.

Identifier: MQCACF_OPERATION_DATE.

Data type: MQCFST.

Included in PCF group: *Operation*.

Maximum length: MQ_DATE_LENGTH.

Returned: Always.

OperationTime

Description: Time when the operation was performed.

Identifier: MQCACF_OPERATION_TIME.

Data type: MQCFST.

Included in PCF group: *Operation*.

Maximum length: MQ_TIME_LENGTH.

Returned: Always.

Message

Description:	Grouped parameters describing the message that caused the activity.
Identifier:	MQGACF_MESSAGE.
Data type:	MQCFGR.
Included in PCF group:	*Operation*.
Parameters in group:	*MsgLength* *MQMD* *EmbeddedMQMD*
Returned:	Always, except for Excluded Publish operations.

MsgLength

Description:	Length of the message that caused the activity, before the activity occurred.
Identifier:	MQIACF_MSG_LENGTH.
Data type:	MQCFIN.
Included in PCF group:	*Message*.
Returned:	Always.

MQMD

Description:	Grouped parameters related to the message descriptor of the message that caused the activity.
Identifier:	MQGACF_MQMD.
Data type:	MQCFGR.
Included in PCF group:	*Message*.

Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	Always, except for Excluded Publish operations.

EmbeddedMQMD

Description:	Grouped parameters describing the message descriptor embedded within a message on a transmission queue.
Identifier:	MQGACF_EMBEDDDED_MQMD.
Data type:	MQCFGR.
Included in PCF group:	*Message*.

Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	For Get operations where the queue resolves to a transmission queue.

StrucId

Description:	Structure identifier
Identifier:	MQCACF_STRUC_ID.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	4.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Version

Description:	Structure version number.
Identifier:	MQIACF_VERSION.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.

Returned: Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Report

Description: Options for report messages.

Identifier: MQIACF_REPORT.

Data type: MQCFIN.

Included in PCF group: *MQMD* or *EmbeddedMQMD*.

Returned: Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

MsgType

Description: Indicates type of message.

Identifier: MQIACF_MSG_TYPE.

Data type: MQCFIN.

Included in PCF group: *MQMD* or *EmbeddedMQMD*.

Returned: Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Expiry

Description: Message lifetime.

Identifier: MQIACF_EXPIRY.

Data type: MQCFIN.

Included in PCF group: *MQMD* or *EmbeddedMQMD*.

Returned: Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Feedback

Description: Feedback or reason code.

Identifier: MQIACF_FEEDBACK.

Data type: MQCFIN.

Included in PCF group: *MQMD* or *EmbeddedMQMD*.

Returned: Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Encoding

Description: Numeric encoding of message data.

Identifier: MQIACF_ENCODING.

Data type: MQCFIN.

Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

CodedCharSetId

Description:	Character set identifier of message data.
Identifier:	MQIA_CODED_CHAR_SET_ID.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Format

Description:	Format name of message data
Identifier:	MQCACH_FORMAT_NAME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_FORMAT_LENGTH.
Returned:	Always, except for Excluded Publish operations.

Priority

Description:	Message priority.
Identifier:	MQIACF_PRIORITY.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

Persistence

Description:	Message persistence.
Identifier:	MQIACF_PERSISTENCE.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

MsgId

Description:	Message identifier.
Identifier:	MQBACF_MSG_ID.
Data type:	MQCFBS.

Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_MSG_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

CorrelId

Description:	Correlation identifier.
Identifier:	MQBACF_CORREL_ID.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_CORREL_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

BackoutCount

Description:	Backout counter.
Identifier:	MQIACF_BACKOUT_COUNT.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQ

Description:	Name of reply queue.
Identifier:	MQCACF_REPLY_TO_Q.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQMgr

Description:	Name of reply queue manager.
Identifier:	MQCACF_REPLY_TO_Q_MGR.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

UserIdentifier

Description:	The user identifier of the application that originated the message.
Identifier:	MQCACF_USER_IDENTIFIER.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Identifier:	MQBACF_ACCOUNTING_TOKEN.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_ACCOUNTING_TOKEN_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

ApplIdentityData

Description:	Application data relating to identity.
Identifier:	MQCACF_APPL_IDENTITY_DATA.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_IDENTITY_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

PutApplType

Description:	Type of application that put the message.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutApplName

Description:	Name of application that put the message.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.

Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutDate

Description:	Date when message was put.
Identifier:	MQCACF_PUT_DATE.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_DATE_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutTime

Description:	Time when message was put.
Identifier:	MQCACF_PUT_TIME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_TIME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

ApplOriginData

Description:	Application data relating to origin.
Identifier:	MQCACF_APPL_ORIGIN_DATA.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

GroupId

Description:	Identifies to which message group or logical message the physical message belongs.
Identifier:	MQBACF_GROUP_ID.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_GROUP_ID_LENGTH.
Returned:	If the *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgSeqNumber

Description:	Sequence number of logical message within group.
Identifier:	MQIACH_MSG_SEQUENCE_NUMBER.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

Offset

Description:	Offset of data in physical message from start of logical message.
Identifier:	MQIACF_OFFSET.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Identifier:	MQIACF_MSG_FLAGS.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

OriginalLength

Description:	Length of original message.
Identifier:	MQIACF_ORIGINAL_LENGTH.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

QMgrName

Description:	Name of the queue manager where the activity was performed.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH

Returned:	Always.

QSGName

Description:	Name of the queue sharing group to which the queue manager where the activity was performed belongs.
Identifier:	MQCA_QSG_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_QSG_NAME_LENGTH
Returned:	If the activity was performed on an IBM MQ for z/OS queue manager.

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message.
Identifier:	MQGACF_TRACE_ROUTE.
Data type:	MQCFGR.
Contained in PCF group:	*Activity*.
Parameters in group:	*Detail* *RecordedActivities* *UnrecordedActivities* *DiscontinuityCount* *MaxActivities* *Accumulate* *Forward* *Deliver*
Returned:	If the activity was performed on behalf of the trace-route message.

The values of the parameters in the *TraceRoute* PCF group are those from the trace-route message at the time the activity report was generated.

Operation-specific activity report message data

Use this page to view the additional PCF parameters that might be returned in the PCF group *Operation* in an activity report, depending on the value of the *OperationType* parameter

The additional parameters vary depending on the following operation types:

Get/Browse (MQOPER_GET/MQOPER_BROWSE)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Get/Browse (MQOPER_GET/MQOPER_BROWSE) operation type (a message on a queue was got, or browsed).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.

Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

Discard (MQOPER_DISCARD)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Discard (MQOPER_DISCARD) operation type (a message was discarded).

Feedback

Description:	The reason for the message being discarded.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was unsuccessfully put to a queue.

RemoteQMgrName

Description:	The name of the queue manager to which the message was destined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the value of *Feedback* is MQFB_NOT_FORWARDED.

Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/MQOPER_DISCARDED_PUBLISH/ MQOPER_EXCLUDED_PUBLISH)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/MQOPER_DISCARDED_PUBLISH/ MQOPER_EXCLUDED_PUBLISH) operation type (a publish/subscribe message was delivered, discarded, or excluded).

SubId

Description:	The subscription identifier.
Identifier:	MQBACF_SUB_ID.
Data type:	MQCFBS.
Included in PCF group:	*Operation*.
Returned:	Always.

SubLevel

Description:	The subscription level.
Identifier:	MQIACF_SUB_LEVEL.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

Feedback

Description:	The reason for discarding the message.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was not delivered to a subscriber, or the message was not delivered because the subscriber was excluded.

The Publish operation MQOPER_PUBLISH provides information about a message delivered to a particular subscriber. This operation describes the elements of the onward message that might have changed from the message described in the associated Put operation. Similarly to a Put operation, it contains a message group MQGACF_MESSAGE and, inside that, an MQMD group MQGACF_MQMD. However, this MQMD group contains only the following fields, which can be overridden by a subscriber: *Format, Priority, Persistence, MsgId, CorrelId, UserIdentifier, AccountingToken, ApplIdentityData.*

The *SubId* and *SubLevel* of the subscriber are included in the operation information. You can use the *SubID* with the MQCMD_INQUIRE_SUBSCRIBER PCF command to retrieve all other attributes for a subscriber.

The Discarded Publish operation MQOPER_DISCARDED_PUBLISH is analogous to the Discard operation that is used when a message is not delivered in point-to-point messaging. A message is not delivered to a subscriber if the message was explicitly requested not to be delivered to a local destination and this subscriber specifies a local destination. A message is also considered not delivered if there is a problem getting the message to the destination queue, for example, because the queue is full.

The information in a Discarded Publish operation is the same as for a Publish operation, with the addition of a *Feedback* field that gives the reasons why the message was not delivered. This feedback field contains MQFB_* or MQRC_* values that are common with the MQOPER_DISCARD operation. The reason for discarding a publish, as opposed to excluding it, are the same as the reasons for discarding a put.

The Excluded Publish operation MQOPER_EXCLUDED_PUBLISH provides information about a subscriber that was considered for delivery of the message, because the topic on which the subscriber is subscribing matches that of the associated Put operation, but the message was not delivered to the subscriber because other selection criteria do not match with the message that is being put to the topic. As with a Discarded Publish operation, the *Feedback* field provides information about the reason why this subscription was excluded. However, unlike the Discarded Publish operation, no message-related information is provided because no message was generated for this subscriber.

Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/MQOPER_PUT_REPORT)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/MQOPER_PUT_REPORT) operation type (a message, reply message, or report message was put to a queue).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always, apart from one exception: not returned if the Put operation is to a topic, contained within a publish activity.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When the opened queue could be resolved. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQName

Description:	The name of the opened queue, as it is known on the remote queue manager.
Identifier:	MQCA_REMOTE_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQMgrName

Description:	The name of the remote queue manager on which the remote queue is defined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

TopicString

Description:	The full topic string to which the message is being put.
Identifier:	MQCA_TOPIC_STRING.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Returned:	If the Put operation is to a topic, contained within a publish activity.

Feedback

Description:	The reason for the message being put on the dead-letter queue.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was put on the dead-letter queue.

Receive (MQOPER_RECEIVE)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Receive (MQOPER_RECEIVE) operation type (a message was received on a channel).

ChannelName

Description:	The name of the channel on which the message was received.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	Always.

ChannelType

Description:	The type of channel on which the message was received.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.

Included in PCF group:	*Operation*.
Returned:	Always.

RemoteQMgrName

Description:	The name of the queue manager from which the message was received.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Send (MQOPER_SEND)
The additional activity report message data parameters that are returned in the PCF group *Operation* for the Send (MQOPER_SEND) operation type (a message was sent on a channel).

ChannelName

Description:	The name of the channel where the message was sent.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The type of channel where the message was sent.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

XmitQName

Description:	The transmission queue from which the message was retrieved.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

RemoteQMgrName

Description:	The name of the remote queue manager to which the message was sent.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Trace-route message reference

Use this page to obtain an overview of the trace-route message format. The trace-route message data includes parameters that describe the activities that the trace-route message has caused

Trace-route message format

Trace-route messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network.

Trace-route messages contain the following information:

A message descriptor
An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
Consists of either:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN, or
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF.

When using the IBM MQ display route application to generate a trace-route message, *Format* is set to MQFMT_ADMIN.

The content of the trace-route message data is determined by the *Accumulate* parameter from the *TraceRoute* PCF group, as follows:

- If *Accumulate* is set to MQROUTE_ACCUMULATE_NONE, the trace-route message data contains the *TraceRoute* PCF group.
- If *Accumulate* is set to either MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, the trace-route message data contains the *TraceRoute* PCF group and zero or more *Activity* PCF groups.

Table 20 on page 133 shows the structure of a trace-route message.

Table 20. Trace-route message format

MQMD structure	Embedded PCF header MQEPH structure	Trace-route message data
Structure identifier Structure version Report options Message type Expiration time Feedback Encoding Coded character set ID Message format Priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier Accounting token Application identity data Application type Application name Put date Put time Application origin data Group identifier Message sequence number Offset Message flags Original length	Structure identifier Structure version Structure length Encoding Coded character set ID Message format Flags PCF header (MQCFH) Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	TraceRoute Detail Recorded activities Unrecorded activities Discontinuity count Max activities Accumulate Deliver

Trace-route message MQMD (message descriptor)

Use this page to view the values contained by the MQMD structure for a trace-route message

StrucId

Description: Structure identifier.

Data type: MQCHAR4.

Value: MQMD_STRUC_ID.

Version

Description: Structure version number.

Data type: MQLONG.

Values: **MQMD_VERSION_1.**

Report

Description: Options for report messages.

Data type: MQLONG.

Value: Set according to requirements. Common report options follow:

MQRO_DISCARD_MSG
The message is discarded on arrival to a local queue.

MQRO_PASS_DISCARD_AND_EXPIRY
Every response (activity reports or trace-route reply message) will have the report option MQRO_DISCARD_MSG set, and the remaining expiry passed on. This ensures that responses do not remain in the queue manager network indefinitely.

MsgType

Description: Type of message.

Data type: MQLONG.

Value: If the *Accumulate* parameter in the TraceRoute group is specified as MQROUTE_ACCUMULATE_AND_REPLY, then message type is MQMT_REQUEST

Otherwise:

MQMT_DATAGRAM.

Expiry

Description: Message lifetime.

Data type: MQLONG.

Value: Set according to requirements. This parameter can be used to ensure trace-route messages are not left in a queue manager network indefinitely.

Feedback

Description: Feedback or reason code.

Data type: MQLONG.

Value: **MQFB_NONE.**

Encoding

Description: Numeric encoding of message data.

Data type: MQLONG.

Value: Set as appropriate.

CodedCharSetId

Description: Character set identifier of message data.

Data type: MQLONG.

Value: Set as appropriate.

Format

Description: Format name of message data

Data type: MQCHAR8.

Value:

MQFMT_ADMIN
Admin message. No user data follows the *TraceRoute* PCF group.

MQFMT_EMBEDDED_PCF
Embedded PCF message. User data follows the *TraceRoute* PCF group.

Priority

Description: Message priority.

Data type: MQLONG.

Value: Set according to requirements.

Persistence

Description: Message persistence.

Data type: MQLONG.

Value: Set according to requirements.

MsgId

Description: Message identifier.

Data type: MQBYTE24.

Value: Set according to requirements.

CorrelId

Description: Correlation identifier.

Data type: MQBYTE24.

Value: Set according to requirements.

BackoutCount

Description: Backout counter.

Data type: MQLONG.

Value: 0.

ReplyToQ

Description: Name of reply queue.

Data type: MQCHAR48.

Values: Set according to requirements.

If *MsgType* is set to MQMT_REQUEST or if *Report* has any report generating options set, then this parameter must be non-blank.

ReplyToQMgr

Description: Name of reply queue manager.

Data type: MQCHAR48.

Value: Set according to requirements.

UserIdentifier

Description:	The user identifier of the application that originated the message.
Data type:	MQCHAR12.
Value:	Set as normal.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Data type:	MQBYTE32.
Value:	Set as normal.

ApplIdentityData

Description:	Application data relating to identity.
Data type:	MQCHAR32.
Values:	Set as normal.

PutApplType

Description:	Type of application that put the message.
Data type:	MQLONG.
Value:	Set as normal.

PutApplName

Description:	Name of application that put the message.
Data type:	MQCHAR28.
Value:	Set as normal.

PutDate

Description:	Date when message was put.
Data type:	MQCHAR8.
Value:	Set as normal.

PutTime

Description:	Time when message was put.
Data type:	MQCHAR8.
Value:	Set as normal.

ApplOriginData

Description:	Application data relating to origin.
Data type:	MQCHAR4.
Value:	Set as normal..

Trace-route message MQEPH (Embedded PCF header)

Use this page to view the values contained by the MQEPH structure for a trace-route message

The MQEPH structure contains a description of both the PCF information that accompanies the message data of a trace-route message, and the application message data that follows it. An MQEPH structure is used only if additional user message data follows the TraceRoute PCF group.

For a trace-route message, the MQEPH structure contains the following values:

StrucId

Description:	Structure identifier.
Data type:	MQCHAR4.
Value:	MQEPH_STRUC_ID.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	The encoding of the message data.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	The character set of the message data.

Format

Description:	Format name of the message data that follows the last PCF parameter structure.
Data type:	MQCHAR8.
Value:	The format name of the message data.

Flags

Description:	Flags that specify attributes of the structure or control its processing.
Data type:	MQLONG.

Value: **MQEPH_NONE**
No flags specified.

MQEPH_CCSID_EMBEDDED
Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description: Programmable Command Format Header

Data type: MQCFH.

Value: See "Trace-route message MQCFH (PCF header)" on page 138.

Trace-route message MQCFH (PCF header)

Use this page to view the PCF values contained by the MQCFH structure for a trace-route message

For a trace-route message, the MQCFH structure contains the following values:

Type

Description: Structure type that identifies the content of the message.

Data type: MQLONG.

Value: **MQCFT_TRACE_ROUTE**
Message is a trace-route message.

StrucLength

Description: Structure length.

Data type: MQLONG.

Value: **MQCFH_STRUC_LENGTH**
Length in bytes of MQCFH structure.

Version

Description: Structure version number.

Data type: MQLONG.

Values: MQCFH_VERSION_3

Command

Description: Command identifier. This identifies the category of the message.

Data type: MQLONG.

Values: **MQCMD_TRACE_ROUTE**
Trace-route message.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a group of related messages.

Data type: MQLONG.

Values: 1.

Control

Description:	Control options.
Data type:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Data type:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	1 or greater.

Trace-route message data

Use this page to view the parameters that make up the *TraceRoute* PCF group part of trace-route message data

The content of trace-route message data depends on the *Accumulate* parameter from the *TraceRoute* PCF group. Trace-route message data consists of the *TraceRoute* PCF group, and zero or more *Activity* PCF groups. The *TraceRoute* PCF group is detailed in this topic. Refer to the related information for details of the *Activity* PCF group.

Trace-route message data contains the following parameters:

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message. For a trace-route message, some of these parameters can be altered to control how it is processed.
Identifier:	MQGACF_TRACE_ROUTE.
Data type:	MQCFGR.
Contained in PCF group:	None.

Parameters in group:
Detail
RecordedActivities
UnrecordedActivities
DiscontinuityCount
MaxActivities
Accumulate
Forward
Deliver

Detail

Description: The detail level that will be recorded for the activity.

Identifier: MQIACF_ROUTE_DETAIL.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

Values:

MQROUTE_DETAIL_LOW
Activities performed by user-written application are recorded.

MQROUTE_DETAIL_MEDIUM
Activities specified in MQROUTE_DETAIL_LOW are recorded. Additionally, activities performed by MCAs are recorded.

MQROUTE_DETAIL_HIGH
Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM are recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user-written applications that are to record further activity information.

RecordedActivities

Description: The number of activities that the trace-route message has caused, where information was recorded.

Identifier: MQIACF_RECORDED_ACTIVITIES.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

UnrecordedActivities

Description: The number of activities that the trace-route message has caused, where information was not recorded.

Identifier: MQIACF_UNRECORDED_ACTIVITIES.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

DiscontinuityCount

Description: The number of times a trace-route message has been received from a queue manager that does not support trace-route messaging.

Identifier: MQIACF_DISCONTINUITY_COUNT.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

MaxActivities

Description: The maximum number of activities the trace-route message can be involved in before it stops being processed.

Identifier: MQIACF_MAX_ACTIVITIES.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

Value:

A positive integer
The maximum number of activities.

MQROUTE_UNLIMITED_ACTIVITIES
An unlimited number of activities.

Accumulate

Description: Specifies whether activity information is accumulated within the trace-route message, and whether a reply message containing the accumulated activity information is generated before the trace-route message is discarded or is put on a non-transmission queue.

Identifier: MQIACF_ROUTE_ACCUMULATION.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

Value:

MQROUTE_ACCUMULATE_NONE
Activity information is not accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_IN_MSG
Activity information is accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_AND_REPLY
Activity information is accumulated in the message data of the trace-route message, and a trace-route reply message will be generated.

Forward

Description: Specifies queue managers that the trace-route message can be forwarded to. When determining whether to forward a message to a remote queue manager, queue managers use the algorithm that is described in Forwarding.

Identifier: MQIACF_ROUTE_FORWARDING.

Data type: MQCFIN.

Contained in PCF group: *TraceRoute*.

Value:

MQROUTE_FORWARD_IF_SUPPORTED
The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Deliver

Description:	Specifies the action to be taken if the trace-route message arrives at the destination queue successfully.
Identifier:	MQIACF_ROUTE_DELIVERY.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	**MQROUTE_DELIVER_YES** On arrival, the trace-route message is put on the target queue. Any application performing a destructive get on the target queue can receive the trace-route message. **MQROUTE_DELIVER_NO** On arrival, the trace-route message is discarded.

Trace-route reply message reference

Use this page to obtain an overview of the trace-route reply message format. The trace-route reply message data is a duplicate of the trace-route message data from the trace-route message for which it was generated

Trace-route reply message format

Trace-route reply messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network.

Trace-route reply messages contain the following information:

A message descriptor
An MQMD structure

Message data
A PCF header (MQCFH) and trace-route reply message data

Trace-route reply message data consists of one or more *Activity* PCF groups.

When a trace-route message reaches its target queue, a trace-route reply message can be generated that contains a copy of the activity information from the trace-route message. The trace-route reply message will be delivered to a reply-to queue or to a system queue.

Table 21 on page 143 shows the structure of a trace-route reply message, including parameters that are only returned under certain conditions.

Table 21. Trace-route reply message format		
MQMD structure	**PCF header MQCFH structure**	**Trace-route reply message data**
Structure identifier Structure version Report options Message type Expiration time Feedback Encoding Coded character set ID Message format Priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier Accounting token Application identity data Application type Application name Put date Put time Application origin data Group identifier Message sequence number Offset Message flags Original length	PCF header (MQCFH) Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Activity Activity application name Activity application type Activity description Operation Operation type Operation date Operation time Message Message length MQMD EmbeddedMQMD Queue manager name Queue sharing group name Queue name [1 2 3] Resolved queue name [1 3] Remote queue name [3] Remote queue manager-name [2 3 4 5] Feedback [2] Channel name [4 5] Channel type [4 5] Transmission queue name [5] TraceRoute Detail Recorded activities Unrecorded activities Discontinuity count Max activities Accumulate Deliver

Note:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.
5. Returned for Send operations.

Trace-route reply message MQMD (message descriptor)

Use this page to view the values contained by the MQMD structure for a trace-route reply message

For a trace-route reply message, the MQMD structure contains the parameters described in Activity report message descriptor. Some of the parameter values in a trace-route reply message descriptor are different from those in an activity report message descriptor, as follows:

MsgType

Description: Type of message.

Data type: MQLONG.

Value: **MQMT_REPLY**

Feedback

Description: Feedback or reason code.

Data type: MQLONG.

Value: **MQFB_NONE**

Encoding

Description: Numeric encoding of message data.

Data type: MQLONG.

Value: Copied from trace-route message descriptor.

CodedCharSetId

Description: Character set identifier of message data.

Data type: MQLONG.

Value: Copied from trace-route message descriptor.

Format

Description: Format name of message data

Data type: MQCHAR8.

Value: **MQFMT_ADMIN**
Admin message.

Trace-route reply message MQCFH (PCF header)

Use this page to view the PCF values contained by the MQCFH structure for a trace-route reply message

The PCF header (MQCFH) for a trace-route reply message is the same as for a trace-route message.

Trace-route reply message data

The trace-route reply message data is a duplicate of the trace-route message data from the trace-route message for which it was generated

The trace-route reply message data contains one or more *Activity* groups. The parameters are described in "Activity report message data" on page 114.

Accounting and statistics messages

Queue managers generate accounting and statistics messages to record information about the MQI operations performed by IBM MQ applications, or to record information about the activities occurring in an IBM MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by IBM MQ applications, see "Accounting messages" on page 145.

Statistics messages
Statistics messages are used to record information about the activities occurring in an IBM MQ system, see "Statistics messages" on page 148.

z/OS Accounting messages and statistics messages as described here are not available on IBM MQ for z/OS, but equivalent functionality is available through the System Management Facility (SMF).

Accounting and statistics messages are delivered to one of two system queues. User applications can retrieve the messages from these system queues and use the recorded information for various purposes:

- Account for application resource use.
- Record application activity.
- Capacity planning.
- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.

Related concepts

"Using System Management Facility" on page 306
You can use SMF to collect statistics and accounting information. To use SMF, certain parameters must be set in z/OS and in IBM MQ.

Accounting messages

Accounting messages record information about the MQI operations performed by IBM MQ applications. An accounting message is a PCF message that contains a number of PCF structures.

When an application disconnects from a queue manager, an accounting message is generated and delivered to the system accounting queue (SYSTEM.ADMIN.ACCOUNTING.QUEUE). For long running IBM MQ applications, intermediate accounting messages are generated as follows:

- When the time since the connection was established exceeds the configured interval.
- When the time since the last intermediate accounting message exceeds the configured interval.

Accounting messages are in the following categories:

MQI accounting messages
MQI accounting messages contain information relating to the number of MQI calls made using a connection to a queue manager.

Queue accounting messages
Queue accounting messages contain information relating to the number of MQI calls made using connections to a queue manager, grouped by queue.

Each queue accounting message can contain up to 100 records, with every record relating to an activity performed by the application with respect to a specific queue.

Accounting messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the accounting data is recorded against the base queue, and, for a remote queue, the accounting data is recorded against the transmission queue.

Related reference

"MQI accounting message data" on page 163
Use this page to view the structure of an MQI accounting message

"Queue accounting message data" on page 174
Use this page to view the structure of a queue accounting message

Accounting message format

Accounting messages comprise a set of PCF fields that consist of a message descriptor and message data.

Message descriptor

- An accounting message MQMD (message descriptor)

Accounting message data

- An accounting message MQCFH (PCF header)
- Accounting message data that is always returned
- Accounting message data that is returned if available

The accounting message MQCFH (PCF header) contains information about the application, and the interval for which the accounting data was recorded.

Accounting message data comprises PCF parameters that store the accounting information. The content of accounting messages depends on the message category as follows:

MQI accounting message
MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and in the range 1 through 100 *QAccountingData* PCF groups.

There is one *QAccountingData* PCF group for every queue that had accounting data collected. If an application accesses more than 100 queues, multiple accounting messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Accounting information collection

Use queue and queue manager attributes to control the collection of accounting information. You can also use MQCONNX options to control collection at the connection level.

MQI accounting information
Use the queue manager attribute ACCTMQI to control the collection of MQI accounting information

To change the value of this attribute, use the MQSC command, `ALTER QMGR`, and specify the parameter `ACCTMQI`. Accounting messages are generated only for connections that begin after accounting is enabled. The `ACCTMQI` parameter can have the following values:

ON
MQI accounting information is collected for every connection to the queue manager.

OFF
MQI accounting information is not collected. This is the default value.

For example, to enable MQI accounting information collection use the following MQSC command:

```
ALTER QMGR ACCTMQI(ON)
```

Queue accounting information
Use the queue attribute ACCTQ and the queue manager attribute ACCTQ to control the collection of queue accounting information.

To change the value of the queue attribute, use the MQSC command, `ALTER QLOCAL`, and specify the parameter ACCTQ. Accounting messages are generated only for connections that begin after accounting is enabled. Note that changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

The queue attribute ACCTQ can have the following values:

ON
Queue accounting information for this queue is collected for every connection to the queue manager that opens the queue.

OFF
Queue accounting information for this queue is not collected.

QMGR
: The collection of queue accounting information for this queue is controlled according to the value of the queue manager attribute ACCTQ. This is the default value.

To change the value of the queue manager attribute, use the MQSC command, `ALTER QMGR` and specify the parameter ACCTQ. The queue manager attribute ACCTQ can have the following values:

ON
: Queue accounting information is collected for queues that have the queue attribute ACCTQ set as QMGR.

OFF
: Queue accounting information is not collected for queues that have the queue attribute ACCTQ set as QMGR. This is the default value.

NONE
: The collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

If the queue manager attribute, ACCTQ, is set to NONE, the collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

For example, to enable accounting information collection for the queue, Q1, use the following MQSC command:

```
ALTER QLOCAL(Q1) ACCTQ(ON)
```

To enable accounting information collection for all queues that specify the queue attribute ACCTQ as QMGR, use the following MQSC command:

```
ALTER QMGR ACCTQ(ON)
```

MQCONNX options
Use the **`ConnectOpts`** parameter on the MQCONNX call to modify the collection of both MQI and queue accounting information at the connection level by overriding the effective values of the queue manager attributes ACCTMQI and ACCTQ

The **`ConnectOpts`** parameter can have the following values:

MQCNO_ACCOUNTING_MQI_ENABLED
: If the value of the queue manager attribute ACCTMQI is specified as OFF, MQI accounting is enabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as ON.

  If the value of the queue manager attribute ACCTMQI is not specified as OFF, this attribute has no effect.

MQCNO_ACCOUNTING_MQI_DISABLED
: If the value of the queue manager attribute ACCTMQI is specified as ON, MQI accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as OFF.

  If the value of the queue manager attribute ACCTMQI is not specified as ON, this attribute has no effect.

MQCNO_ACCOUNTING_Q_ENABLED
: If the value of the queue manager attribute ACCTQ is specified as OFF, queue accounting is enabled for this connection. All queues with ACCTQ specified as QMGR, are enabled for queue accounting. This is equivalent of the queue manager attribute ACCTQ being specified as ON.

  If the value of the queue manager attribute ACCTQ is not specified as OFF, this attribute has no effect.

MQCNO_ACCOUNTING_Q_DISABLED
: If the value of the queue manager attribute ACCTQ is specified as ON, queue accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTQ being specified as OFF.

If the value of the queue manager attribute ACCTQ is not specified as ON, this attribute has no effect.

These overrides are by disabled by default. To enable them, set the queue manager attribute ACCTCONO to ENABLED. To enable accounting overrides for individual connections use the following MQSC command:

```
ALTER QMGR ACCTCONO(ENABLED)
```

Accounting message generation
Accounting messages are generated when an application disconnects from the queue manager. Intermediate accounting messages are also written for long running IBM MQ applications.

Accounting messages are generated in either of the following ways when an application disconnects:

- The application issues an MQDISC call
- The queue manager recognises that the application has terminated

Intermediate accounting messages are written for long running IBM MQ applications when the interval since the connection was established or since the last intermediate accounting message that was written exceeds the configured interval. The queue manager attribute, ACCTINT, specifies the time, in seconds, after which intermediate accounting messages can be automatically written. Accounting messages are generated only when the application interacts with the queue manager, so applications that remain connected to the queue manager for long periods without executing MQI requests do not generate accounting messages until the execution of the first MQI request following the completion of the accounting interval.

The default accounting interval is 1800 seconds (30 minutes). For example, to change the accounting interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR ACCTINT(900)
```

Statistics messages

Statistics messages record information about the activities occurring in an IBM MQ system. An statistics messages is a PCF message that contains a number of PCF structures.

Statistics messages are delivered to the system queue (SYSTEM.ADMIN.STATISTICS.QUEUE) at configured intervals, whenever there is some activity.

Statistics messages are in the following categories:

MQI statistics messages
MQI statistics messages contain information relating to the number of MQI calls made during a configured interval. For example, the information can include the number of MQI calls issued by a queue manager.

Queue statistics messages
Queue statistics messages contain information relating to the activity of a queue during a configured interval. The information includes the number of messages put on, and retrieved from, the queue, and the total number of bytes processed by a queue.

Each queue statistics message can contain up to 100 records, with each record relating to the activity per queue for which statistics were collected.

Statistics messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the statistics data is recorded against the base queue, and, for a remote queue, the statistics data is recorded against the transmission queue.

Channel statistics messages
Channel statistics messages contain information relating to the activity of a channel during a configured interval. For example the information might be the number of messages transferred by the channel, or the number of bytes transferred by the channel.

Each channel statistics message contains up to 100 records, with each record relating to the activity per channel for which statistics were collected.

Related reference

"MQI statistics information" on page 149
Use the queue manager attribute STATMQI to control the collection of MQI statistics information

"Queue statistics information" on page 150
Use the queue attribute STATQ and the queue manager attribute STATQ to control the collection of queue statistics information

"Channel statistics information" on page 151
Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection.

Statistics messages format

Statistics messages comprise a set of PCF fields that consist of a message descriptor and message data.

Message descriptor

- A statistics message MQMD (message descriptor)

Accounting message data

- A statistics message MQCFH (PCF header)
- Statistics message data that is always returned
- Statistics message data that is returned if available

The statistics message MQCFH (PCF header) contains information about the interval for which the statistics data was recorded.

Statistics message data comprises PCF parameters that store the statistics information. The content of statistics messages depends on the message category as follows:

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *QStatisticsData* PCF groups.

There is one *QStatisticsData* PCF group for every queue was active in the interval. If more than 100 queues were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *ChlStatisticsData* PCF groups.

There is one *ChlStatisticsData* PCF group for every channel that was active in the interval. If more than 100 channels were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Statistics information collection

Use queue, queue manager, and channel attributes to control the collection of statistics information

MQI statistics information
Use the queue manager attribute STATMQI to control the collection of MQI statistics information

To change the value of this attribute, use the MQSC command, `ALTER QMGR` and specify the parameter `STATMQI`. Statistics messages are generated only for queues that are opened after statistics collection has been enabled. The `STATMQI` parameter can have the following values:

ON
MQI statistics information is collected for every connection to the queue manager.

OFF
MQI statistics information is not collected. This is the default value.

For example, to enable MQI statistics information collection use the following MQSC command:

```
ALTER QMGR STATMQI(ON)
```

Queue statistics information
Use the queue attribute STATQ and the queue manager attribute STATQ to control the collection of queue statistics information

You can enable or disable queue statistics information collection for individual queues or for multiple queues. To control individual queues, set the queue attribute STATQ. You enable or disable queue statistics information collection at the queue manager level by using the queue manager attribute STATQ. For all queues that have the queue attribute STATQ specified with the value QMGR, queue statistics information collection is controlled at the queue manager level.

Queue statistics are incremented only for operations using IBM MQ MQI Object Handles that were opened after statistics collection has been enabled.

Queue Statistics messages are generated only for queues for which statistics data has been collected in the previous time period.

The same queue can have several put operations and get operations through several Object Handles. Some Object Handles might have been opened before statistics collection was enabled, but others were opened afterwards. Therefore, it is possible for the queue statistics to record the activity of some put operations and get operations, and not all.

To ensure that the Queue Statistics are recording the activity of all applications, you must close and reopen new Object Handles on the queue, or queues, that you are monitoring. The best way to achieve this, is to end and restart all applications after enabling statistics collection.

To change the value of the queue attribute STATQ, use the MQSC command, `ALTER QLOCAL` and specify the parameter STATQ. The queue attribute STATQ can have the following values:

ON
Queue statistics information is collected for every connection to the queue manager that opens the queue.

OFF
Queue statistics information for this queue is not collected.

QMGR
The collection of queue statistics information for this queue is controlled according to the value of the queue manager attribute, STATQ. This is the default value.

To change the value of the queue manager attribute STATQ, use the MQSC command, `ALTER QMGR` and specify the parameter STATQ. The queue manager attribute STATQ can have the following values:

ON
Queue statistics information is collected for queues that have the queue attribute STATQ set as QMGR

OFF
Queue statistics information is not collected for queues that have the queue attribute STATQ set as QMGR. This is the default value.

NONE
The collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

If the queue manager attribute STATQ is set to NONE, the collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

For example, to enable statistics information collection for the queue, Q1, use the following MQSC command:

```
ALTER QLOCAL(Q1) STATQ(ON)
```

To enable statistics information collection for all queues that specify the queue attribute STATQ as QMGR, use the following MQSC command:

```
ALTER QMGR STATQ(ON)
```

IBM i distributed *Channel statistics information*

Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection.

You can enable or disable channel statistics information collection for individual channels, or for multiple channels. To control individual channels, you must set the channel attribute STATCHL to enable or disable channel statistic information collection. To control many channels together, you enable or disable channel accounting information collection at the queue manager level by using the queue manager attribute STATCHL. For all channels that have the channel attribute STATCHL specified with the value QMGR, channel accounting information collection is controlled at the queue manager level.

Automatically defined cluster-sender channels are not IBM MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute STATACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel statistics information collection.

You can set channel statistics information collection to one of the three monitoring levels: low, medium or high. You can set the monitoring level at either object level or at the queue manager level. The choice of which level to use is dependent on your system. Collecting statistics information data might require some instructions that are relatively expensive computationally, so to reduce the impact of channel statistics information collection, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 22 on page 151 summarizes the levels available with channel statistics information collection:

Table 22. Detail level of channel statistics information collection

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

To change the value of the channel attribute STATCHL, use the MQSC command, `ALTER CHANNEL` and specify the parameter STATCHL.

To change the value of the queue manager attribute STATCHL, use the MQSC command, `ALTER QMGR` and specify the parameter STATCHL.

To change the value of the queue manager attribute STATACLS, use the MQSC command, `ALTER QMGR` and specify the parameter STATACLS.

The channel attribute, STATCHL, can have the following values:

LOW
: Channel statistics information is collected with a low level of detail.

MEDIUM
: Channel statistics information is collected with a medium level of detail.

HIGH
: Channel statistics information is collected with a high level of detail.

OFF
: Channel statistics information is not collected for this channel.

QMGR
: The channel attribute is set as QMGR. The collection of statistics information for this channel is controlled by the value of the queue manager attribute, STATCHL.

  This is the default value.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

The queue manager attribute, STATCHL, can have the following values:

LOW
: Channel statistics information is collected with a low level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

MEDIUM
: Channel statistics information is collected with a medium level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

HIGH
: Channel statistics information is collected with a high level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

OFF
: Channel statistics information is not collected for all channels that have the channel attribute STATCHL set as QMGR.

  This is the default value.

NONE
: The collection of channel statistics information is disabled for all channel, regardless of the channel attribute STATCHL.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

The queue manager attribute, STATACLS, can have the following values:

LOW
: Statistics information is collected with a low level of detail for automatically defined cluster-sender channels.

MEDIUM
: Statistics information is collected with a medium level of detail for automatically defined cluster-sender channels.

HIGH
: Statistics information is collected with a high level of detail for automatically defined cluster-sender channels.

OFF
: Statistics information is not collected for automatically defined cluster-sender channels.

QMGR
: The collection of statistics information for automatically defined cluster-sender channels is controlled by the value of the queue manager attribute, STATCHL.

  This is the default value.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

For example, to enable statistics information collection, with a medium level of detail, for the sender channel `QM1.TO.QM2`, use the following MQSC command:

```
ALTER CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all channels that specify the channel attribute STATCHL as QMGR, use the following MQSC command:

```
ALTER QMGR STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all automatically defined cluster-sender channels, use the following MQSC command:

```
ALTER QMGR STATACLS(MEDIUM)
```

Statistics message generation
Statistics messages are generated at configured intervals, and when a queue manager shuts down in a controlled fashion.

The configured interval is controlled by the STATINT queue manager attribute, which specifies the interval, in seconds, between the generation of statistics messages. The default statistics interval is 1800 seconds (30 minutes). To change the statistics interval, use the MQSC command `ALTER QMGR` and specify the `STATINT` parameter. For example, to change the statistics interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR STATINT(900)
```

To write the currently collected statistics data to the statistics queue before the statistics collection interval is due to expire, use the MQSC command `RESET QMGR TYPE(STATISTICS)`. Issuing this command causes the collected statistics data to be written to the statistics queue and a new statistics data collection interval to begin.

Displaying accounting and statistics information

To use the information recorded in accounting and statistics messages, run an application such as the **amqsmon** sample program to transform the recorded information into a suitable format

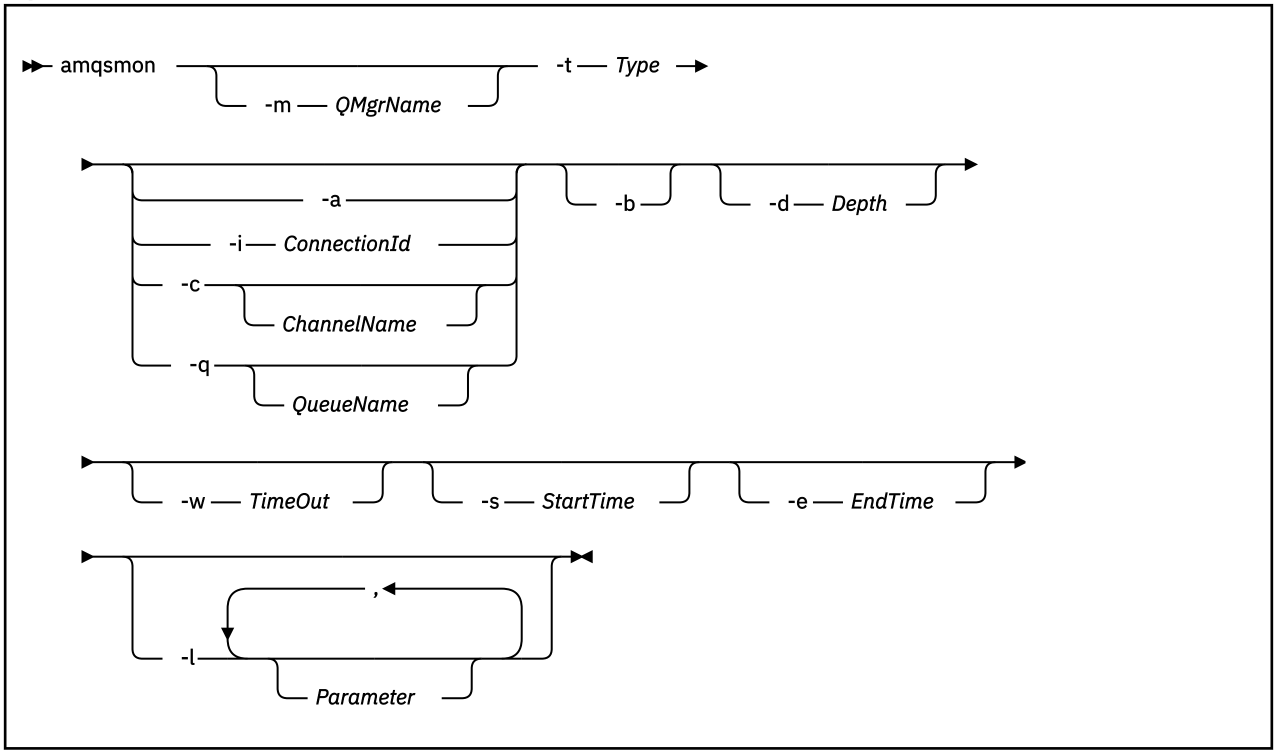
Accounting and statistics messages are written to the system accounting and statistics queues. **amqsmon** is a sample program supplied with IBM MQ that processes messages from the accounting and statistics queues and displays the information to the screen in a readable form.

Because **amqsmon** is a sample program, you can use the supplied source code as template for writing your own application to process accounting or statistics messages, or modify the **amqsmon** source code to meet your own particular requirements.

amqsmon (Display formatted monitoring information)

Use the **amqsmon** sample program to display in a readable format the information contained within accounting and statistics messages. The **amqsmon** program reads accounting messages from the accounting queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE. and reads statistics messages from the statistics queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Syntax



Required parameters

-t *Type*
The type of messages to process. Specify *Type* as one of the following:

accounting
Accounting records are processed. Messages are read from the system queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE.

statistics
Statistics records are processed. Messages are read from the system queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Optional Parameters

-m *QMgrName*
The name of the queue manager from which accounting or statistics messages are to be processed.

If you do not specify this parameter, the default queue manager is used.

-a
Process messages containing MQI records only.

Only display MQI records. Messages not containing MQI records will always be left on the queue they were read from.

-q *QueueName*
QueueName is an optional parameter.

If *QueueName* is not supplied:	Displays queue accounting and queue statistics records only.

If *QueueName* is supplied:	Displays queue accounting and queue statistics records for the queue specified by *QueueName* only. If `-b` is not specified then the accounting and statistics messages from which the records came are discarded. Since accounting and statistics messages can also contain records from other queues, if `-b` is not specified then unseen records can be discarded.

-c *ChannelName*
ChannelName is an optional parameter.

If *ChannelName* is not supplied:	Displays channel statistics records only.
If *ChannelName* is supplied:	Displays channel statistics records for the channel specified by *ChannelName* only. If `-b` is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if `-b` is not specified then unseen records can be discarded.

This parameter is available when displaying statistics messages only, (*-t statistics*).

-i *ConnectionId*
Displays records related to the connection identifier specified by *ConnectionId* only.

This parameter is available when displaying accounting messages only, (*-t accounting*).

If `-b` is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if `-b` is not specified then unseen records can be discarded.

-b
Browse messages.

Messages are retrieved non-destructively.

-d *Depth*
The maximum number of messages that can be processed.

If you do not specify this parameter, then an unlimited number of messages can be processed.

-w *TimeOut*
Time maximum number of seconds to wait for a message to become available.

If you do not specify this parameter, `amqsmon` will end once there are no more messages to process.

-s *StartTime*
Process messages put after the specified *StartTime* only.

StartTime is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `00.00.00` on the date specified. Times are in GMT.

For the effect of not specifying this parameter, see Note 1.

-e *EndTime*
Process messages put before the specified *EndTime* only.

The *EndTime* is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `23.59.59` on the date specified. Times are in GMT.

For the effect of not specifying this parameter, see Note 1.

-l *Parameter*

Only display the selected fields from the records processed. *Parameter* is a comma-separated list of integer values, with each integer value mapping to the numeric constant of a field, see amqsmon example 5.

If you do not specify this parameter, then all available fields are displayed.

Note:

1. If you do not specify *-s StartTime* or *-e EndTime*, the messages that can be processed are not restricted by put time.

amqsmon (Display formatted monitoring information) examples

Use this page to view examples of running the amqsmon (Display formatted monitoring information) sample program

1. See "Queue statistics message data" on page 196 for an explanation of the attributes.

   The following command displays all MQI statistics messages from queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t statistics -a
   ```

   The output from this command follows:

   ```
      RecordType: MQIStatistics
      QueueManager: 'saturn.queue.manager'
      IntervalStartDate: '2005-04-30'
      IntervalStartTime: '15.09.02'
      IntervalEndDate: '2005-04-30'
      IntervalEndTime: '15.39.02'
      CommandLevel: 600
      ConnCount: 23
      ConnFailCount: 0
      ConnsMax: 8
      DiscCount: [17, 0, 0]
      OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
      OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      CloseCount: [0, 73, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
      CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      InqCount: [4, 2102, 0, 0, 0, 46, 0, 0, 0, 0, 0, 0, 0]
      InqFailCount: [0, 31, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      SetCount: [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
      SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      PutCount: [26, 1]
      PutFailCount: 0
      Put1Count: [40, 0]
      Put1FailCount: 0
      PutBytes: [57064, 12320]
      GetCount: [18, 1]
      GetBytes: [52, 12320]
      GetFailCount: 2254
      BrowseCount: [18, 60]
      BrowseBytes: [23784, 30760]
      BrowseFailCount: 9
      CommitCount: 0
      CommitFailCount: 0
      BackCount: 0
      ExpiredMsgCount: 0
      PurgeCount: 0
   ```

2. The following command displays all queue statistics messages for queue `LOCALQ` on queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t statistics -q LOCALQ
   ```

   The output from this command follows:

   ```
      RecordType: QueueStatistics
   ```

```
    QueueManager: 'saturn.queue.manager'
    IntervalStartDate: '2005-04-30'
    IntervalStartTime: '15.09.02'
    IntervalEndDate: '2005-04-30'
    IntervalEndTime: '15.39.02'
    CommandLevel: 600
    ObjectCount: 3
    QueueStatistics:
      QueueName: 'LOCALQ'
      CreateDate: '2005-03-08'
      CreateTime: '17.07.02'
      QueueType: Predefined
      QueueDefinitionType: Local
      QMinDepth: 0
      QMaxDepth: 18
      AverageQueueTime: [29827281, 0]
      PutCount: [26, 0]
      PutFailCount: 0
      Put1Count: [0, 0]
      Put1FailCount: 0
      PutBytes: [88, 0]
      GetCount: [18, 0]
      GetBytes: [52, 0]
      GetFailCount: 0
      BrowseCount: [0, 0]
      BrowseBytes: [0, 0]
      BrowseFailCount: 1
      NonQueuedMsgCount: 0
      ExpiredMsgCount: 0
      PurgedMsgCount: 0
```

3. The following command displays all of the statistics messages recorded since 15:30 on 30 April 2005 from queue manager `saturn.queue.manager`.

   ```
   amqsmon -m saturn.queue.manager -t statistics -s "2005-04-30 15.30.00"
   ```

   The output from this command follows:

   ```
       RecordType: MQIStatistics
       QueueManager: 'saturn.queue.manager'
       IntervalStartDate: '2005-04-30'
       IntervalStartTime: '15.09.02'
       IntervalEndDate: '2005-04-30'
       IntervalEndTime: '15.39.02'
       CommandLevel: 600
       ConnCount: 23
       ConnFailCount: 0
       ConnHighwater: 8
       DiscCount: [17, 0, 0]
       OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
         ...
       RecordType: QueueStatistics
       QueueManager: 'saturn.queue.manager'
       IntervalStartDate: '2005-04-30'
       IntervalStartTime: '15.09.02'
       IntervalEndDate: '2005-04-30'
       IntervalEndTime: '15.39.02'
       CommandLevel: 600
       ObjectCount: 3
       QueueStatistics: 0
         QueueName: 'LOCALQ'
         CreateDate: '2005-03-08'
         CreateTime: '17.07.02'
         QueueType: Predefined
           ...
       QueueStatistics: 1
         QueueName: 'SAMPLEQ'
         CreateDate: '2005-03-08'
         CreateTime: '17.07.02'
         QueueType: Predefined
           ...
   ```

4. See "Queue accounting message data" on page 174 for an explanation of the attributes.

The following command displays all accounting messages recorded on 30 April 2005 from queue manager `saturn.queue.manager`:

```
amqsmon -m saturn.queue.manager -t accounting -s "2005-04-30" -e "2005-04-30"
```

The output from this command follows:

```
RecordType: MQIAccounting
QueueManager: 'saturn.queue.manager'
IntervalStartDate: '2005-04-30'
IntervalStartTime: '15.09.29'
IntervalEndDate: '2005-04-30'
IntervalEndTime: '15.09.30'
CommandLevel: 600
ConnectionId: x'414d5143545245563120202020202020208d0b3742010a0020'
SeqNumber: 0
ApplicationName: 'amqsput'
ApplicationPid: 8572
ApplicationTid: 1
UserId: 'admin'
ConnDate: '2005-03-16'
ConnTime: '15.09.29'
DiscDate: '2005-03-16'
DiscTime: '15.09.30'
DiscType: Normal
OpenCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
CloseCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
PutCount: [1, 0]
PutFailCount: 0
PutBytes: [4, 0]
GetCount: [0, 0]
GetFailCount: 0
GetBytes: [0, 0]
BrowseCount: [0, 0]
BrowseFailCount: 0
BrowseBytes: [0, 0]
CommitCount: 0
CommitFailCount: 0
BackCount: 0
InqCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
InqFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
SetCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

RecordType: MQIAccounting
QueueManager: 'saturn.queue.manager'
IntervalStartDate: '2005-03-16'
IntervalStartTime: '15.16.22'
IntervalEndDate: '2005-03-16'
IntervalEndTime: '15.16.22'
CommandLevel: 600
ConnectionId: x'414d5143545245563120202020202020208d0b3742010c0020'
SeqNumber: 0
ApplicationName: 'runmqsc'
ApplicationPid: 8615
ApplicationTid: 1
    ...
```

5. The following command browses the accounting queue and displays the application name and connection identifier of every application for which MQI accounting information is available:

```
amqsmon -m saturn.queue.manager -t accounting -b -l 7006,3024
```

The output from this command follows:

```
MonitoringType: QueueAccounting
ConnectionId: x'414d5143514d39303520202020202020fcf1855e01e80322'
ApplicationName: 'WebSphere MQ\bin\amqsput.exe'
QueueAccounting: 0

MonitoringType: QueueAccounting
```

```
ConnectionId: x'414d5143514d39303520202020202020fcf1855e01ea0322'
ApplicationName: 'BM\MQ_4\bin64\MQExplorer.exe'
QueueAccounting: 0
QueueAccounting: 1
QueueAccounting: 2
QueueAccounting: 3
QueueAccounting: 4
QueueAccounting: 5
QueueAccounting: 6
QueueAccounting: 7
QueueAccounting: 8
QueueAccounting: 9

MonitoringType: QueueAccounting
ConnectionId: x'414d5143514d39303520202020202020fcf1855e01e90322'
ApplicationName: 's\IBM\MQ_4\bin64\amqsput.exe'
QueueAccounting: 0

MonitoringType: QueueAccounting
ConnectionId: x'414d5143514d39303520202020202020fcf1855e01ef0322'
ApplicationName: 'BM\MQ_4\bin64\MQExplorer.exe'
QueueAccounting: 0
QueueAccounting: 1
QueueAccounting: 2
QueueAccounting: 3
QueueAccounting: 4
QueueAccounting: 5
QueueAccounting: 6
QueueAccounting: 7
QueueAccounting: 8
QueueAccounting: 9

MonitoringType: QueueAccounting
ConnectionId: x'414d5143514d39303520202020202020fcf1855e01e60322'
ApplicationName: 's\IBM\MQ_4\bin64\runmqsc.exe'
QueueAccounting: 0

5 Records Processed
```

See “Finding the mapping for ApplicationName (3024) and ConnectionId (7006)” on page 159 for details on how you locate the variables used in this topic.

See “Finding the mapping for AvgTimeOnQ (703) and QmaxDepth (739)” on page 160 for details on how you locate variables if you selected *statistics* for the **Type** parameter in the **amqsmon** command.

*Finding the mapping for **ApplicationName** (3024) and **ConnectionId** (7006)*

Use this topic to explain how the variables used in the example are found, for monitoring accounting data.

Summary:

- 3024 means MQCACF_APPL_NAME which is the **ApplicationName**
- 7006 means MQBACF_CONNECTION_ID which is the **ConnectionId**

To find out the mapping you need to carry out two procedures:

1. Visit “Queue accounting message data” on page 174 for an explanation of the attributes “ApplicationName” on page 175 and “ConnectionId” on page 175 as you selected *accounting* for the **Type** parameter in the **amqsmon** command.

   In each case, look for the attribute **Identifier**.

   The **Identifier** for **ApplicationName** is MQCACF_APPL_NAME, and for **ConnectionId** is MQBACF_CONNECTION_ID

2. Search for the identifiers you found in Step “1” on page 159.

   Go to the Constants section and scroll down until you find the MQCACF_* (Command format Character Parameter Types) list. Locate MQCACF_APPL_NAME and you see the value 3024.

   Similarly, find the MQBACF_* (Command format Byte Parameter Types)list. Locate MQBACF_CONNECTION_ID and you see the value 7006.

Finding the mapping for AvgTimeOnQ (703) and QmaxDepth (739)
Use this topic to explain how the variables used in the example are found for monitoring statistics data.

Summary:

- 703 means MQIAMO64_AVG_Q_TIME which is the **AvgTimeOnQ**
- 739 means MQIAMO_Q_MAX_DEPTH which is the **QMaxDepth**

To find out the mapping you need to carry out two procedures:

1. Visit "Queue statistics message data" on page 196 for an explanation of the attributes "AvgTimeOnQ" on page 199 and "QMaxDepth" on page 199 as you selected *statistics* for the **Type** parameter in the **amqsmon** command.

   In each case, look for the attribute **Identifier**.

   The **Identifier** for **AvgTimeOnQ** is MQIAMO64_AVG_Q_TIME and for . **QMaxDepth** is MQIAMO_Q_MAX_DEPTH.
2. Search for the identifiers you found in Step "1" on page 160.

   Go to the Constants section and scroll down until you find the MQIAMO_* (Command format Integer Monitoring Parameter Types) list. Locate MQIAMO_Q_MAX_DEPTH and you see the value 739.

   Similarly, find the MQIAMO64_* (Command format 64-bit Integer Monitoring Parameter Types) list. Locate MQIAMO64_AVG_Q_TIME and you see the value 703.

Accounting and statistics message reference

Use this page to obtain an overview of the format of accounting and statistics messages and the information returned in these messages

Accounting and statistics message messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by IBM MQ applications, or information about the activities occurring in an IBM MQ system.

Message descriptor

- An MQMD structure

Message data

- A PCF header (MQCFH)
- Accounting or statistics message data that is always returned
- Accounting or statistics message data that is returned if available

Accounting and statistics message format

Use this page as an example of the structure of an MQI accounting message

Table 23. MQI accounting message structure

MQMD structure	**Accounting message header MQCFH structure**	**MQI accounting message data** [1]
Structure identifier	Structure type	Queue manager
Structure version	Structure length	Interval start date
Report options	Structure version	Interval start time
Message type	Command identifier	Interval end date
Expiration time	Message sequence number	Interval end time
Feedback code	Control options	Command level
Encoding	Completion code	Connection identifier
Coded character set ID	Reason code	Sequence number
Message format	Parameter count	Application name
Message priority		Application process identifier
Persistence		Application thread identifier
Message identifier		User identifier
Correlation identifier		Connection date
Backout count		Connection time
Reply-to queue		Connection name
Reply-to queue manager		Channel name
User identifier		Disconnect date
Accounting token		Disconnect time
Application identity data		Disconnect type
Application type		Open count
Application name		Open fail count
Put date		Close count
Put time		Close fail count
Application origin data		Put count
Group identifier		Put fail count
Message sequence number		Put1 count
Offset		Put1 fail count
Message flags		Put bytes
Original length		Get count
		Get fail count
		Get bytes
		Browse count
		Browse fail count
		Browse bytes
		Commit count
		Commit fail count
		Backout count
		Inquire count
		Inquire fail count
		Set count
		Set fail count

Note:

1. The parameters shown are those returned for an MQI accounting message. The actual accounting or statistics message data depends on the message category.

Accounting and statistics message MQMD (message descriptor)

Use this page to understand the differences between the message descriptor of accounting and statistics messages and the message descriptor of event messages

The parameters and values in the message descriptor of accounting and statistics message are the same as in the message descriptor of event messages, with the following exception:

Format

Description: Format name of message data.

Data type: MQCHAR8.

Value:

MQFMT_ADMIN
Admin message.

Some of the parameters contained in the message descriptor of accounting and statistics message contain fixed data supplied by the queue manager that generated the message.

The MQMD also specifies the name of the queue manager (truncated to 28 characters) that put the message, and the date and time when the message was put on the accounting, or statistics, queue.

Message data in accounting and statistics messages

The message data in accounting and statistics messages is based on the programmable command format (PCF), which is used in PCF command inquiries and responses. The message data in accounting and statistics messages consists of a PCF header (MQCFH) and an accounting or statistics report.

Accounting and statistics message MQCFH (PCF header)

The message header of accounting and statistics messages is an MQCFH structure. The parameters and values in the message header of accounting and statistics message are the same as in the message header of event messages, with the following exceptions:

Command

Description: Command identifier. This identifies the accounting or statistics message category.

Data type: MQLONG.

Values:

MQCMD_ACCOUNTING_MQI
MQI accounting message.

MQCMD_ACCOUNTING_Q
Queue accounting message.

MQCMD_STATISTICS_MQI
MQI statistics message.

MQCMD_STATISTICS_Q
Queue statistics message.

MQCMD_STATISTICS_CHANNEL
Channel statistics message.

Version

Description: Structure version number.

Data type: MQLONG.

Value:

MQCFH_VERSION_3
Version-3 for accounting and statistics messages.

Accounting and statistics message data

The content of accounting and statistics message data is dependent on the category of the accounting or statistics message, as follows:

MQI accounting message
MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and in the range 1 through 100 *QAccountingData* PCF groups.

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *QStatisticsData* PCF groups.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *ChlStatisticsData* PCF groups.

MQI accounting message data

Use this page to view the structure of an MQI accounting message

Message name:	MQI accounting message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN
Returned:	Always

ConnectionId

Description:	The connection identifier for the IBM MQ connection
Identifier:	MQBACF_CONNECTION_ID
Data type:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH
Returned:	Always

SeqNumber

Description:	The sequence number. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Data type:	MQCFIN
Returned:	Always

ApplicationName

Description:	The name of the application. The contents of this field are equivalent to the contents of the *PutApplName* field in the message descriptor.
Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Always

ApplicationPid

Description:	The operating system process identifier of the application
Identifier:	MQIACF_PROCESS_ID
Data type:	MQCFIN
Returned:	Always

ApplicationTid

Description:	The IBM MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Data type:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ConnDate

Description:	Date of MQCONN operation
Identifier:	MQCAMO_CONN_DATE
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnTime

Description:	Time of MQCONN operation
Identifier:	MQCAMO_CONN_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnName

Description:	Connection name for client connection
Identifier:	MQCACH_CONNECTION_NAME
Data type:	MQCFST
Maximum length:	MQ_CONN_NAME_LENGTH
Returned:	When available

ChannelName

Description: Channel name for client connection

Identifier: MQCACH_CHANNEL_NAME

Data type: MQCFST

Maximum length: MQ_CHANNEL_NAME_LENGTH

Returned: When available

DiscDate

Description: Date of MQDISC operation

Identifier: MQCAMO_DISC_DATE

Data type: MQCFST

Maximum length: MQ_DATE_LENGTH

Returned: When available

DiscTime

Description: Time of MQDISC operation

Identifier: MQCAMO_DISC_TIME

Data type: MQCFST

Maximum length: MQ_TIME_LENGTH

Returned: When available

DiscType

Description: Type of disconnect

Identifier: MQIAMO_DISC_TYPE

Data type: MQCFIN

Values: The possible values are:

MQDISCONNECT_NORMAL
Requested by application

MQDISCONNECT_IMPLICIT
Abnormal application termination

MQDISCONNECT_Q_MGR
Connection broken by queue manager

Returned: When available

OpenCount

Description: The number of objects successfully opened, either by directly issuing a call to MQOPEN or by using the MQPUT1 verb. This parameter is an integer list indexed by object type, see Reference note 1.

Identifier: MQIAMO_OPENS

Data type: MQCFIL

Returned: When available

OpenFailCount

Description:	The number of unsuccessful attempts to open an object. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_OPENS_FAILED
Data type:	MQCFIL
Returned:	When available

CloseCount

Description:	The number of objects closed. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES
Data type:	MQCFIL
Returned:	When available

CloseFailCount

Description:	The number of unsuccessful attempts to close an object. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES_FAILED
Data type:	MQCFIL
Returned:	When available

PutCount

Description:	The number persistent and nonpersistent messages successfully put to a queue, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*

Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Returned:	When available

GetCount

Description:	The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GETS
Data type:	MQCFIL
Returned:	When available

GetFailCount

Description:	The number of failed destructive MQGET calls
Identifier:	MQIAMO_GETS_FAILED
Data type:	MQCFIN
Returned:	When available

GetBytes

Description:	Total number of bytes retrieved for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSES

Data type:	MQCFIL
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN
Returned:	When available

BrowseBytes

Description:	Total number of bytes browsed for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Returned:	When available

CommitCount

Description:	The number of successful transactions. This number includes those transactions committed implicitly by the connected application. Commit requests where there is no outstanding work are included in this count.
Identifier:	MQIAMO_COMMITS
Data type:	MQCFIN
Returned:	When available

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction
Identifier:	MQIAMO_COMMITS_FAILED
Data type:	MQCFIN
Returned:	When available

BackCount

Description:	The number of backouts processed, including implicit backouts due to abnormal disconnection
Identifier:	MQIAMO_BACKOUTS
Data type:	MQCFIN
Returned:	When available

InqCount

Description:	The number of successful objects inquired upon. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS
Data type:	MQCFIL
Returned:	When available

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS_FAILED
Data type:	MQCFIL
Returned:	When available

SetCount

Description:	The number of successful MQSET calls. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS
Data type:	MQCFIL
Returned:	When available

SetFailCount

Description:	The number of unsuccessful MQSET calls. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS_FAILED
Data type:	MQCFIL
Returned:	When available

SubCountDur

Description:	The number of succesful subscribe requests which created, altered or resumed durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_DUR
Data type:	MQCFIL
Returned:	When available.

SubCountNDur

Description:	The number of succesful subscribe requests which created, altered or resumed non-durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_NDUR
Data type:	MQCFIL
Returned:	When available.

SubFailCount

Description:	The number of unsuccessful Subscribe requests.
Identifier:	MQIAMO_SUBS_FAILED
Data type:	MQCFIN
Returned:	When available.

UnsubCountDur

Description:	The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_DUR
Data type:	MQCFIL
Returned:	When available.

UnsubCountNDur

Description:	The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_NDUR
Data type:	MQCFIL
Returned:	When available.

UnsubFailCount

Description:	The number of unsuccessful unsubscribe requests.
Identifier:	MQIAMO_UNSUBS_FAILED
Data type:	MQCFIN
Returned:	When available.

SubRqCount

Description:	The number of successful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS
Data type:	MQCFIN
Returned:	When available.

SubRqFailCount

Description:	The number of unsuccessful MQSUB requests.
Identifier:	MQIAMO_SUBRQS_FAILED
Data type:	MQCFIN
Returned:	When available.

CBCount

Description: The number of successful MQCB requests. This is an array of values indexed by the type of operation

0 - A callback was created or altered

1 - A callback was removed

2 - A callback was resumed

3 - A callback was suspended

Identifier: MQIAMO_CBS

Data type: MQCFIN

Returned: When available.

CBFailCount

Description: The number of unsuccessful MQCB requests.

Identifier: MQIAMO_CBS_FAILED

Data type: MQCFIN

Returned: When available.

CtlCount

Description: The number of successful MQCTL requests. This is an array of values indexed by the type of operation

0 - The connection was started

1 - The connection was stopped

2 - The connection was resumed

3 - The connection was suspended

Identifier: MQIAMO_CTLS

Data type: MQCFIL

Returned: When available.

CtlFailCount

Description: The number of unsuccessful MQCTL requests.

Identifier: MQIAMO_CTLS_FAILED

Data type: MQCFIN

Returned: When available.

StatCount

Description: The number of successful MQSTAT requests.

Identifier: MQIAMO_STATS.

Data type: MQCFIN

Returned: When available.

StatFailCount

Description: The number of unsuccessful MQSTAT requests.

Identifier: MQIAMO_STATS_FAILED

Data type: MQCFIN

Returned: When available.

PutTopicCount

Description: The number persistent and nonpersistent messages successfully put to a topic, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2.

Note: Messages put using a queue alias which resolve to a topic are included in this value.

Identifier: MQIAMO_TOPIC_PUTS

Data type: MQCFIL

Returned: When available.

PutTopicFailCount

Description: The number of unsuccessful attempts to put a message to a topic.

Identifier: MQIAMO_TOPIC_PUTS_FAILED

Data type: MQCFIN

Returned: When available.

Put1TopicCount

Description: The number of persistent and nonpersistent messages successfully put to a topic using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.

Note: Messages put using a queue alias which resolve to a topic are included in this value.

Identifier: MQIAMO_TOPIC_PUT1S

Data type: MQCFIL

Returned: When available.

Put1TopicFailCount

Description: The number of unsuccessful attempts to put a message to a topic using MQPUT1 calls.

Identifier: MQIAMO_TOPIC_PUT1S_FAILED

Data type: MQCFIN

Returned: When available.

PutTopicBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages which resolve to a publish operation. This is number of bytes put by the application and not the resultant number of bytes delivered to subscribers. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_TOPIC_PUT_BYTES
Data type:	MQCFIL64
Returned:	When available.

Queue accounting message data

Use this page to view the structure of a queue accounting message

Message name:	Queue accounting message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH

Returned: Always

IntervalEndTime

Description: The time of the end of the monitoring period

Identifier: MQCAMO_END_TIME

Data type: MQCFST

Maximum length: MQ_TIME_LENGTH

Returned: Always

CommandLevel

Description: The queue manager command level

Identifier: MQIA_COMMAND_LEVEL

Data type: MQCFIN

Returned: Always

ConnectionId

Description: The connection identifier for the IBM MQ connection

Identifier: MQBACF_CONNECTION_ID

Data type: MQCFBS

Maximum length: MQ_CONNECTION_ID_LENGTH

Returned: Always

SeqNumber

Description: The sequence number. This value is incremented for each subsequent record for long running connections.

Identifier: MQIACF_SEQUENCE_NUMBER

Data type: MQCFIN

Returned: Always

ApplicationName

Description: The name of the application. The contents of this field are equivalent to the contents of the PutApplName field in the message descriptor.

Identifier: MQCACF_APPL_NAME

Data type: MQCFST

Maximum length: MQ_APPL_NAME_LENGTH

Returned: Always

ApplicationPid

Description: The operating system process identifier of the application

Identifier: MQIACF_PROCESS_ID

Data type: MQCFIN

Returned: Always

ApplicationTid

Description:	The IBM MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Data type:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ObjectCount

Description:	The number of queues accessed in the interval for which accounting data has been recorded. This value is set to the number of *QAccountingData* PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN
Returned:	Always

QAccountingData

Description:	Grouped parameters specifying accounting details for a queue
Identifier:	MQGACF_Q_ACCOUNTING_DATA
Data type:	MQCFGR

Parameters in group:

QName
CreateDate
CreateTime
QType
QDefinitionType
OpenCount
OpenDate
OpenTime
CloseDate
CloseTime
PutCount
PutFailCount
Put1Count
Put1FailCount
PutBytes
PutMinBytes
PutMaxBytes
GetCount
GetFailCount
GetBytes
GetMinBytes
GetMaxBytes
BrowseCount
BrowseFailCount
BrowseBytes
BrowseMinBytes
BrowseMaxBytes
TimeOnQMin
TimeOnQAvg
TimeOnQMax

Returned: Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When available

CreateDate

Description:	The date the queue was created
Identifier:	MQCA_CREATION_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*

Maximum length:	MQ_DATE_LENGTH
Returned:	When available

CreateTime

Description:	The time the queue was created
Identifier:	MQCA_CREATION_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Value:	MQQT_LOCAL
Returned:	When available

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Values:	Possible values are: **MQQDT_PREDEFINED** **MQQDT_PERMANENT_DYNAMIC** **MQQDT_TEMPORARY_DYNAMIC**
Returned:	When available

OpenCount

Description:	The number of times this queue was opened by the application in this interval, either by directly issuing a call to MQOPEN or by using the MQPUT1 verb.
Identifier:	MQIAMO_OPENS
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

OpenDate

Description:	The date the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the date the queue was originally opened.
Identifier:	MQCAMO_OPEN_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

OpenTime

Description:	The time the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the time the queue was originally opened.
Identifier:	MQCAMO_OPEN_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseDate

Description:	The date of the final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseTime

Description:	The time of final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with the exception of MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL

Included in PCF group:	*QAccountingData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message, with the exception of MQPUT1 calls
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The total number of bytes put for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

PutMinBytes

Description:	The smallest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.

Identifier: MQIAMO_PUT_MIN_BYTES

Data type: MQCFIL

Included in PCF group: *QAccountingData*

Returned: When available

PutMaxBytes

Description: The largest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.

Identifier: MQIAMO_PUT_MAX_BYTES

Data type: MQCFIL

Included in PCF group: *QAccountingData*

Returned: When available

GeneratedMsgCount

Description: The number of generated messages. Generated messages are

- Queue Depth Hi Events
- Queue Depth Low Events

Identifier: MQIAMO_GENERATED_MSGS

Data type: MQCFIN

Included in PCF group: *QAccountingData*

Returned: When available

GetCount

Description: The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.

Identifier: MQIAMO_GETS

Data type: MQCFIL

Included in PCF group: *QAccountingData*

Returned: When available

GetFailCount

Description: The number of failed destructive MQGET calls

Identifier: MQIAMO_GETS_FAILED

Data type: MQCFIN

Included in PCF group: *QAccountingData*

Returned: When available

GetBytes

Description:	The number of bytes read in destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMinBytes

Description:	The size of the smallest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GET_MIN_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMaxBytes

Description:	The size of the largest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GET_MAX_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN

Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive MQGET calls that returned persistent messages
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMinBytes

Description:	The size of the smallest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSE_MIN_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMaxBytes

Description:	The size of the largest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSE_MAX_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQMin

Description:	The shortest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_MIN
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQAvg

Description:	The average time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_AVG
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQMax

Description:	The longest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_MAX
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS
Data type:	MQCFIN
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED
Data type:	MQCFIN
Returned:	When available.

MQI statistics message data

Use this page to view the structure of an MQI statistics message

Message name:	MQI statistics message.

Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	Name of the queue manager.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period.
Identifier:	MQCAMO_END_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period.
Identifier:	MQCAMO_END_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.

Data type: MQCFIN.

Returned: Always.

ConnCount

Description: The number of successful connections to the queue manager.

Identifier: MQIAMO_CONNS.

Data type: MQCFIN.

Returned: When available.

ConnFailCount

Description: The number of unsuccessful connection attempts.

Identifier: MQIAMO_CONNS_FAILED.

Data type: MQCFIN.

Returned: When available.

ConnsMax

Description: The maximum number of concurrent connections in the recording interval.

Identifier: MQIAMO_CONNS_MAX.

Data type: MQCFIN.

Returned: When available.

DiscCount

Description: The number of disconnects from the queue manager. This is an integer array, indexed by the following constants:

- MQDISCONNECT_NORMAL
- MQDISCONNECT_IMPLICIT
- MQDISCONNECT_Q_MGR

Identifier: MQIAMO_DISCS.

Data type: MQCFIL.

Returned: When available.

OpenCount

Description: The number of objects successfully opened, either by directly issuing a call to MQOPEN or by using the MQPUT1 verb. This parameter is an integer list indexed by object type, see Reference note 1.

Identifier: MQIAMO_OPENS.

Data type: MQCFIL.

Returned: When available.

OpenFailCount

Description: The number of unsuccessful open object attempts. This parameter is an integer list indexed by object type, see Reference note 1.

Identifier: MQIAMO_OPENS_FAILED.

Data type:	MQCFIL.
Returned:	When available.

CloseCount

Description:	The number of objects successfully closed. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES.
Data type:	MQCFIL.
Returned:	When available.

CloseFailCount

Description:	The number of unsuccessful close object attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES_FAILED.
Data type:	MQCFIL.
Returned:	When available.

InqCount

Description:	The number of objects successfully inquired upon. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS.
Data type:	MQCFIL.
Returned:	When available.

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS_FAILED.
Data type:	MQCFIL.
Returned:	When available.

SetCount

Description:	The number of objects successfully updated (SET). This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS.
Data type:	MQCFIL.
Returned:	When available.

SetFailCount

Description:	The number of unsuccessful SET attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS_FAILED.
Data type:	MQCFIL.

Returned: When available.

PutCount

Description: The number of persistent and nonpersistent messages successfully put to a queue, with the exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2.

Identifier: MQIAMO_PUTS.

Data type: MQCFIL.

Returned: When available.

PutFailCount

Description: The number of unsuccessful put message attempts.

Identifier: MQIAMO_PUTS_FAILED.

Data type: MQCFIN.

Returned: When available.

Put1Count

Description: The number of persistent and nonpersistent messages successfully put to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO_PUT1S.

Data type: MQCFIL.

Returned: When available.

Put1FailCount

Description: The number of unsuccessful attempts to put a persistent and nonpersistent message to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO_PUT1S_FAILED.

Data type: MQCFIL.

Returned: When available.

PutBytes

Description: The number bytes for persistent and nonpersistent messages written in using put requests. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO64_PUT_BYTES.

Data type: MQCFIL64.

Returned: When available.

GetCount

Description: The number of successful destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO_GETS.

Data type: MQCFIL.

Returned: When available.

GetFailCount

Description: The number of unsuccessful destructive get requests.

Identifier: MQIAMO_GETS_FAILED.

Data type: MQCFIN.

Returned: When available.

GetBytes

Description: The number of bytes read in destructive gets requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO64_GET_BYTES.

Data type: MQCFIL64.

Returned: When available.

BrowseCount

Description: The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO_BROWSES.

Data type: MQCFIL.

Returned: When available.

BrowseFailCount

Description: The number of unsuccessful non-destructive get requests.

Identifier: MQIAMO_BROWSES_FAILED.

Data type: MQCFIN.

Returned: When available.

BrowseBytes

Description: The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2

Identifier: MQIAMO64_BROWSE_BYTES.

Data type: MQCFIL64.

Returned: When available.

CommitCount

Description: The number of transactions successfully completed. This number includes transactions committed implicitly by the application disconnecting, and commit requests where there is no outstanding work.

Identifier: MQIAMO_COMMITS.

Data type:	MQCFIN.
Returned:	When available.

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction.
Identifier:	MQIAMO_COMMITS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

BackCount

Description:	The number of backouts processed, including implicit backout upon abnormal disconnect.
Identifier:	MQIAMO_BACKOUTS.
Data type:	MQCFIN.
Returned:	When available.

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired, before they could be retrieved.
Identifier:	MQIAMO_MSGS_EXPIRED.
Data type:	MQCFIN.
Returned:	When available.

PurgeCount

Description:	The number of times the queue has been cleared.
Identifier:	MQIAMO_MSGS_PURGED.
Data type:	MQCFIN.
Returned:	When available.

SubCountDur

Description:	The number of successful Subscribe requests which created, altered or resumed durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_DUR.
Data type:	MQCFIL
Returned:	When available.

SubCountNDur

Description: The number of successful Subscribe requests which created, altered or resumed non-durable subscriptions. This is an array of values indexed by the type of operation

0 = The number of subscriptions created

1 = The number of subscriptions altered

2 = The number of subscriptions resumed

Identifier: MQIAMO_SUBS_NDUR.

Data type: MQCFIL.

Returned: When available.

SubFailCount

Description: The number of unsuccessful Subscribe requests.

Identifier: MQIAMO_SUBS_FAILED.

Data type: MQCFIN.

Returned: When available.

UnsubCountDur

Description: The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation

0 - The subscription was closed but not removed

1 - The subscription was closed and removed

Identifier: MQIAMO_UNSUBS_DUR.

Data type: MQCFIL.

Returned: When available.

UnsubCountNDur

Description: The number of succesful unsubscribe requests for non-durable subscriptions. This is an array of values indexed by the type of operation

0 - The subscription was closed but not removed

1 - The subscription was closed and removed

Identifier: MQIAMO_UNSUBS_NDUR.

Data type: MQCFIL.

Returned: When available.

UnsubFailCount

Description: The number of failed unsubscribe requests.

Identifier: MQIAMO_UNSUBS_FAILED.

Data type: MQCFIN.

Returned: When available.

SubRqCount

Description:	The number of successful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS
Data type:	MQCFIN
Returned:	When available.

SubRqFailCount

Description:	The number of unsuccessful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS.
Data type:	MQCFIL.
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

CtlCount

Description:	The number of successful MQCTL requests. This is an array of values indexed by the type of operation : 0 - The connection was started 1 - The connection was stopped 2 - The connection was resumed 3 - The connection was suspended
Identifier:	MQIAMO_CTLS.
Data type:	MQCFIL.
Returned:	When available.

CtlFailCount

Description: The number of unsuccessful MQCTL requests.

Identifier: MQIAMO_CTLS_FAILED.

Data type: MQCFIN.

Returned: When available.

StatCount

Description: The number of successful MQSTAT requests.

Identifier: MQIAMO_STATS.

Data type: MQCFIN.

Returned: When available.

StatFailCount

Description: The number of unsuccessful MQSTAT requests.

Identifier: MQIAMO_STATS_FAILED.

Data type: MQCFIN.

Returned: When available.

SubCountDurHighWater

Description: The high-water mark on the number of durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE

0 - The high-water mark for all durable subscriptions in the system

1 - The high-water mark for durable application subscriptions (MQSUBTYPE_API)

2 - The high-water mark for durable admin subscription (MQSUBTYPE_ADMIN)

3 - The high-water mark for durable proxy subscriptions (MQSUBTYPE_PROXY)

Identifier: MQIAMO_SUB_DUR_HIGHWATER

Data type: MQCFIL.

Returned: When available.

SubCountDurLowWater

Description: The low-water mark on the number of durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE.

0 - The low-water mark for all durable subscriptions in the system

1 - The low-water mark for durable application subscriptions (MQSUBTYPE_API)

2 - The low-water mark for durable admin subscriptions (MQSUBTYPE_ADMIN)

3 - The low-water mark for durable proxy subscriptions (MQSUBTYPE_PROXY)

Identifier: MQIAMO_SUB_DUR_LOWWATER

Data type: MQCFIL.

Returned: When available.

SubCountNDurHighWater

Description:	The high-water mark on the number of non-durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE 0 - The high-water mark for all non-durable subscriptions in the system 1 - The high-water mark for non-durable application subscriptions (MQSUBTYPE_API) 2 - The high-water mark for non-durable admin subscription (MQSUBTYPE_ADMIN) 3 - The high-water mark for non-durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_NDUR_HIGHWATER
Data type:	MQCFIL.
Returned:	When available.

SubCountNDurLowWater

Description:	The low-water mark on the number of non-durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE. 0 - The low-water mark for all non-durable subscriptions in the system 1 - The low-water mark for non-durable application subscriptions (MQSUBTYPE_API) 2 - The low-water mark for non-durable admin subscriptions (MQSUBTYPE_ADMIN) 3 - The low-water mark for non-durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_NDUR_LOWWATER
Data type:	MQCFIL.
Returned:	When available.

PutTopicCount

Description:	The number persistent and nonpersistent messages successfully put to a topic, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUTS.
Data type:	MQCFIL.
Returned:	When available.

PutTopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic.
Identifier:	MQIAMO_TOPIC_PUTS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

Put1TopicCount

Description:	The number of persistent and nonpersistent messages successfully put to a topic using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUT1S.
Data type:	MQCFIL.
Returned:	When available.

Put1TopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic using MQPUT1 calls.
Identifier:	MQIAMO_TOPIC_PUT1S_FAILED.
Data type:	MQCFIN.
Returned:	When available.

PutTopicBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages which resolve to a publish operation. This is number of bytes put by the application and not the resultant number of bytes delivered to subscribers, see PublishMsgBytes for this value. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_TOPIC_PUT_BYTES.
Data type:	MQCFIL64.
Returned:	When available.

PublishMsgCount

Description:	The number of messages delivered to subscriptions in the time interval. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUBLISH_MSG_COUNT
Data type:	MQCFIL.
Returned:	When available.

PublishMsgBytes

Description:	The number of bytes delivered to subscriptions in the time interval. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUBLISH_MSG_BYTES
Data type:	MQCFIL64.
Returned:	When available.

Queue statistics message data

Use this page to view the structure of a queue statistics message

Message name:	Queue statistics message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	Name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date at the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time at the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN
Returned:	Always

ObjectCount

Description:	The number of queue objects accessed in the interval for which statistics data has been recorded. This value is set to the number of QStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN
Returned:	Always

QStatisticsData

Description:	Grouped parameters specifying statistics details for a queue
Identifier:	MQGACF_Q_STATISTICS_DATA
Data type:	MQCFGR
Parameters in group:	*QName* *CreateDate* *CreateTime* *QType* *QDefinitionType* *QMinDepth* *QMaxDepth* *AvgTimeOnQ* *PutCount* *PutFailCount* *Put1Count* *Put1FailCount* *PutBytes* *GetCount* *GetFailCount* *GetBytes* *BrowseCount* *BrowseFailCount* *BrowseBytes* *NonQueuedMsgCount* *ExpiredMsgCount* *PurgeCount*
Returned:	Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME

Data type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always

CreateDate

Description:	The date when the queue was created
Identifier:	MQCA_CREATION_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

CreateTime

Description:	The time when the queue was created
Identifier:	MQCA_CREATION_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Data type:	MQCFIN
Value:	MQOT_LOCAL
Returned:	Always

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Data type:	MQCFIN
Values:	Possible values are • MQQDT_PREDEFINED • MQQDT_PERMANENT_DYNAMIC • MQQDT_TEMPORARY_DYNAMIC
Returned:	When available

QMinDepth

Description:	The minimum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MIN_DEPTH
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*

Returned:	When available

QMaxDepth

Description:	The maximum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MAX_DEPTH
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

AvgTimeOnQ

Description:	The average latency, in microseconds, of messages destructively retrieved from the queue during the monitoring period. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_AVG_Q_TIME
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message to the queue
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL

Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutBytes

Description:	The number of bytes written in put requests to the queue
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetCount

Description:	The number of successful destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_GETS
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetFailCount

Description:	The number of unsuccessful destructive get requests
Identifier:	MQIAMO_GETS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetBytes

Description:	The number of bytes read in destructive put requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64

Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_BROWSES
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive get requests
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

NonQueuedMsgCount

Description:	The number of messages that bypassed the queue and were transferred directly to a waiting application. Bypassing a queue can only occur in certain circumstances. This number represents how many times IBM MQ was able to bypass the queue, and not the number of times an application was waiting.
Identifier:	MQIAMO_MSGS_NOT_QUEUED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired before they could be retrieved.
Identifier:	MQIAMO_MSGS_EXPIRED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

PurgeCount

Description:	The number of messages purged.
Identifier:	MQIAMO_MSGS_PURGED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

Channel statistics message data

Use this page to view the structure of a channel statistics message

Message name:	Channel statistics message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	The name of the queue manager.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH.

Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.
Data type:	MQCFIN.
Returned:	Always.

ObjectCount

Description:	The number of Channel objects accessed in the interval for which statistics data has been recorded. This value is set to the number of ChlStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN.
Returned:	Always.

ChlStatisticsData

Description:	Grouped parameters specifying statistics details for a channel.
Identifier:	MQGACF_CHL_STATISTICS_DATA.
Data type:	MQCFGR.

Parameters in group:	*ChannelName* *ChannelType* *RemoteQmgr* *ConnectionName* *MsgCount* *TotalBytes* *NetTimeMin* *NetTimeAvg* *NetTimeMax* *ExitTimeMin* *ExitTimeAvg* *ExitTimeMax* *FullBatchCount* *IncmplBatchCount* *AverageBatchSize* *PutRetryCount*
Returned:	Always.

ChannelName

Description:	The name of the channel.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The channel type.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.
Values:	Possible values are: **MQCHT_SENDER** Sender channel. **MQCHT_SERVER** Server channel. **MQCHT_RECEIVER** Receiver channel. **MQCHT_REQUESTER** Requester channel. **MQCHT_CLUSRCVR** Cluster receiver channel. **MQCHT_CLUSSDR** Cluster sender channel.
Returned:	Always.

RemoteQmgr

Description:	The name of the remote queue manager.

Identifier: MQCA_REMOTE_Q_MGR_NAME.

Data type: MQCFST.

Maximum length: MQ_Q_MGR_NAME_LENGTH

Returned: When available.

ConnectionName

Description: Connection name of remote queue manager.

Identifier: MQCACH_CONNECTION_NAME.

Data type: MQCFST

Maximum length: MQ_CONN_NAME_LENGTH

Returned: When available.

MsgCount

Description: The number of persistent and nonpersistent messages sent or received.

Identifier: MQIAMO_MSGS.

Data type: MQCFIN

Returned: When available.

TotalBytes

Description: The number of bytes sent or received for persistent and nonpersistent messages.

Identifier: MQIAMO64_BYTES.

Data type: MQCFIN64.

Returned: When available.

NetTimeMin

Description: The shortest recorded channel round trip measured in the recording interval, in microseconds.

Identifier: MQIAMO_NET_TIME_MIN.

Data type: MQCFIN.

Returned: When available.

NetTimeAvg

Description: The average recorded channel round trip measured in the recording interval, in microseconds.

Identifier: MQIAMO_NET_TIME_AVG.

Data type: MQCFIN.

Returned: When available.

NetTimeMax

Description: The longest recorded channel round trip measured in the recording interval, in microseconds.

Identifier: MQIAMO_NET_TIME_MAX.

Data type: MQCFIN.

Returned: When available.

ExitTimeMin

Description: The shortest recorded time, in microseconds, spent executing a user exit in the recording interval,

Identifier: MQIAMO_EXIT_TIME_MIN.

Data type: MQCFIN.

Returned: When available.

ExitTimeAvg

Description: The average recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.

Identifier: MQIAMO_EXIT_TIME_AVG.

Data type: MQCFIN.

Returned: When available.

ExitTimeMax

Description: The longest recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.

Identifier: MQIAMO_EXIT_TIME_MAX.

Data type: MQCFIN.

Returned: When available.

FullBatchCount

Description: The number of batches processed by the channel that were sent because the value of the channel attributes `BATCHSZ` or `BATCHLIM` was reached.

Identifier: MQIAMO_FULL_BATCHES.

Data type: MQCFIN.

Returned: When available.

IncmplBatchCount

Description: The number of batches processed by the channel, that were sent without the value of the channel attribute BATCHSZ being reached.

Identifier: MQIAMO_INCOMPLETE_BATCHES.

Data type: MQCFIN.

Returned: When available.

AverageBatchSize

Description: The average batch size of batches processed by the channel.

Identifier: MQIAMO_AVG_BATCH_SIZE.

Data type: MQCFIN.

Returned: When available.

PutRetryCount

Description:	The number of times in the time interval that a message failed to be put, and entered a retry loop.
Identifier:	MQIAMO_PUT_RETRIES.
Data type:	MQCFIN.
Returned:	When available.

Reference notes

Use this page to view the notes to which descriptions of the structure of accounting and statistics messages refer

The following message data descriptions refer to these notes:

- "MQI accounting message data" on page 163
- "Queue accounting message data" on page 174
- "MQI statistics message data" on page 184
- "Queue statistics message data" on page 196
- "Channel statistics message data" on page 202

1. This parameter relates to IBM MQ objects. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 24. Array indexed by object type

Object type	Value context
MQOT_Q (1)	Contains the value relating to queue objects.
MQOT_NAMELIST (2)	Contains the value relating to namelist objects.
MQOT_PROCESS (3)	Contains the value relating to process objects.
MQOT_Q_MGR (5)	Contains the value relating to queue manager objects.
MQOT_CHANNEL (6)	Contains the value relating to channel objects.
MQOT_AUTH_INFO (7)	Contains the value relating to authentication information objects.
MQOT_TOPIC (8)	Contains the value relating to topic objects.

Note: An array of 13 MQCFIL or MQCFIL64 values are returned but only those listed are meaningful.

2. This parameter relates to IBM MQ messages. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 25. Array indexed by persistence value

Constant	Value
1	Contains the value for nonpersistent messages.
2	Contains the value for persistent messages.

Note: The index for each of these arrays starts at zero, so an index of 1 refers to the second row of the array. Elements of these arrays not listed in these tables contain no accounting or statistics information.

Application activity trace

Application activity trace produces detailed information about the behavior of applications connected to a queue manager. It traces the behavior of an application and provides a detailed view of the parameters used by an application as it interacts with IBM MQ resources. It also shows the sequence of MQI calls issued by an application.

Use Application activity trace when you require more information than is provided by Event monitoring, Message monitoring, Accounting and statistics messages, and Real-time monitoring.

Note that activity trace is not supported by IBM MQ for z/OS.

Collecting application activity trace information

An application activity trace message is a PCF message. You configure activity trace using a configuration file. To collect application activity trace information you set the ACTVTRC queue manager attribute. You can override this setting at connection level using MQCONNX options, or at application stanza level using the activity trace configuration file.

About this task

Activity trace messages are composed of an MQMD structure: a PCF (MQCFH) header structure, followed by a number of PCF parameters. A sequence of ApplicationTraceData PCF groups follows the PCF parameters. These PCF groups collect information about the MQI operations that an application performs while connected to a queue manager. You configure activity trace using a configuration file called `mqat.ini`.

To control whether or not application activity trace information is collected, you configure one or more of the following settings:

1. The ACTVTRC queue manager attribute.
2. The ACTVCONO settings (in the MQCNO structure passed in MQCONNX).
3. The matching stanza for the application in the activity trace configuration file `mqat.ini`.

The previous sequence is significant. The ACTVTRC attribute is overridden by the ACTVCONO settings, which are overridden by the settings in the `mqat.ini` file.

Trace entries are written after each operation has completed, unless otherwise stated. These entries are first written to the system queue SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE, then written to application activity trace messages when the application disconnects from the queue manager. For long running applications, intermediate messages are written if any of the following events occurs:

- The lifetime of the connection reaches a defined timeout value.
- The number of operations reaches a specified number.
- The amount of data collected in memory reaches the maximum message length allowed for the queue.

You set the timeout value using the `ActivityInterval` parameter. You set the number of operations using the `ActivityCount` parameter. Both parameters are specified in the activity trace configuration file `mqat.ini`.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **`ActivityCount`** and the **`ActivityInterval`** settings. See "Tuning the performance impact of application activity trace" on page 215.

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 216.

Procedure

1. "Setting ACTVTRC to control collection of activity trace information" on page 209.
2. "Setting MQCONNX options to control collection of activity trace information" on page 209.

3. “Configuring activity trace behavior using mqat.ini” on page 210.
4. “Tuning the performance impact of application activity trace” on page 215.

Setting ACTVTRC to control collection of activity trace information

Use the queue manager attribute ACTVTRC to control the collection of MQI application activity trace information

About this task

Application activity trace messages are generated only for connections that begin after application activity trace is enabled. The **ACTVTRC** parameter can have the following values:

ON
: API activity trace collection is switched on

OFF
: API activity trace collection is switched off

Note: The **ACTVTRC** setting can be overridden by the queue manager **ACTVCONO** parameter. If you set the **ACTVCONO** parameter to ENABLED, then the **ACTVTRC** setting can be overridden for a given connection using the **Options** field in the MQCNO structure. See“Setting MQCONNX options to control collection of activity trace information” on page 209.

Example

To change the value of the **ACTVTRC** parameter, you use the MQSC command ALTER QMGR. For example, to enable MQI application activity trace information collection use the following MQSC command:

```
ALTER QMGR ACTVTRC(ON)
```

What to do next

The simplest way to view the contents of application activity trace messages is to use the “amqsact sample program” on page 216.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **ActivityCount** and the **ActivityInterval** settings. See “Tuning the performance impact of application activity trace” on page 215.

Setting MQCONNX options to control collection of activity trace information

If the queue manager attribute **ACTVCONO** is set to ENABLED, you can use the **ConnectOpts** parameter on the MQCONNX call to enable or disable application activity reports on a per connection basis. These options override the activity trace behavior defined by the queue manager attribute **ACTVTRC**, and can be overridden by settings in the activity trace configuration file mqat.ini.

Procedure

1. Set the queue manager attribute **ACTVCONO** to ENABLED.

   Note: If an application attempts to modify the accounting behavior of an application using the **ConnectOpts** parameter, and the QMGR attribute **ACTVCONO** is set to DISABLED, then no error is returned to the application, and activity trace collection is defined by the queue manager attributes or the activity trace configuration file mqat.ini.
2. Set the **ConnectOpts** parameter on the MQCONNX call to MQCNO_ ACTIVITY_ TRACE_ENABLED.

   The **ConnectOpts** parameter on the MQCONNX call can have the following values:

   MQCNO_ACTIVITY_ TRACE_DISABLED
   : Activity trace is switched off for the connection.

MQCNO_ ACTIVITY_ TRACE_ENABLED
Activity trace is switched on for the connection.

Note: If an application selects both MQCNO_ ACTIVITY_ TRACE_ENABLED and MQCNO_ACTIVITY_ TRACE_DISABLED for MQCONNX, the call fails with a reason code of MQRC_OPTIONS_ERROR.

3. Check that these activity trace settings are not being overridden by settings in the activity trace configuration file mqat.ini.

   See“Configuring activity trace behavior using mqat.ini” on page 210.

What to do next

The simplest way to view the contents of application activity trace messages is to use the “amqsact sample program” on page 216.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **ActivityCount** and the **ActivityInterval** settings. See “Tuning the performance impact of application activity trace” on page 215.

Configuring activity trace behavior using ***mqat.ini***

Activity trace behavior is configured using a configuration file called mqat.ini. This file is used to define the level and frequency of reporting activity trace data. The file also provides a way to define rules to enable and disable activity trace based on the name of an application.

About this task

Linux UNIX On UNIX and Linux systems, mqat.ini is located in the queue manager data directory, which is the same location as the qm.ini file.

Windows On Windows systems, mqat.ini is located in the queue manager data directory C:\Program Files\IBM\WebSphere MQ\qmgrs\queue_manager_name. Users running applications to be traced need permission to read this file.

Note: Queue managers migrated from IBM WebSphere MQ 7.1 or earlier will have the mqat.ini file missing. In such cases, the mqat.ini file needs to be created manually and 660 permissions need to be set on the file.

When the mqat.ini file is modified, newly created IBM MQ connections will be processed according to the modified version. Existing connections will continue to use the previous version unless the queue manager parameters are altered, for example following an ALTER QMGR command.

This file follows the same stanza key and parameter-value pair format as the mqs.ini and qm.ini files. It is located in the queue manager data directory, this is, the same location as the qm.ini file for the queue manager.

The file consists of a single stanza, **AllActivityTrace**, to configure the level and frequency of reporting activity trace data by default for all activity trace.

The file can also contain multiple **ApplicationTrace** stanzas. Each one of these, defines a rule for the trace behavior for one or more connections, based on matching the application name of the connections to the rule.

AllActivityTrace stanza

A single **AllActivityTrace** stanza defines settings for the activity trace that is applied to all IBM MQ connections, unless overridden.

Individual values in the **AllActivityTrace** stanza can be overridden by more specific information in an **ApplicationTrace stanza** stanza.

If more than one **AllActivityTrace** stanza is specified then the values in the last stanza is used. Parameters missing from the chosen **AllActivityTrace** take default values. Parameters and values from previous **AllActivityTrace** stanzas are ignored.

The following parameters can be specified under the **AllActivityTrace** stanza:

Table 26. Parameter/value pairs that can be used in the activity trace configuration file

Name	Values (default in bold type)	Description
ActivityInterval	0-99999999 (**1**)	Approximate time interval in seconds between trace messages. All activity performed by a connection in that interval will be written in a single message. If this value is 0, the trace message is written when the connection disconnects (or when the activity count is reached).
ActivityCount	0-99999999 (**100**)	Number of MQI or XA operations between trace messages. If this value is 0, the trace message is written when the connection disconnects (or when the activity interval has elapsed).
TraceLevel	LOW / **MEDIUM** / HIGH	Amount of parameter detail traced for each operation. The description of individual operations details which parameters are included for each trace level.
TraceMessageData	**0** - 104 857 600 (maximum 100 MB)	Amount of message data traced in bytes for MQGET, MQPUT, MQPUT1, and Callback operations
StopOnGetTraceMsg	**ON** / OFF	Using activity trace, to trace applications that are also processing activity trace messages, is not advisable due to possible looping occurring.

ApplicationTrace stanza

An ApplicationTrace stanza contains a rule which defines which IBM MQ connections will be traced or not trace based on the application name. Optionally, the default behaviour defined under the Allsettings which override the global trace level and frequency settings.

This stanza can include ApplName, ApplFunction and ApplClass parameters which are used according to the matching rules defined in Connection Matching Rules to determine whether the stanza applies to a particular connection or not.

The stanza must include the Trace parameter to determine if this rule turns activity trace on or off for matching connections.

An off rule can be used to explicitly disable trace for more specific application names and to override the ACTVTRC setting of the queue manager or activity trace connection options.

The following parameters can be specified under the **ApplicationTrace** stanza:

Table 27. Parameter/value pairs that can be used in the application trace configuration file

Name	Values (default in bold type)	Description
Trace	ON / OFF (Required parameter - no default value)	Activity trace switch. This switch can be used in the application-specific stanza to determine whether activity trace is active for the scope of the current application stanza. Note that this value overrides ACTVTRC and ACTVCONO settings for the queue manager.

Table 27. Parameter/value pairs that can be used in the application trace configuration file (continued)		
Name	**Values (default in bold type)**	**Description**
ApplName	Character string (Required parameter - no default)	This value is used to determine which applications the ApplicationTrace stanza applies to. It is matched to the ApplName value from the API exit context structure (which is equivalent to the MQMD.PutApplName). The content of the ApplName value varies according to the application environment. For platforms other than z/OS, only the filename portion of the MQAXC.ApplName is matched to the value in the stanza. Characters to the left of the rightmost path separator are ignored when the comparison is made. z/OS For z/OS applications, the entire MQAXC.ApplName is matched to the value in the stanza. A single wildcard character (*) can be used at the end of the ApplName value to match any number of characters after that point. If the ApplName value is set to a single wildcard character (*) then the ApplName value matches all applications.
IBM i IBM i ApplFunction	Character string (default value *****)	This value is used to qualify which application programs the **ApplicationTrace** stanza and **`ApplName`** value applies to. The stanza is optional, but is only valid for IBM i queue managers. A single wildcard character (*) can be used at the end of the **`ApplName`** value to match any number of characters. For example, an **ApplicationTrace** stanza specifying **`ApplName`** = * and **`ApplFunction`** = *AMQSPUT0* applies to all invocations of the AMQSPUT0 program from any job.
ApplClass	USER / MCA / **ALL**	The class of application. See the following table for an explanation of how the **AppType** values correspond to IBM MQ connections.

The following table shows how the *AppClass* values correspond to the **`APICallerType`** and **`APIEnvironment`** fields in the connection API exit context structure.

Table 28. Appclass values and how they correspond to the APICallerType and APIEnvironment fields			
APPLCLASS	**API Caller Type:**	**API Environment:**	**Description**
USER	MQXACT_EXTERNAL	MQXE_OTHER	Only user applications are traced
MCA	(Any value)	MQXE_MCA MQXE_MCA_CLNTCONN MQXE_MCA_SVRCONN	Clients and channels (amqrmppa)
ALL	(Any value)	(Any value)	All connections are traced

Attention: You must use an **APPLCLASS** of *MCA* for client user applications, as a class of *USER* does not match these.

For example, to trace the **amqsputc** sample application, you could use the following code:

```
ApplicationTrace:
ApplClass=MCA                              # Application type
                                            #   Values:  (USER | MCA | INTERNAL | ALL)
                                            #   Default: USER
ApplName=amqsputc     # Application name (may be wildcarded)
                                            #   (matched to app name without path)
                                            #   Default: *
Trace=ON                                    # Activity trace switch for application
                                            #   Values:  ( ON | OFF )
                                            #   Default: OFF
ActivityInterval=30                          # Time interval between trace messages
                                            #   Values: 0-99999999 (0=off)
                                            #   Default: 0
ActivityCount=1                             # Number of operations between trace msgs
                                            #   Values: 0-99999999  (0=off)
                                            #   Default: 0
TraceLevel=MEDIUM                           # Amount of data traced for each operation
                                            #   Values: LOW | MEDIUM | HIGH
                                            #   Default: MEDIUM
TraceMessageData=1000                       # Amount of message data traced
                                            #   Values: 0-100000000
                                            #   Default: 0
```

The default `mqat.ini` generated when a queue manager is created, contains a single rule to explicitly disable activity trace for the supplied activity trace sample, **amqsact**.

Connection Matching Rules

The queue manager applies the following rules to determine which stanzas settings to use for a connection.

1. A value specified in the **AllActivityTrace** stanza is used for the connection unless the value also occurs in an **ApplicationTrace** stanza and the stanza fulfills the matching criteria for the connection described in points 2, 3, and 4.
2. The **ApplClass** is matched against the type of the IBM MQ connection. If the **ApplClass** does not match the connection type then the stanza is ignored for this connection.
3. The *ApplName* value in the stanza is matched against the file name portion of the **ApplName** field from the API exit context structure (MQAXC) for the connection.

   The file name portion is derived from the characters to the right of the final path separator (/ or \) character. If the stanza **ApplName** includes a wildcard (*) then only the characters to the left of the wildcard are compared with the equivalent number of characters from the **ApplName** of the connection.

   For example, if a stanza value of "FRE*" is specified then only the first three characters are used in the comparison, so "path/FREEDOM" and "path\FREDDY" match, but "path/FRIEND" does not. If the *ApplName* value of the stanza does not match the connection **ApplName**, the stanza is ignored for this connection.
4. If more than one stanza matches the **ApplName** and **ApplClass** of the connection, then the stanza with the most specific **ApplName** is used.

   The most specific **ApplName** is defined as the one that uses the most characters to match the **ApplName** of the connection.

   For example, if the `ini` file contains a stanza with **ApplName** = *"FRE*"* and another stanza with **ApplName** = *"FREE*"* then the stanza with **ApplName** = *"FREE*"* is chosen as the best match for a connection with **ApplName** = *"path/FREEDOM"* because it matches four characters (whereas **ApplName** = *"FRE*"* matches only three).
5. If after applying the rules in points 2, 3, and 4, there is more than one stanza that matches the connections **ApplName** and **ApplClass** of the connection, the values from the last matching will be used and all other stanzas will be ignored.

Overriding default settings for each rule

Optionally, the global trace level and frequency settings under the **AllActivityTrace** stanza can be overridden for those connections matching an **ApplicationTrace** stanza.

The following parameters can be set under an **ApplicationTrace** stanza. If they are not set, the value is inherited from the **AllActivityTrace** stanza settings:

- **`ActivityInterval`**
- **`ActivityCount`**
- **`TraceLevel`**
- **`TraceMessageData`**
- **`StopOnTraceMsg`**

`mqat.ini` syntax

The syntax rules for the format of the `mqat.ini` file are:

- Text beginning with a hash or semicolon is considered to be a comment that extends to the end of the line.
- The first significant (non-comment) line must be a stanza key.
- A stanza key consists of the name of the stanza followed by a colon.
- A parameter-value pair consists of the name of a parameter followed by an equals sign and then the value.
- Only a single parameter-value pair can appear on a line. (A parameter-value must not wrap onto another line).
- Leading and trailing whitespace is ignored. There is no limit on the amount of white space between stanza names, parameter names and values, or parameter/value pairs. Line breaks are significant and not ignored
- The maximum length for any line is 2048 characters
- The stanza keys, parameter names, and constant parameter values are not case-sensitive, but the variable parameter values (*ApplName* and *DebugPath*) are case-sensitive.

Application Activity Trace File Example

The following example shows how the configuration data is specified in the Activity Trace ini file.

```
AllActivityTrace:
ActivityInterval=1
ActivityCount=100
TraceLevel=MEDIUM
TraceMessageData=0
StopOnGetTraceMsg=ON

ApplicationTrace:
ApplName=amqs*
Trace=ON
TraceLevel=HIGH
TraceMessageData=1000

ApplicationTrace:
ApplName=amqsact*
Trace=OFF
```

The above **AllActivityTrace** stanza defines how activity trace will perform by default when enabled, either through ApplicationTrace rules or through the queue manager ACTVTRC attribute or programmatically enabled by an application.

The first **ApplicationTrace** stanza defines a rule that will result in any MQI activity by an application whose name starts with "amqs" being traced. Trace generated for these applications will be of high detail and include up to 1000 bytes of message data. The activity interval and count parameters will be inherited

The second **ApplicationTrace** stanza defines a rule that turns trace off for applications with names starting "amqsact" (the activity trace sample). This rule will override the earlier 'on' rule for the amqsact application, resulting in no trace for that application.

An example is also shipped as a sample called `mqat.ini` in the C samples directory (the same directory as the amqsact.c file). This file can be copied to the queue manager data directory, for queue managers that have been migrated from an earlier release of IBM MQ.

What to do next

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **`ActivityCount`** and the **`ActivityInterval`** settings. See "Tuning the performance impact of application activity trace" on page 215.

Tuning the performance impact of application activity trace

Enabling application activity trace can incur a performance penalty. This can be reduced by only tracing the applications that you need, by increasing the number of applications draining the queue, and by tuning **`ActivityInterval`**, **`ActivityCount`** and **`TraceLevel`** in `mqat.ini`.

About this task

Enabling application activity trace selectively for an application or for all queue manager applications can result in additional messaging activity, and in the queue manager requiring additional storage space. In environments where messaging performance is critical, for example, in high workload applications or where a service level agreement (SLA) requires a minimum response time from the messaging provider, it might not be appropriate to collect application activity trace or it might be necessary to adjust the detail or frequency of trace activity messages that are produced. The preset values of **`ActivityInterval`**, **`ActivityCount`** and **`TraceLevel`** in the `mqat.ini` file give a default balance of detail and performance. However, you can tune these values to meet the precise functional and performance requirements of your system.

Procedure

- Only trace the applications that you need.

  Do this by creating an ApplicationTrace application-specific stanza in `mqat.ini`, or by changing the application to specify `MQCNO_ACTIVITY_TRACE_ENABLED` in the options field on the **`MQCNO`** structure on an MQCONNX call. See"Configuring activity trace behavior using mqat.ini" on page 210and"Setting MQCONNX options to control collection of activity trace information" on page 209.
- Before starting trace, check that at least one application is running and is ready to retrieve the activity trace message data from the `SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE`.
- Keep the queue depth as low as possible, by increasing the number of applications draining the queue.
- Set the **`TraceLevel`** value in the `mqat.ini` file to collect the minimum amount of data required.

  `TraceLevel=LOW` has the lowest impact to messaging performance. See"Configuring activity trace behavior using mqat.ini" on page 210.
- Tune the **`ActivityCount`** and **`ActivityInterval`** values in `mqat.ini`, to adjust how often activity trace messages are generated.

  If you are tracing multiple applications, the activity trace messages might be being produced faster than they can be removed from the `SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE`. However, when you reduce how often activity trace messages are generated, you are also increasing the storage space required by the queue manager and the size of the messages when they are written to the queue.

What to do next

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 216.

amqsact sample program

amqsact formats Application Activity Trace messages for you and is provided with IBM MQ.

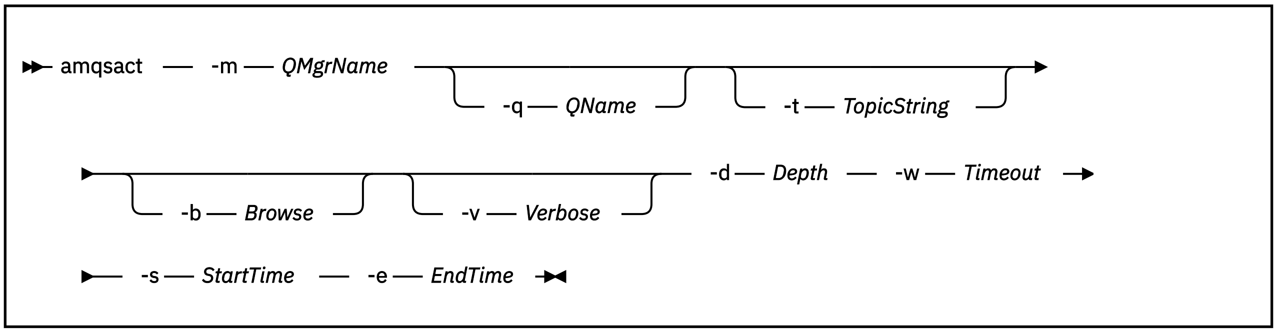
The compiled program is located in the samples directory:

- On Linux and UNIX platforms `MQ_INSTALLATION_PATH/samp/bin`
- On Windows `MQ_INSTALLATION_PATH\tools\c\Samples\Bin`

Display mode

By default, **amqsact** in display mode processes messages on SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. You can override this behavior by specifying a queue name or topic string.

You can also control the trace period displayed and specify whether the activity trace messages are removed or retained after display.



Required parameters

-m *QMgrName*
: Name of the queue manager.

-d *Depth*
: Number of records to display.

-w *Timeout*
: Time to wait, in seconds. If no trace messages appear in the specified period, **amqsact** exits.

-s *StartTime*
: Start time of record to process.

-e *EndTime*
: End time of record to process.

Optional parameters

-q *QName*
: Specify a specific queue to override the default queue name

-t *TopicString*
: Subscribe to an event topic

-b
: Browse records only

-v
: Verbose output

Example output

Use **amqsact** on queue manager *TESTQM*, with verbose output, on an MQCONN API call:

```
amqsact -m TESTQM -v
```

The preceding command gives the following example output:

```
MonitoringType: MQI Activity Trace
Correl_id:
00000000: 414D 5143 5445 5354 514D 2020 2020 2020 'AMQCTESTQM      '
00000010: B5F6 4251 2000 E601                      '        '
QueueManager: 'TESTQM'
Host Name: 'ADMINIB-1VTJ6N1'
IntervalStartDate: '2014-03-15'
IntervalStartTime: '12:08:10'
IntervalEndDate: '2014-03-15'
IntervalEndTime: '12:08:10'
CommandLevel: 750
SeqNumber: 0
ApplicationName: 'IBM MQ_1\bin\amqsput.exe'
Application Type: MQAT_WINDOWS_7
ApplicationPid: 14076
UserId: 'Emma_Bushby'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_OTHER
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 4
Platform: MQPL_WINDOWS_7
MQI Operation: 0
Operation Id: MQXF_CONN
ApplicationTid: 1
OperationDate: '2014-03-15'
OperationTime: '12:08:10'
ConnectionId:
00000000: 414D 5143 5445 5354 514D 2020 2020 2020 'AMQCTESTQM      '
00000010: FFFFFFB5FFFFFFF6 4251 2000 FFFFFFE601  '        '
QueueManager: 'TESTQM'
Completion Code: MQCC_OK
Reason Code: 0
```

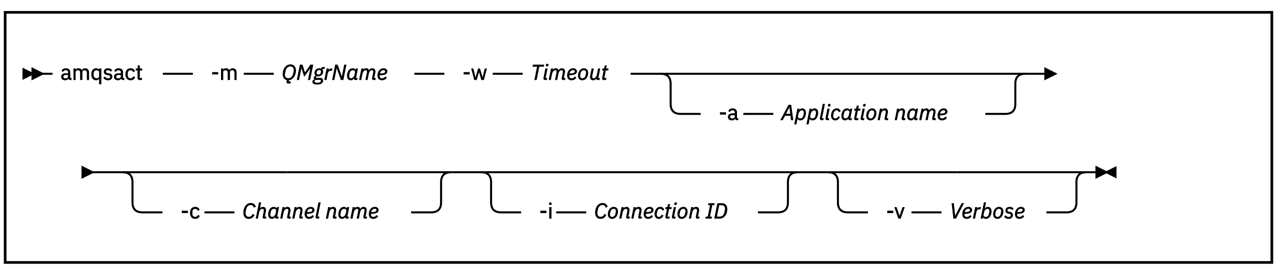
Dynamic mode

V 8.0.0.2

Dynamic mode applies only when using IBM MQ Appliance.

You enable dynamic mode by specifying an application name, a channel name, or a connection identifier as an argument to **amqsact**. Note that you can use wildcard characters in the name.

In dynamic mode, activity trace data is enabled at the start of the sample by use of a nondurable subscription to a system topic. Collecting activity trace data stops when **amqsact** stops. You must specify a timeout for **amqsact** in dynamic mode. You can run multiple copies of **amqsact** concurrently, with each instance receiving a copy of any activity trace data.



Required parameters

V 8.0.0.2

-m *QMgrName*
: Name of the queue manager.

-w *Timeout*
: Time to wait, in seconds. If no trace messages appear in the specified period, **amqsact** exits.

Optional parameters

V 8.0.0.2

-a ***Application name***
: Specify an application name to collect messages for

-c ***Channel name***
: Specify a channel to collect messages for

-i ***Connection ID***
: Specify a connection to collect messages for.

-v
: Verbose output

Example output

V 8.0.0.2

The following command generates and displays activity trace messages for any connections made by applications that start with the text "amqs". After 30 seconds of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 30 -a amqs*
```

The following command generates and displays activity trace messages for any activity on the QMGR1.TO.QMGR2 channel. After 10 seconds of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 10 -c QMGR1.TO.QMGR2
```

The following command generates and displays verbose activity trace messages for any activity on the existing IBM MQ connection that has a CONN of "6B576B5420000701", and an EXTCONN of "414D5143514D47523120202020202020". After a minute of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 60 -i 414D5143514D475231202020202020206B576B5420000701 -v
```

Application activity trace message reference

Use this page to obtain an overview of the format of application activity trace messages and the information returned in these messages

Application activity trace messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by IBM MQ applications, or information about the activities occurring in an IBM MQ system.

Message descriptor

- An MQMD structure

Message data

- A PCF header (MQCFH)
- Application activity trace message data that is always returned
- Application activity trace message data that is operation-specific

Application activity trace message MQMD (message descriptor)

Use this page to understand the differences between the message descriptor of application activity trace messages and the message descriptor of event messages

The parameters and values in the message descriptor of application activity trace message are the same as in the message descriptor of event messages, with the following exception:

Format

Description:	Format name of message data.
Value:	**MQFMT_ADMIN** Admin message.

CorrelId

Description:	Correlation identifier.
Value:	Initialized with the ConnectionId of the application

MQCFH (PCF Header)

Use this page to view the PCF values contained by the MQCFH structure for an activity trace message

For an activity trace message, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the message.
Data type:	MQLONG.
Value:	MQCFT_APP_ACTIVITY

StrucLength

Description:	Length in bytes of MQCFH structure.
Data type:	MQLONG.
Value:	MQCFH_STRUC_LENGTH

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQCFH_VERSION_3

Command

Description:	Command identifier. This field identifies the category of the message.
Data type:	MQLONG.
Values:	MQCMD_ACTIVITY_TRACE

MsgSeqNumber

Description:	Message sequence number. This field is the sequence number of the message within a group of related messages.
Data type:	MQLONG.
Values:	1

Control

Description:	Control options.
Data type:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Data type:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This field is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	1 or greater

Application activity trace message data

Immediately following the PCF header is a set of parameters describing the time interval for the activity trace. These parameters also indicate the sequence of messages in the event of messages being written. The order and number of fields following the header is not guaranteed, allowing additional information to be added in the future.

Message name:	Activity trace message.
System queue:	SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

QSGName

z/OS

Description:	The name of QSG that the Queue Manager is a member of (z/OS only)
Identifier:	MQCA_QSG_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

HostName

Description:	The host name of the machine the Queue Manager is running on
Identifier:	MQCACF_HOST_NAME
Data type:	MQCFST

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH

CommandLevel

Description:	The IBM MQ command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN

SeqNumber

Description:	The sequence number normally zero. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Data type:	MQCFIN

ApplicationName

Description:	The name of the application. (program name)

Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH

ApplClass

Description:	Type of application that performed the activity. Possible values: MQAT_*
Identifier:	MQIA_APPL_TYPE
Data type:	MQCFIN

ApplicationPid

Description:	The operating system Process ID of the application.
Identifier:	MQIACF_PROCESS_ID
Data type:	MQCFIN

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH

APICallerType

Description:	The type of the application. Possible values: MQXACT_EXTERNAL or MQXACT_INTERNAL
Identifier:	MQIACF_API_CALLER_TYPE
Data type:	MQCFIN

Environment

Description:	The runtime environment of the application. Possible values: MQXE_OTHER MQXE_MCA MQXE_MCA_SVRCONN MQXE_COMMAND_SERVER MQXE_MQSC
Identifier:	MQIACF_API_ENVIRONMENT
Data type:	MQCFIN

Detail

Description:	The detail level that is recorded for the connection. Possible values: 1=LOW 2=MEDIUM 3=HIGH
Identifier:	MQIACF_TRACE_DETAIL
Data type:	MQCFIN

TraceDataLength

Description:	The length of message data (in bytes) that is traced for this connection.
Identifier:	MQIACF_TRACE_DATA_LENGTH
Data type:	MQCFIN

Pointer Size

Description:	The length (in bytes) of pointers on the platform the application is running (to assist in interpretation of binary structures)
Identifier:	MQIACF_POINTER_SIZE
Data type:	MQCFIN

Platform

Description:	The platform on which the queue manager is running. Value is one of the MQPL_* values.
Identifier:	MQIA_PLATFORM
Data type:	MQCFIN

Variable parameters for application activity MQI operations

The application activity data `MQCFGR` structure is followed by the set of PCF parameters which corresponds to the operation being performed . The parameters for each operation are defined in the following section.

The trace level indicates the level of trace granularity that is required for the parameters to be included in the trace. The possible trace level values are:

1. Low

   The parameter is included when "low", "medium" or "high" activity tracing is configured for an application. This setting means that a parameter is always included in the `AppActivityData` group for the operation. This set of parameters is sufficient to trace the MQI calls an application makes, and to see if they are successful.

2. Medium

   The parameter is only included in the `AppActivityData` group for the operation when "medium" or "high" activity tracing is configured for an application. This set of parameters adds information about the resources, for example, queue and topic names used by the application.

3. High

   The parameter is only included in the `AppActivityData` group for the operation when "high" activity tracing is configured for an application. This set of parameters includes memory dumps of the structures passed to the MQI and XA functions. For this reason, it contains more information about the parameters used in MQI and XA calls. The structure memory dumps are shallow copies of the structures. To avoid erroneous attempts to dereference pointers, the pointer values in the structures are set to `NULL`.

   Note: The version of the structure that is dumped is not necessarily identical to the version used by an application. The structure can be modified by an API crossing exit, by the activity trace code, or by the queue manager. A queue manager can modify a structure to a later version, but the queue manager never changes it to an earlier version of the structure. To do so, would risk losing data.

MQBACK
Application has started the MQBACK MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQBEGIN
Application has started the MQBEGIN MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

MQBO

Description:	The MQBEGIN options structure. This parameter is not included if a NULL pointer is used on the MQBEGIN call.
PCF Parameter:	MQBACF_MQBO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQBO structure.

V 8.0.0.2 ***QMgrOpDuration***

Description:
: Approximate API call duration, in microseconds, within the queue manager.

  The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

  Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter:
: MQIAMO64_QMGR_OP_DURATION

Trace level:
: 2

Type
: MQCFIN64

MQCALLBACK
Application has started the MQCALLBACK function

ObjectHandle

Description:
: The object handle

PCF Parameter:
: MQIACF_HOBJ

Trace level:
: 1

Type
: MQCFIN

CallType

Description:
: Why function has been called. One of the MQCBCT_* values

PCF Parameter:
: MQIACF_CALL_TYPE

Trace level:
: 1

Type
: MQCFIN

MsgBuffer

Description:
: Message data.

PCF Parameter:
: MQBACF_MESSAGE_DATA

Trace level:
: 1

Type
: MQCFBS

Length:
: Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:
: Length of the message. (Taken from the DataLength field in the MQCBC structure).

PCF Parameter:
: MQIACF_MSG_LENGTH

Trace level:
: 1

Type
: MQCFIN

HighResTime

Description: Time of operation in microseconds since midnight, January 1st 1970 (UTC)

Note: The accuracy of this timer varies according to platform support for high a resolution timer

PCF Parameter: MQIAMO64_HIGHRES_TIME

Trace level: 2

Type MQCFIN64

ReportOptions

Description: Options for report messages

PCF Parameter: MQIACF_REPORT

Trace level: 2

Type MQCFIN

MsgType

Description: Type of message

PCF Parameter: MQIACF_MSG_TYPE

Trace level: 2

Type MQCFIN

Expiry

Description: Message lifetime

PCF Parameter: MQIACF_EXPIRY

Trace level: 2

Type MQCFIN

Format

Description: Format name of message data

PCF Parameter: MQCACH_FORMAT_NAME

Trace level: 2

Type MQCFST

Length: MQ_FORMAT_LENGTH

Priority

Description: Message priority

PCF Parameter: MQIACF_PRIORITY

Trace level: 2

Type MQCFIN

Persistence

Description: Message persistence

PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

ObjectName

Description:	The name of the opened object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQName

Description:	The local name of the queue from which the message was retrieved.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

ReplyToQueue

Description:	MQ_Q_NAME_LENGTH
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type	MQCFST

ReplyToQMgr

Description:	MQ_Q_MGR_NAME_LENGTH
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR

Trace level:	2
Type	MQCFST

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type	MQCFIN

PutDate

Description:	MQ_PUT_DATE_LENGTH
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type	MQCFST

PutTime

Description:	MQ_PUT_TIME_LENGTH
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type	MQCFST

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL _Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_Name
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type	MQCFST
Length:	MQ_MSG_ID_LENGTH

MsgDescStructure

Description:	The MQMD structure. This parameter is omitted if a version 4 MQGMO was used to request that a Message Handle be returned instead of an MQMD
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQMD structure (actual size is dependent on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQGMO structure (actual size is dependent on structure version)

MQCBContextStructure

Description:	The MQCBC structure.
PCF Parameter:	MQBACF_MQCBC_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQCBC structure (actual size is dependent on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCB
Application has started the manage callback MQI function

CallbackOperation

Description:	The manage callback function operation. Set to one of the MQOP_* values
PCF Parameter:	MQIACF_MQCB_OPERATION
Trace level:	1
Type	MQCFIN

CallbackType

Description:	The type of the callback function (CallbackType field from the MQCBD structure). Set to one of the MQCBT_* values
PCF Parameter:	MQIACF_MQCB_TYPE
Trace level:	1
Type	MQCFIN

CallbackOptions

Description:	The callback options. Set to one of the MQCBDO_* values
PCF Parameter:	MQIACF_MQCB_OPTIONS
Trace level:	1
Type	MQCFIN

CallbackFunction

Description:	The pointer to the callback function if started as a function call.
PCF Parameter:	MQBACF_MQCB_FUNCTION

Trace level: 1

Type MQCFBS

Length: Size of MQPTR

CallbackName

Description: The name of the callback function if started as a dynamically linked program.

PCF Parameter: MQCACF_MQCB_NAME

Trace level: 1

Type MQCFST

Length: Size of MQCHAR128

ObjectHandle

Description: The object handle

PCF Parameter: MQIACF_HOBJ

Trace level: 1

Type MQCFIN

MaxMsgLength

Description: Maximum message length. Set to an integer, or the special value MQCBD_FULL_MSG_LENGTH

PCF Parameter: MQIACH_MAX_MSG_LENGTH

Trace level: 2

Type MQCFIN

CompCode

Description: The completion code indicating the result of the operation

PCF Parameter: MQIACF_COMP_CODE

Trace level: 1

Type MQCFIN

Reason

Description: The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type MQCFIN

ResolvedQName

Description: The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.

PCF Parameter: MQCACF_RESOLVED_LOCAL_Q_NAME

Trace level: 2

Type MQCFST

Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

CallBack DescriptorStructure

Description:	The MQCBD structure. This parameter is omitted if a NULL MQCBC value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQCBD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQCBC structure

MsgDescStructure

Description:	The MQMD structure. The MsgDescStructure parameter is omitted if a NULL MQMD value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQMD structure (actual size depends on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure. This parameter is omitted if a NULL MQGMO value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQGMO structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCLOSE
Application has started the MQCLOSE MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type	MQCFIN

CloseOptions

Description:	Close options
PCF Parameter:	MQIACF_CLOSE_OPTIONS
Trace level:	1
Type	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST

Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCMIT
Application has started the MQCMIT MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCONN and MQCONNX
Application has started the MQCONN or MQCONNX MQI function

ConnectionId

Description:	The Connection ID if available or MQCONNID_NONE if not
PCF Parameter:	MQBACF_CONNECTION_ID
Trace level:	1
Type:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH

QueueManagerName

Description:	The (unresolved) name of the queue manager used in the MQCONN(X) call
PCF Parameter:	MQCA_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

ConnectOptions

Description:	Connect Options Derived from MQCNO_* values **Note:** MQCONNX only

PCF Parameter: MQIACF_CONNECT_OPTIONS

Trace level: 2

Type: MQCFIN

ConnectionOptionsStructure

Description: The MQCNO structure.

Note: MQCONNX only)

PCF Parameter: MQBACF_MQCNO_STRUCT

Trace level: 3

Type: MQCFBS

Maximum length: The length in bytes of the MQCNO structure (actual size depends on structure version)

ChannelDefinitionStructure

Description: The MQCD structure.

Note: Client connections only

PCF Parameter: MQBACF_MQCD_STRUCT

Trace level: 3

Type: MQCFBS

Maximum length: The length in bytes of the MQCD structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description: Approximate API call duration, in microseconds, within the queue manager.

The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter: MQIAMO64_QMGR_OP_DURATION

Trace level: 2

Type MQCFIN64

MQCTL
Application has started the MQCTL MQI function

CompCode

Description: The completion code indicating the result of the operation

PCF Parameter: MQIACF_COMP_CODE

Trace level: 1

Type: MQCFIN

Reason

Description: The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type: MQCFIN

CtlOperation

Description: One of MQOP_* values

PCF Parameter: MQIACF_CTL_OPERATION

Trace level: 1

Type: MQCFIN

V 8.0.0.2 ***QMgrOpDuration***

Description: Approximate API call duration, in microseconds, within the queue manager.

The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter: MQIAMO64_QMGR_OP_DURATION

Trace level: 2

Type MQCFIN64

MQDISC
Application has started the MQDISC MQI function

CompCode

Description: The completion code indicating the result of the operation

PCF Parameter: MQIACF_COMP_CODE

Trace level: 1

Type: MQCFIN

Reason

Description: The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type: MQCFIN

MQGET
Application has started the MQGET MQI function

ObjectHandle

Description: The object handle

PCF Parameter: MQIACF_HOBJ

Trace level: 1

Type: MQCFIN

GetOptions

Description: The get options from MQGMO.Options

PCF Parameter: MQIACF_GET_OPTIONS

Trace level: 1

Type: MQCFIN

CompCode

Description: The completion code indicating the result of the operation

PCF Parameter: MQIACF_COMP_CODE

Trace level: 1

Type: MQCFIN

Reason

Description: The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type: MQCFIN

MsgBuffer

Description: Message data. If TRACEDATA=NONE then this parameter is omitted

PCF Parameter: MQBACF_MESSAGE_DATA

Trace level: 1

Type: MQCFBS

Maximum length: Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. (Included in the trace message as MQIACF_TRACE_DATA_LENGTH).

MsgLength

Description: Length of the message.

PCF Parameter: MQIACF_MSG_LENGTH

Trace level: 1

Type: MQCFIN

HighResTime

Description: Time of operation in microseconds since midnight, January 1 1970 (UTC)

Note: The accuracy of this timer varies according to platform support for high a resolution timer

PCF Parameter: MQIAMO64_HIGHRES_TIME

Trace level: 2

Type: MQCFIN64

BufferLength

Description: Length of the buffer provided by the application

PCF Parameter: MQIACF_BUFFER_LENGTH

Trace level: 2

Type: MQCFIN

ObjectName

Description: The name of the opened object

PCF Parameter: MQCACF_OBJECT_NAME

Trace level: 2

Type: MQCFST

Length: MQ_Q_NAME_LENGTH

ResolvedQName

Description: The local name of the queue from which the message was retrieved.

PCF Parameter: MQCACF_RESOLVED_Q_NAME

Trace level: 2

Type: MQCFST

Maximum length: MQ_Q_NAME_LENGTH

ReportOptions

Description: Message report options

PCF Parameter: MQIACF_REPORT

Trace level: 2

Type: MQCFIN

MsgType

Description: Type of message

PCF Parameter: MQIACF_MSG_TYPE

Trace level: 2

Type: MQCFIN

Expiry

Description: Message lifetime

PCF Parameter: MQIACF_EXPIRY

Trace level: 2

Type: MQCFIN

Format

Description: Format name of message data

PCF Parameter: MQCACH_FORMAT_NAME

Trace level: 2

Type: MQCFST

Maximum length: MQ_FORMAT_LENGTH

Priority

Description: Message priority

PCF Parameter: MQIACF_PRIORITY

Trace level: 2

Type: MQCFIN

Persistence

Description: Message persistence

PCF Parameter: MQIACF_PERSISTENCE

Trace level: 2

Type: MQCFIN

MsgId

Description: Message identifier

PCF Parameter: MQBACF_MSG_ID

Trace level: 2

Type: MQCFBS

Maximum length: MQ_MSG_ID_LENGTH

CorrelId

Description: Correlation identifier

PCF Parameter: MQBACF_CORREL_ID

Trace level: 2

Type: MQCFBS

Maximum length: MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:

PCF Parameter: MQCACF_REPLY_TO_Q

Trace level: 2

Type: MQCFST

Maximum length: MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:

PCF Parameter: MQCACF_REPLY_TO_Q_MGR

Trace level: 2

Type: MQCFST

Maximum length: MQ_Q_MGR_NAME_LENGTH

CodedCharSetId

Description: Character set identifier of message data

PCF Parameter: MQIA_CODED_CHAR_SET_ID

Trace level: 2

Type: MQCFIN

Encoding

Description: Numeric encoding of message data.

PCF Parameter: MQIACF_ENCODING

Trace level: 2

Type: MQCFIN

PutDate

Description:

PCF Parameter: MQCACF_PUT_DATE

Trace level: 2

Type: MQCFST

Maximum length: MQ_PUT_DATE_LENGTH

PutTime

Description:

PCF Parameter: MQCACF_PUT_TIME

Trace level: 2

Type: MQCFST

Maximum length: MQ_PUT_TIME_LENGTH

ResolvedQName

Description: The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.

PCF Parameter: MQCACF_RESOLVED_LOCAL_Q_NAME

Trace level: 2

Type MQCFST

Length: MQ_Q_NAME_LENGTH.

ResObjectString

Description: The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.

PCF Parameter: MQCACF_RESOLVED_OBJECT_STRING

Trace level: 2

Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type:	MQCFST

Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The remote queue manager destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

MsgDescStructure

Description:	The MQMD structure.
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type:	MQCFBS
Maximum length:	The length in bytes of the MQMD structure (actual size depends on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type:	MQCFBS

Maximum length:	The length in bytes of the MQGMO structure (actual size depends on structure version)

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQINQ
Application has started the MQINQ MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SelectorCount

Description:	The count of selectors that are supplied in the Selectors array.
PCF Parameter:	MQIACF_SELECTOR_COUNT
Trace level:	2
Type:	MQCFIN

Selectors

Description:	The list of attributes (integer or character) whose values must be returned by MQINQ.

PCF Parameter: MQIACF_SELECTORS

Trace level: 2

Type: MQCFIL

ResolvedQName

Description: The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.

PCF Parameter: MQCACF_RESOLVED_Q_NAME

Trace level: 2

Type: MQCFST

Maximum length: MQ_Q_NAME_LENGTH

ResObjectString

Description: The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.

PCF Parameter: MQCACF_RESOLVED_OBJECT_STRING

Trace level: 2

Type: MQCFST

Maximum length: Length varies

ResolvedType

Description: The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.

PCF Parameter: MQIACF_RESOLVED_TYPE

Trace level: 2

Type: MQCFIN

IntAttrCount

Description: The number of integer attributes returned by the inquire operation

PCF Parameter: MQIACF_INTATTR_COUNT

Trace level: 3

Type: MQCFIN

IntAttrs

Description: The integer attribute values returned by the inquire operation. This parameter is only present if IntAttrCount is > 0 when MQINQ returns.

PCF Parameter: MQIACF_INT_ATTRS

Trace level: 3

Type: MQCFIL

CharAttrs

Description:	The character attributes returned by the inquire operation. The values are concatenated together. This parameter is only included if CharAttrLength is > 0 when MQINQ returns.
PCF Parameter:	MQCACF_CHAR_ATTRS
Trace level:	3
Type:	MQCFST

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQOPEN
Application has started the MQOPEN MQI function

ObjectType

Description:	The object type passed in MQOT.ObjectType
PCF Parameter:	MQIACF_OBJECT_TYPE
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ObjectQMgrName

Description:	The name of the object queue manager passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

OpenOptions

Description:	Options used to open the object
PCF Parameter:	MQIACF_OPEN_OPTIONS
Trace level:	1
Type:	MQCFIN

AlternateUserId

Description:	Only included if MQOO_ALTERNATE_USER_AUTHORITY is specified
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH

RecsPresent

Description:	The number of object name records present. Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of local queues opened successfully Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT

Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of remote queues opened successfully Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of queues that failed to open Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

DynamicQName

Description:	The dynamic queue name passed as input to the MQOPEN call.
PCF Parameter:	MQCACF_DYNAMIC_Q_NAME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ResolvedLocalQName [1 2]

Description:	Contains the local queue name after name resolution has been carried out. (e.g. for remote queues this will be the name of the transmit queue)
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST
Range:	If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD.ObjectName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQName field.
Maximum length:	MQ_Q_NAME_LENGTH

ResolvedLocalQMgrName [1 2]

Description:	The local queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Range:	Only if MQOD.Version >= MQOD_VERSION_3
Maximum length:	MQ_Q_MGR_NAME_LENGTH

ResolvedQName [1] [2]

Description: The queue name after name resolution has been carried out.

PCF Parameter: MQCACF_RESOLVED_Q_NAME

Trace level: 2

Type: MQCFST

Range: If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD.ObjectName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQName field.

Maximum length: MQ_Q_NAME_LENGTH

ResolvedQMgrName [1] [2]

Description: Contains the queue manager name after name resolution has been carried out. If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD. ObjectQMgrName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQMgrName field.

PCF Parameter: MQCACF_RESOLVED_Q_MGR

Trace level: 2

Type: MQCFST

Maximum length: MQ_Q_MGR_NAME_LENGTH

AlternateSecurityId

Description: Alternative security identifier. Only present if MQOD.Version is equal or greater than MQOD_VERSION_3, MQOO_ALTERNATE_USER_AUTHORITY is specified, and MQOD.AlternateSecurityId is not equal to MQSID_NONE.

PCF Parameter: MQBACF_ALTERNATE_SECURITYID

Trace level: 2

Type: MQCFBS

Maximum length: MQ_SECURITY_ID_LENGTH

ObjectString

Description: Long object name. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD.ObjectString is MQVS_NULL_TERMINATED or greater than zero.

PCF Parameter: MQCACF_OBJECT_STRING

Trace level: 2

Type: MQCFST

Maximum length: Length varies.

SelectionString

Description: Selection string. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD. SelectionString is MQVS_NULL_TERMINATED or greater than zero.

PCF Parameter: MQCACF_SELECTION_STRING

Trace level: 2

Type: MQCFST

Maximum length: Length varies.

ResObjectString

Description: The long object name after the queue manager resolves the name provided in the ObjectName field. Only included for topics and queue aliases that reference a topic object if MQOD.Version is equal or greater than MQOD_VERSION_4 and VSLength is MQVS_NULL_TERMINATED or greater than zero.

PCF Parameter: MQCACF_RESOLVED_OBJECT_STRING

Trace level: 2

Type: MQCFST

Maximum length: Length varies.

ResolvedType

Description: The type of the resolved (base) object being opened. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.

PCF Parameter: MQIACF_RESOLVED_TYPE

Trace level: 2

Type: MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description: Approximate API call duration, in microseconds, within the queue manager.

The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter: MQIAMO64_QMGR_OP_DURATION

Trace level: 2

Type MQCFIN64

Application Activity Distribution List PCF Group Header Structure
If the MQOPEN function opens a distribution list, then the MQOPEN parameters includes one AppActivityDistList PCF group for each of the queues in the distribution list up to the number of structures numbered in RecsPresent. The Ap-pActivityDistList PCF group combines information from the MQOR, and MQRR structures to identify the queue name, and indicate the result of the open operation on the queue. An AppActivityDistList group always starts with the following MQCFGR structure:

[1] This parameter is only included if the object being opened resolves to a queue, and the queue is opened for MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_BROWSE

[2] The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Table 29. AppActivityDistList group MQCFGR structure		
MQCFGR field	**Value**	**Description**
Type	MQCFT_GROUP	
StrucLength	Length in bytes of the MQCFGR structure	
Parameter	MQGACF_APP_DIST_LIST	Distribution list group parameter
ParameterCount	4	The number of parameter structures following the MQCFGR structure that are contained within this group.

ObjectName

Description: The name of a queue in the distribution list MQ_Q_NAME_LENGTH. Only included if MQOR structures are provided.

PCF Parameter: MQCACF_OBJECT_NAME

Trace level: 2

Type: MQCFST

Length: MQ_Q_NAME_LENGTH. Only included if MQOR structures are provided.

ObjectQMgrName

Description: The name of the queue manager on which the queue named in ObjectName is defined.

PCF Parameter: MQCACF_OBJECT_Q_MGR_NAME

Trace level: 2

Type: MQCFST

Length: MQ_Q_MGR_NAME_LENGTH. Only included if MQOR structures are provided.

CompCode

Description: The completion code indicating the result of the open for this object. Only included if MQRR structures are provided and the reason code for the MQOPEN is MQRC_MULTIPLE_REASONS

PCF Parameter: MQIACF_COMP_CODE

Trace level: 2

Type: MQCFIN

Reason

Description: The reason code indicating the result of the open for this object. Only included if MQRR structures are provided and the reason code for the MQOPEN is MQRC_MULTIPLE_REASONS

PCF Parameter: MQIACF_REASON_CODE

Trace level: 2

Type: MQCFIN

MQPUT
Application has started the MQPUT MQI function.

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

PutOptions

Description:	The put options from MQPMO.Options
PCF Parameter:	MQIACF_PUT_OPTIONS
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

MsgBuffer

Description:	Message data.
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type:	MQCFBS
Length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:	Length of the message.
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type:	MQCFIN

RecsPresent

Description:	The number of put message records or response records present. Only included if MQPMO Version >= MQPMO_VERSION_2
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of messages sent successfully to local queues
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of messages sent successfully to remote queues
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of messages that could not be sent
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1st 1970 (UTC) **Note:** The accuracy of this timer varies according to platform support for high a resolution timer.
PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type:	MQCFIN64

ObjectName

Description:	The name of the opened object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQName

Description:	The name of the queue after queue name resolution has been performed.

PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQMgrName

Description:	The queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ResolvedLocalQName [3]

Description:	Contains the local queue name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST

ResolvedLocalQMgrName [3]

Description:	Contains the local queue manager name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ReportOptions

Description:	Message report options
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type:	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type:	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2

Type:	MQCFIN

Format

Description:	Format name of message data
PCF Parameter:	MQCACH_FORMAT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_FORMAT_LENGTH

Priority

Description:	Message priority
PCF Parameter:	MQIACF_PRIORITY
Trace level:	2
Type:	MQCFIN

Persistence

Description:	Message persistence
PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type:	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type:	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type:	MQCFIN

PutDate

Description:	
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

PutTime

Description:	
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description: The put time of the message in the transmission queue header.

Note: Only when Format is MQFMT_XMIT_Q_HEADER

PCF Parameter: MQCACF_XQH_PUT_TIME

Trace level: 2

Type: MQCFST

Length: MQ_PUT_TIME_LENGTH

XmitqPutDate

Description: The put date of the message in the transmission queue header.

Note: Only when Format is MQFMT_XMIT_Q_HEADER

PCF Parameter: MQCACF_XQH_PUT_DATE

Trace level: 2

Type: MQCFST

Length: MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description: The remote queue destination of the message in the transmission queue header.

Note: Only when Format is MQFMT_XMIT_Q_HEADER

PCF Parameter: MQCACF_XQH_REMOTE_Q_NAME

Trace level: 2

Type: MQCFST

Length: MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description: The remote queue manager destination of the message in the transmission queue header.

Note: Only when Format is MQFMT_XMIT_Q_HEADER

PCF Parameter: MQCACF_XQH_REMOTE_Q_MGR

Trace level: 2

Type: MQCFST

Length: MQ_Q_NAME_LENGTH

PutMsgOptsStructure

Description: The MQPMO structure.

PCF Parameter: MQBACF_MQPMO_STRUCT

Trace level: 3

Type: MQCFBS

Length: The length in bytes of the MQPMO structure (actual size depends on structure version)

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQPUT Application Activity Distribution List PCF Group Header Structure

If the MQPUT function is putting to a distribution list, then the MQPUT parameters include one AppActivityDistList PCF group. For each of the queues in the distribution list, see "Application Activity Distribution List PCF Group Header Structure" on page 251. The AppActivityDistList PCF group combines information from the MQPMR, and MQRR structures to identify the PUT parameters, and indicate the result of the PUT operation on each queue. For MQPUT operations the AppActivityDistList group contains some or all of the following parameters (the CompCode and Reason is present if the reason code is MQRC_MULTIPLE_REASONS and the other parameters are determined by the MQPMO.PutMsgRecFields field):

CompCode

Description:	The completion code indicating the result of the operation. Only included if MQRR structures are provided and the reason code for the MQPUT is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	2
Type:	MQCFIN

Reason

Description:	The reason code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	2
Type:	MQCFIN

MsgId

Description:	Message identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_MSG_ID
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS

[3] The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_CORREL_ID
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

GroupId

Description:	Group identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_GROUP_ID
PCF Parameter:	MQBACF_GROUP_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_GROUP_ID_LENGTH

Feedback

Description:	Feedback. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_FEEDBACK
PCF Parameter:	MQIACF_FEEDBACK
Trace level:	2
Type:	MQCFIN

AccountingToken

Description:	AccountingToken. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_ACCOUNTING_TOKEN
PCF Parameter:	MQBACF_ACCOUNTING_TOKEN
Trace level:	2
Type:	MQCFBS
Length:	MQ_ACCOUNTING_TOKEN_LENGTH.

MQPUT1
Application has started the MQPUT1 MQI function

ObjectType

Description:	The object type passed in MQOT.ObjectType
PCF Parameter:	MQIACF_OBJECT_TYPE
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ObjectQMgrName

Description:	The name of the object queue manager passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

PutOptions

Description:	The put options from MQPMO.Options
PCF Parameter:	MQIACF_PUT_OPTIONS
Trace level:	1
Type:	MQCFIN

AlternateUserId

Description:	Only included if MQPMO_ALTERNATE_USER_AUTHORITY is specified.
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Length:	MQ_USER_ID_LENGTH

RecsPresent

Description:	The number of object name records present
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of local queues opened successfully
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of remote queues opened successfully
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of queues that failed to open
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

MsgBuffer

Description:	Message data.
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type:	MQCFBS
Length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:	Length of the message.
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type:	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1st 1970 (UTC) **Note:** The accuracy of this timer will vary according to platform support for high a resolution timer.

PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type:	MQCFIN64

ResolvedQName

Description:	The name of the queue after queue name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQMgrName

Description:	The queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ResolvedLocalQName [4]

Description:	Contains the local queue name after name resolution has been carried out
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST

ResolvedLocalQMgrName [4]

Description:	Contains the local queue manager name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

AlternateSecurityId

Description:	Alternate security identifier. Only present if MQOD.Version is equal or greater than MQOD_VERSION_3 and MQOD.AlternateSecurityId is not equal to MQSID_NONE.
PCF Parameter:	MQBACF_ALTERNATE_SECURITYID
Trace level:	2
Type:	MQCFBS
Length:	MQ_SECURITY_ID_LENGTH

ObjectString

Description:	Long object name. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD.ObjectString is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResObjectString

Description:	The long object name after the queue manager resolves the name provided in the ObjectName field. Only included for topics and queue aliases that reference a topic object if MQOD.Version is equal or greater than MQOD_VERSION_4 and VSLength is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the resolved (base) object being opened. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

ReportOptions

Description:	Message report options
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type:	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type:	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2

Type: MQCFIN

Format

Description: Format name of message data

PCF Parameter: MQCACH_FORMAT_NAME

Trace level: 2

Type: MQCFST

Length: MQ_FORMAT_LENGTH

Priority

Description: Message priority

PCF Parameter: MQIACF_PRIORITY

Trace level: 2

Type: MQCFIN

Persistence

Description: Message persistence

PCF Parameter: MQIACF_PERSISTENCE

Trace level: 2

Type: MQCFIN

MsgId

Description: Message identifier

PCF Parameter: MQBACF_MSG_ID

Trace level: 2

Type: MQCFBS

Length: MQ_MSG_ID_LENGTH

CorrelId

PCF Parameter: Correlation identifier

Description: MQBACF_CORREL_ID

Trace level: 2

Type: MQCFBS

Length: MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:

PCF Parameter: MQCACF_REPLY_TO_Q

Trace level: 2

Type: MQCFST

Length: MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQCFST

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type:	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type:	MQCFIN

PutDate

Description:	
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

PutTime

Description:	
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The remote queue manager destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

PutMsgOptsStructure

Description:	The MQPMO structure.
PCF Parameter:	MQBACF_MQPMO_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQPMO structure (actual size depends on structure version)

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQPUT1 AppActivityDistList PCF Group Header Structure

If the MQPUT1 function is putting to a distribution list, then the variable parameters include one AppActivityDistList PCF group. For each of the queues in the distribution list, see “Application Activity Distribution List PCF Group Header Structure” on page 251. The AppActivityDistList PCF group combines information from the MQOR, MQPMR, and MQRR structures to identify the objects, and the PUT parameters , and indicate the result of the PUT operation on each queue. For MQPUT1 operations the AppActivityDistList group contains some or all of the following parameters (the CompCode, Reason, ObjectName, and ObjectQMgrName is present if the reason code is MQRC_MULTIPLE_REASONS and the other parameters is determined by the MQPMO.PutMsgRecFields field):

CompCode

Description:	The completion code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT1 is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_COMP_CODE

[4] The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Trace level:	2
Type:	MQCFIN

Reason

Description:	The reason code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT1 is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	2
Type:	MQCFIN

ObjectName

Description:	The name of a queue in the distribution list. Only included if MQOR structures are provided.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

MsgId

Description:	Message identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_MSG_ID
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_CORREL_ID
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

GroupId

Description:	Group identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_GROUP_ID
PCF Parameter:	MQBACF_GROUP_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_GROUP_ID_LENGTH

Feedback

Description:	Feedback. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_FEEDBACK
PCF Parameter:	MQIACF_FEEDBACK
Trace level:	2
Type:	MQCFIN

AccountingToken

Description:	AccountingToken. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_ACCOUNTING_TOKEN
PCF Parameter:	MQBACF_ACCOUNTING_TOKEN
Trace level:	2
Type:	MQCFBS
Length:	MQ_ACCOUNTING_TOKEN_LENGTH.

MQSET
Application has started the MQSET MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SelectorCount

Description:	The count of selectors that are supplied in the Selectors array.
PCF Parameter:	MQIACF_SELECTOR_COUNT
Trace level:	2
Type:	MQCFIN

Selectors

Description:	The list of attributes (integer or character) whose values are being updated by MQSET.
PCF Parameter:	MQIACF_SELECTORS
Trace level:	2
Type:	MQCFIL

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

IntAttrCount

Description:	The number of integer attributes to be updated by the set operation.
PCF Parameter:	MQIACF_INTATTR_COUNT
Trace level:	3
Type:	MQCFIN

IntAttrs

Description:	The integer attribute values
PCF Parameter:	MQIACF_INT_ATTRS
Trace level:	3
Type:	MQCFIL
Range:	This parameter is only present if IntAttrCount is > 0

CharAttrs

Description:	The character attributes to be updated by the set operation. The values are concatenated together.
PCF Parameter:	MQCACF_CHAR_ATTRS
Trace level:	3
Type:	MQCFST
Range:	This parameter is only included if CharAttrLength is > 0

V 8.0.0.2 ***QMgrOpDuration***

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQSUB
Application has started the MQSUB MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SubHandle

Description:	The subscription handle
PCF Parameter:	MQIACF_HSUB
Trace level:	1
Type:	MQCFIN

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1

Type:	MQCFIN

Options

Description:	Subscription options
PCF Parameter:	MQIACF_SUB_OPTIONS
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ObjectString

Description:	Long object name.
PCF Parameter:	MQCACF_OBJECT_STRING
Trace level:	1
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.ObjectString is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

AlternateUserId

Description:	
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Range:	Only included if MQSO_ALTERNATE_USER_AUTHORITY is specified.
Length:	MQ_USER_ID_LENGTH

AlternateSecurityId

Description:	Alternate security identifier.
PCF Parameter:	MQBACF_ALTERNATE_SECURITYID
Trace level:	2
Type:	MQCFBS
Range:	Only present if MQSO_ALTERNATE_USER_AUTHORITY is specified and MQSD.AlternateSecurityId is not equal to MQSID_NONE.
Length:	MQ_SECURITY_ID_LENGTH

SubName

Description:	Subscription Name
PCF Parameter:	MQCACF_SUB_NAME
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.SubName is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

SubUserData

Description:	Subscription User Data
PCF Parameter:	MQCACF_SUB_USER_DATA
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.SubName is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

SubCorrelId

Description:	Subscription Correlation identifier
PCF Parameter:	MQBACF_SUB_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

SelectionString

Description:	Selection string.
PCF Parameter:	MQCACF_SELECTION_STRING
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD. SelectionString is MQVS_NULL_TERMINATED or greater than zero.
Length:	Length varies.

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

SubDescriptorStructure

Description:	The MQSD structure.
PCF Parameter:	MQBACF_MQSD_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQSD structure.

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQSUBRQ
Application has started the MQSUBRQ MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type: MQCFIN

SubHandle

Description: The subscription handle

PCF Parameter: MQIACF_HSUB

Trace level: 1

Type: MQCFIN

SubOptions

Description: The sub options from MQSB.Options

PCF Parameter: MQIACF_SUBRQ_OPTIONS

Trace level: 2

Type: MQCFIN

Action

Description: The subscription request action (MQSR_*)

PCF Parameter: MQIACF_SUBRQ_ACTION

Trace level: 2

Type: MQCFIN

NumPubs

Description: The number of publications sent as a result of this call (from MQSB.NumPubs)

PCF Parameter: MQIACF_NUM_PUBS

Trace level: 2

Type: MQCFIN

V 8.0.0.2 ***QMgrOpDuration***

Description: Approximate API call duration, in microseconds, within the queue manager.

The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter: MQIAMO64_QMGR_OP_DURATION

Trace level: 2

Type MQCFIN64

MQSTAT
Application has started the MQSTAT MQI function

CompCode

Description: The completion code indicating the result of the operation

PCF Parameter: MQIACF_COMP_CODE

Trace level: 1

Type: MQCFIN

Reason

Description: The reason code result of the operation

PCF Parameter: MQIACF_REASON_CODE

Trace level: 1

Type: MQCFIN

Type

Description: Type of status information being requested

PCF Parameter: MQIACF_STATUS_TYPE

Trace level: 2

Type: MQCFIN

StatusStructure

Description: The MQSTS structure.

PCF Parameter: MQBACF_MQSTS_STRUCT

Trace level: 3

Type: MQCFBS

Length: The length in bytes of the MQSTS structure (actual size depends on structure version)

V 8.0.0.2 ***QMgrOpDuration***

Description: Approximate API call duration, in microseconds, within the queue manager.

The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client.

Note: The accuracy of this timer varies according to the platform that your enterprise uses.

PCF Parameter: MQIAMO64_QMGR_OP_DURATION

Trace level: 2

Type MQCFIN64

Variable Parameters for Application Activity XA Operations

XA operations are API calls that applications can make to enable MQ to participate in a transaction. The parameters for each operation are defined in the following section.

The trace level indicates the level of trace granularity that is required for the parameters to be included in the trace. The possible trace level values are:

1. Low

   The parameter is included when "low", "medium" or "high" activity tracing is configured for an application. This setting means that a parameter is always included in the `AppActivityData` group for the operation. This set of parameters is sufficient to trace the MQI calls an application makes, and to see if they are successful.

2. Medium

   The parameter is only included in the `AppActivityData` group for the operation when "medium" or "high" activity tracing is configured for an application. This set of parameters adds information about the resources, for example, queue and topic names used by the application.

3. High

   The parameter is only included in the `AppActivityData` group for the operation when "high" activity tracing is configured for an application. This set of parameters includes memory dumps of the structures passed to the MQI and XA functions. For this reason, it contains more information about the parameters used in MQI and XA calls. The structure memory dumps are shallow copies of the structures. To avoid erroneous attempts to dereference pointers, the pointer values in the structures are set to `NULL`.

   Note: The version of the structure that is dumped is not necessarily identical to the version used by an application. The structure can be modified by an API crossing exit, by the activity trace code, or by the queue manager. A queue manager can modify a structure to a later version, but the queue manager never changes it to an earlier version of the structure. To do so, would risk losing data.

AXREG
Application has started the AXREG AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

AXUNREG
Application has started the AXUNREG AX function

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XACLOSE
Application has started the XACLOSE AX function

Xa_info

Description:	Information used to initialize the resource manager.
PCF Parameter:	MQCACF_XA_INFO
Trace level:	1
Type:	MQCFST

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code

PCF Parameter: MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

XACOMMIT
Application has started the XACOMMIT AX function

XID

Description: The XID structure

PCF Parameter: MQBACF_XA_XID

Trace level: 1

Type: MQCFBS

Length: Sizeof(XID)

Rmid

Description: Resource manager identifier

PCF Parameter: MQIACF_XA_RMID

Trace level: 1

Type: MQCFIN

Flags

Description: Flags

PCF Parameter: MQIACF_XA_FLAGS

Trace level: 1

Type: MQCFIN

XARetCode

Description: Return code

PCF Parameter: MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

XACOMPLETE
Application has started the XACOMPLETE AX function

Handle

Description: Handle to async operation

PCF Parameter: MQIACF_XA_HANDLE

Trace level: 1

Type: MQCFIN

Retval

Description: Return value of the asynchronous function

PCF Parameter: MQIACF_XA_RETVAL

Trace level: 1

Type: MQCFINMQCFBS

Rmid

Description: Resource manager identifier

PCF Parameter: MQIACF_XA_RMID

Trace level: 1

Type: MQCFIN

Flags

Description: Flags

PCF Parameter: MQIACF_XA_FLAGS

Trace level: 1

Type: MQCFIN

XARetCode

Description: Return code

PCF Parameter: MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

XAEND
Application has started the XAEND AX function

XID

Description: The XID structure

PCF Parameter: MQBACF_XA_XID

Trace level: 1

Type: MQCFBS

Length: Sizeof(XID)

Rmid

Description: Resource manager identifier

PCF Parameter: MQIACF_XA_RMID

Trace level: 1

Type: MQCFIN

Flags

Description: Flags

PCF Parameter: MQIACF_XA_FLAGS

Trace level: 1

Type: MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAFORGET
Application has started the AXREG AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAOPEN
Application has started the XAOPEN AX function

Xa_info

Description:	Information used to initialize the resource manager.
PCF Parameter:	MQCACF_XA_INFO
Trace level:	1
Type:	MQCFST

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAPREPARE
Application has started the XAPREPARE AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

XARECOVER
Application has started the XARECOVER AX function

Count

Description: Count of XIDs

PCF Parameter: MQIACF_XA_COUNT

Trace level: 1

Type: MQCFIN

XIDs

Description: The XID structures

Note: There are multiple instances of this PCF parameter - one for every XID structure up to Count XIDs

PCF Parameter: MQBACF_XA_XID

Trace level: 1

Type: MQCFBS

Length: Sizeof(XID)

Rmid

Description: Resource manager identifier

PCF Parameter: MQIACF_XA_RMID

Trace level: 1

Type: MQCFIN

Flags

Description: Flags

PCF Parameter: MQIACF_XA_FLAGS

Trace level: 1

Type: MQCFIN

XARetCode

Description: Return code

PCF Parameter: MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

XAROLLBACK
Application has started the XAROLLBACK AX function

XID

Description: The XID structure

PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XASTART
Application has started the XASTART AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1

Type: MQCFIN

XARetCode

Description: Return code

PCF Parameter: MQIACF_XA_RETCODE

Trace level: 1

Type: MQCFIN

Real-time monitoring

Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

A number of commands are available that when issued return real-time information about queues and channels. Information can be returned for one or more queues or channels and can vary in quantity. Real-time monitoring can be used in the following tasks:

- Helping system administrators understand the steady state of their IBM MQ system. This helps with problem diagnosis if a problem occurs in the system.
- Determining the condition of your queue manager at any moment, even if no specific event or problem has been detected.
- Assisting with determining the cause of a problem in your system.

With real-time monitoring, information can be returned for either queues or channels. The amount of real-time information returned is controlled by queue manager, queue, and channel attributes.

- You monitor a queue by issuing commands to ensure that the queue is being serviced properly. Before you can use some of the queue attributes, you must enable them for real-time monitoring.
- You monitor a channel by issuing commands to ensure that the channel is running properly. Before you can use some of the channel attributes, you must enable them for real-time monitoring.

Real-time monitoring for queues and channels is in addition to, and separate from, performance and channel event monitoring.

Attributes that control real-time monitoring

Some queue and channel status attributes hold monitoring information, if real-time monitoring is enabled. If real-time monitoring is not enabled, no monitoring information is held in these monitoring attributes. Examples demonstrate how you can use these queue and channel status attributes.

You can enable or disable real-time monitoring for individual queues or channels, or for multiple queues or channels. To control individual queues or channels, set the queue attribute MONQ or the channel attribute MONCHL, to enable or disable real-time monitoring. To control many queues or channels together, enable or disable real-time monitoring at the queue manager level by using the queue manager attributes MONQ and MONCHL. For all queue and channel objects with a monitoring attribute that is specified with the default value, QMGR, real-time monitoring is controlled at the queue manager level.

Automatically defined cluster-sender channels are not IBM MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute, MONACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel monitoring.

For real-time monitoring of channels, you can set the MONCHL attribute to one of the three monitoring levels: low, medium, or high. You can set the monitoring level either at the object level or at the queue manager level. The choice of level is dependent on your system. Collecting monitoring data might require some instructions that are relatively expensive computationally, such as obtaining system time. To reduce the effect of real-time monitoring, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 30 on page 288 summarizes the monitoring levels available for real-time monitoring of channels:

Table 30. Monitoring levels

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

For real-time monitoring of queues, you can set the MONQ attribute to one of the three monitoring levels, low, medium or high. However, there is no distinction between these values. The values all enable data collection, but do not affect the size of the sample.

Examples

The following examples demonstrate how to set the necessary queue, channel, and queue manager attributes to control the level of monitoring. For all of the examples, when monitoring is enabled, queue and channel objects have a medium level of monitoring.

1. To enable both queue and channel monitoring for all queues and channels at the queue manager level, use the following commands:

```
ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
ALTER QL(Q1) MONQ(QMGR)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(QMGR)
```

2. To enable monitoring for all queues and channels, with the exception of local queue, Q1, and sender channel, `QM1.TO.QM2`, use the following commands:

```
ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
ALTER QL(Q1) MONQ(OFF)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(OFF)
```

3. To disable both queue and channel monitoring for all queues and channels, with the exception of local queue, Q1, and sender channel, `QM1.TO.QM2`, use the following commands:

```
ALTER QMGR MONQ(OFF) MONCHL(OFF)
ALTER QL(Q1) MONQ(MEDIUM)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(MEDIUM)
```

4. To disable both queue and channel monitoring for all queues and channels, regardless of individual object attributes, use the following command:

```
ALTER QMGR MONQ(NONE) MONCHL(NONE)
```

5. To control the monitoring capabilities of automatically defined cluster-sender channels use the following command:

```
ALTER QMGR MONACLS(MEDIUM)
```

6. To specify that automatically defined cluster-sender channels are to use the queue manager setting for channel monitoring, use the following command:

```
ALTER QMGR MONACLS(QMGR)
```

Related concepts

"Real-time monitoring" on page 287
Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

"Using IBM MQ online monitoring" on page 305
You can collect monitoring data for queues and channels (including automatically defined cluster-server channels) by setting the MONQ, MONCHL, and MONACLS attributes.

Related tasks

"Displaying queue and channel monitoring data" on page 289
To display real-time monitoring information for a queue or channel, use either the IBM MQ Explorer or the appropriate MQSC command. Some monitoring fields display a comma-separated pair of indicator values, which help you to monitor the operation of your queue manager. Examples demonstrate how you can display monitoring data.

Related information

Working with queue managers

Monitoring (MONCHL)

Displaying queue and channel monitoring data

To display real-time monitoring information for a queue or channel, use either the IBM MQ Explorer or the appropriate MQSC command. Some monitoring fields display a comma-separated pair of indicator values, which help you to monitor the operation of your queue manager. Examples demonstrate how you can display monitoring data.

About this task

Monitoring fields that display a pair of values separated by a comma provide short term and long term indicators for the time measured since monitoring was enabled for the object, or from when the queue manager was started:

- The short term indicator is the first value in the pair and is calculated in a way such that more recent measurements are given a higher weighting and will have a greater effect on this value. This gives an indication of recent trend in measurements taken.
- The long term indicator in the second value in the pair and is calculated in a way such that more recent measurements are not given such a high weighting. This gives an indication of the longer term activity on performance of a resource.

These indicator values are most useful to detect changes in the operation of your queue manager. This requires knowledge of the times these indicators show when in normal use, in order to detect increases in these times. By collecting and checking these values regularly you can detect fluctuations in the operation of your queue manager. This can indicate a change in performance.

Obtain real-time monitoring information as follows:

Procedure

1. To display real-time monitoring information for a queue, use either the IBM MQ Explorer or the MQSC command `DISPLAY QSTATUS`, specifying the optional parameter `MONITOR`.
2. To display real-time monitoring information for a channel, use either the IBM MQ Explorer or the MQSC command `DISPLAY CHSTATUS`, specifying the optional parameter `MONITOR`.

Example

The queue, Q1, has the attribute MONQ set to the default value, QMGR, and the queue manager that owns the queue has the attribute MONQ set to MEDIUM. To display the monitoring fields collected for this queue, use the following command:

```
DISPLAY QSTATUS(Q1) MONITOR
```

The monitoring fields and monitoring level of queue, Q1 are displayed as follows:

```
QSTATUS(Q1)
TYPE(QUEUE)
MONQ(MEDIUM)
QTIME(11892157,24052785)
MSGAGE(37)
LPUTDATE(2005-03-02)
LPUTTIME(09.52.13)
LGETDATE(2005-03-02)
LGETTIME(09.51.02)
```

The sender channel, `QM1.TO.QM2`, has the attribute MONCHL set to the default value, QMGR, and the queue manager that owns the queue has the attribute MONCHL set to MEDIUM. To display the monitoring fields collected for this sender channel, use the following command:

```
DISPLAY CHSTATUS(QM1.TO.QM2) MONITOR
```

The monitoring fields and monitoring level of sender channel, `QM1.TO.QM2` are displayed as follows:

```
CHSTATUS(QM1.TO.QM2)
XMITQ(Q1)
CONNAME(127.0.0.1)
CURRENT
CHLTYPE(SDR)
STATUS(RUNNING)
SUBSTATE(MQGET)
MONCHL(MEDIUM)
XQTIME(755394737,755199260)
NETTIME(13372,13372)
EXITTIME(0,0)
XBATCHSZ(50,50)
COMPTIME(0,0)
STOPREQ(NO)
RQMNAME(QM2)
```

Related concepts

"Real-time monitoring" on page 287
Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

Related information

DISPLAY QSTATUS

Monitoring queues

Use this page to view tasks that help you to resolve a problem with a queue and the application that services that queue. Various monitoring options are available to determine the problem

Frequently, the first sign of a problem with a queue that is being serviced is that the number of messages on the queue (CURDEPTH) increases. If you expect an increase at certain times of day or under certain workloads, an increasing number of messages might not indicate a problem. However, if you have no explanation for the increasing number of messages, you might want to investigate the cause.

You might have an application queue where there is a problem with the application, or a transmission queue where there is a problem with the channel. Additional monitoring options are available when the application that services the queue is a channel.

The following examples investigate problems with a particular queue, called Q1, and describe the fields that you look at in the output of various commands:

Determining whether your application has the queue open

If you have a problem with a queue, check whether your application has the queue open

About this task

Perform the following steps to determine whether your application has the queue open:

Procedure

1. Ensure that the application that is running against the queue is the application that you expect. Issue the following command for the queue in question:

   ```
   DISPLAY QSTATUS(Q1) TYPE(HANDLE) ALL
   ```

   In the output, look at the APPLTAG field, and check that the name of your application is shown. If the name of your application is not shown, or if there is no output at all, start your application.
2. If the queue is a transmission queue, look in the output at the CHANNEL field.

   If the channel name is not shown in the CHANNEL field, determine whether the channel is running.
3. Ensure that the application that is running against the queue has the queue open for input. Issue the following command:

   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```

   In the output, look at the IPPROCS field to see if any application has the queue open for input. If the value is 0 and this is a user application queue, make sure that the application opens the queue for input to get the messages off the queue.

Checking that messages on the queue are available

If you have a large number of messages on the queue and your application is not processing any of those messages, check whether the messages on the queue are available to your application

About this task

Perform the following steps to investigate why your application is not processing messages from the queue:

Procedure

1. Ensure that your application is not asking for a specific message ID or correlation ID when it should be processing all the messages on the queue.
2. Although the current depth of the queue might show that there is an increasing number of messages on the queue, some messages on the queue might not be available to be got by an application, because they are not committed; the current depth includes the number of uncommitted MQPUTs of messages to the queue. Issue the following command:

   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```

   In the output, look at the UNCOM field to see whether there are any uncommitted messages on the queue.

3. If your application is attempting to get any messages from the queue, check whether the putting application is committing the messages correctly. Issue the following command to find out the names of applications that are putting messages to this queue:

```
DISPLAY QSTATUS(Q1) TYPE(HANDLE) OPENTYPE(OUTPUT)
```

4. Then issue the following command, inserting in <appltag> the APPLTAG value from the output of the previous command:

```
DISPLAY CONN(*) WHERE(APPLTAG EQ <appltag>) UOWSTDA UOWSTTI
```

   This shows when the unit of work was started and will help you discover whether the application is creating a long running unit of work. If the putting application is a channel, you might want to investigate why a batch is taking a long time to complete.

Checking whether your application is getting messages off the queue

If you have a problem with a queue and the application that services that queue, check whether your application is getting messages off the queue

About this task

To check whether your application is getting messages off the queue, perform the following checks:

Procedure

1. Ensure that the application that is running against the queue is actually processing messages from the queue. Issue the following command:

```
DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
```

   In the output, look at the LGETDATE and LGETTIME fields which show when the last get was done from the queue.
2. If the last get from this queue was longer ago than expected, ensure that the application is processing messages correctly.

   If the application is a channel, check whether messages are moving through that channel

Determining whether the application can process messages fast enough

If messages are building up on the queue, but your other checks have not found any processing problems, check that the application can process messages fast enough. If the application is a channel, check that the channel can process messages fast enough.

About this task

To determine whether the application is processing messages fast enough, perform the following tests:

Procedure

1. Issue the following command periodically to gather performance data about the queue:

```
DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
```

   If the values in the QTIME indicators are high, or are increasing over the period, and you have already ruled out the possibility of long running Units of Work by checking that messages on the queue are available, the getting application might not be keeping up with the putting applications.
2. If your getting application cannot keep up with the putting applications, consider adding another getting application to process the queue.

Whether you can add another getting application depends on the design of the application and whether the queue can be shared by more than one application. Features such as message grouping or getting by correlation ID might help to ensure that two applications can process a queue simultaneously.

Checking the queue when the current depth is not increasing

Even if the current depth of your queue is not increasing, it might still be useful to monitor the queue to check whether your application is processing messages correctly.

About this task

To gather performance data about the queue: Issue the following command periodically:

Procedure

Issue the following command periodically:

```
DISPLAY QSTATUS(Q1) TYPE(QUEUE) MSGAGE QTIME
```

In the output, if the value in MSGAGE increases over the period of time, and your application is designed to process all messages, this might indicate that some messages are not being processed at all.

Monitoring channels

Use this page to view tasks that help you to resolve a problem with a transmission queue and the channel that services that queue. Various channel monitoring options are available to determine the problem.

Frequently, the first sign of a problem with a queue that is being serviced is that the number of messages on the queue (CURDEPTH) increases. If you expect an increase at certain times of day or under certain workloads, an increasing number of messages might not indicate a problem. However, if you have no explanation for the increasing number of messages, you might want to investigate the cause.

You might have a problem with the channel that services a transmission queue. Various channel monitoring options are available to help you to determine the problem.

The following examples investigate problems with a transmission queue called QM2 and a channel called QM1.TO.QM2. This channel is used to send messages from queue manager, QM1, to queue manager, QM2. The channel definition at queue manager QM1 is either a sender or server channel, and the channel definition at queue manager, QM2, is either a receiver or requester channel.

Determining whether the channel is running

If you have a problem with a transmission queue, check whether the channel is running.

About this task

Perform the following steps to check the status of the channel that is servicing the transmission queue:

Procedure

1. Issue the following command to find out which channel you expect to process the transmission queue QM2:

   ```
   DIS CHANNEL(*) WHERE(XMITQ EQ QM2)
   ```

   In this example, the output of this command shows that the channel servicing the transmission queue is QM1.TO.QM2
2. Issue the following command to determine the status of the channel, QM1.TO.QM2:

   ```
   DIS CHSTATUS(QM1.TO.QM2) ALL
   ```

3. Inspect the STATUS field of the output from the **CHSTATUS** command:
   - If the value of the STATUS field is `RUNNING`, check that the channel is moving messages
   - If the output from the command shows no status, or the value of the STATUS field is `STOPPED`, `RETRY`, `BINDING`, or `REQUESTING`, perform the appropriate step, as follows:
4. Optional: If the value of the STATUS field shows no status, the channel is inactive, so perform the following steps:
   a) If the channel should have been started automatically by a trigger, check that the messages on the transmission queue are available.

      If there are messages available on the transmission queue, check that the trigger settings on the transmission queue are correct.

   b) Issue the following command to start the channel again manually:

      ```
      START CHANNEL(QM1.TO.QM2)
      ```
5. Optional: If the value of the STATUS field is `STOPPED`, perform the following steps:
   a) Check the error logs to determine why the channel stopped. If the channel stopped owing to an error, correct the problem.

      Ensure also that the channel has values specified for the retry attributes: *SHORTRTY* and *LONGRTY*. In the event of transient failures such as network errors, the channel will then attempt to restart automatically.

   b) Issue the following command to start the channel again manually:

      ```
      START CHANNEL(QM1.TO.QM2)
      ```

   z/OS On IBM MQ for z/OS, you can detect when a user stops a channel by using command event messages.
6. Optional: If the value of the STATUS field is RETRY, perform the following steps:
   a) Check the error logs to identify the error, then correct the problem.
   b) Issue the following command to start the channel again manually:

      ```
      START CHANNEL(QM1.TO.QM2)
      ```

      or wait for the channel to connect successfully on its next retry.
7. Optional: If the value of the STATUS field is `BINDING` or `REQUESTING`, the channel has not yet successfully connected to the partner. Perform the following steps:
   a) Issue the following command, at both ends of the channel, to determine the substate of the channel:

      ```
      DIS CHSTATUS(QM1.TO.QM2) ALL
      ```

      Note:

      i) In some cases there might be a substate at one end of the channel only.

      ii) Many substates are transitory, so issue the command a few times to detect whether a channel is stuck in a particular substate.

   b) Check Table 31 on page 295 to determine what action to take:

Table 31. Substates seen with status binding or requesting		
Initiating MCA substate [1]	**Responding MCA substate [2]**	**Notes**
NAMESERVER		The initiating MCA is waiting for a name server request to complete. Ensure that the correct host name has been specified in the channel attribute, CONNAME, and that your name servers are set up correctly.
SCYEXIT	SCYEXIT	The MCAs are currently *in conversation* through a security exit. For more information, see "Determining whether the channel can process messages fast enough" on page 297.
	CHADEXIT	The channel autodefinition exit is currently executing. For more information, see "Determining whether the channel can process messages fast enough" on page 297.
RCVEXIT SENDEXIT MSGEXIT MREXIT	RCVEXIT SENDEXIT MSGEXIT MREXIT	Exits are called at channel startup for MQXR_INIT. Review the processing in this part of your exit if this takes a long time. For more information, see "Determining whether the channel can process messages fast enough" on page 297.
SERIALIZE	SERIALIZE	This substate only applies to channels with a disposition of SHARED.
NETCONNECT		This substate is shown if there is a delay in connecting due to incorrect network configuration.
SSLHANDSHAKE	SSLHANDSHAKE	An SSL handshake consists of a number of sends and receives. If network times are slow, or connection to lookup CRLs are slow, this affects the time taken to do the handshake. z/OS On IBM MQ for z/OS this substate can also be indicative of not having enough SSLTASKS.

Notes:

i) The initiating MCA is the end of the channel which started the conversation. This can be senders, cluster-senders, fully-qualified servers and requesters. In a server-requester pair, it is the end from which you started the channel.

ii) The responding MCA is the end of the channel which responded to the request to start the conversation. This can be receivers, cluster-receivers, requesters (when the server or sender is started), servers (when the requester is started) and senders (in a requester-sender call-back pair of channels).

Checking that the channel is moving messages

If you have a problem with a transmission queue, check that the channel is moving messages

Before you begin

Issue the command `DIS CHSTATUS(QM1.TO.QM2) ALL`. If the value of the STATUS field is `RUNNING`, the channel has successfully connected to the partner system.

Check that there are no uncommitted messages on the transmission queue, as described in "Checking that messages on the queue are available" on page 291.

About this task

If there are messages available for the channel to get and send, perform the following checks:

Procedure

1. In the output from the display channel status command, `DIS CHSTATUS(QM1.TO.QM2) ALL`, look at the following fields:

   MSGS
   Number of messages sent or received (or, for server-connection channels, the number of MQI calls handled) during this session (since the channel was started).

   BUFSSENT
   Number of transmission buffers sent. This includes transmissions to send control information only.

   BYTSSENT
   Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

   LSTMSGDA
   Date when the last message was sent or MQI call was handled, see LSTMSGTI.

   LSTMSGTI
   Time when the last message was sent or MQI call was handled. For a sender or server, this is the time the last message (the last part of it if it was split) was sent. For a requester or receiver, it is the time the last message was put to its target queue. For a server-connection channel, it is the time when the last MQI call completed.

   CURMSGS
   For a sending channel, this is the number of messages that have been sent in the current batch. For a receiving channel, it is the number of messages that have been received in the current batch. The value is reset to zero, for both sending and receiving channels, when the batch is committed.

   STATUS
   The status of the channel, which can be `Starting`, `Binding`, `Initializing`, `Running`, `Stopping`, `Retrying`, `Paused`, `Stopped`, or `Requesting`.

   SUBSTATE
   The action that the channel is currently performing.

   INDOUBT
   Whether the channel is currently in doubt. This is only YES while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages that it has sent has been successfully received. It is NO at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested. For a receiving channel, the value is always `NO`.
2. Determine whether the channel has sent any messages since it started. If any have been sent, determine when the last message was sent.
3. The channel might have started a batch that has not yet completed, as indicated by a non-zero value in CURMSGS. If INDOUBT is YES, the channel is waiting to receive acknowledgment that the other end of the channel received the batch. Look at the SUBSTATE field in the output and refer to Table 32 on page 296:

Table 32. Sender and receiver MCA substates

Sender SUBSTATE	Receiver SUBSTATE	Notes
MQGET	RECEIVE	Normal states of a channel at rest.
SEND	RECEIVE	SEND is usually a transitory state. If SEND is seen it indicates that the communication protocol buffers have filled. This can indicate a network problem.
RECEIVE		If the sender is seen in RECEIVE substate for any length of time, it is waiting on a response, either to a batch completion or a heartbeat. You might want to check why a batch takes a long time to complete.

Note: You might also want to determine whether the channel can process messages fast enough, especially if the channel has a substate associated with exit processing.

Checking why a batch takes a long time to complete

Reasons why a batch can take a long time to complete include a slow network or a channel is using message retry processing.

About this task

When a sender channel has sent a batch of messages it waits for confirmation of that batch from the receiver, unless the channel is pipelined. The factors described in this task can affect how long the sender channel waits.

Procedure

- Check whether the network is slow.

  The NETTIME value is the amount of time, displayed in microseconds, taken to send an end of batch request to the remote end of the channel and receive a response minus the time to process the end of batch request. This value can be large for either of the following reasons:

  – The network is slow. A slow network can affect the time it takes to complete a batch. The measurements that result in the indicators for the NETTIME field are measured at the end of a batch. However, the first batch affected by a slowdown in the network is not indicated with a change in the NETTIME value because it is measured at the end of the batch.

  – Requests are queued at the remote end, for example a channel can be retrying a put, or a put request may be slow due to page set I/O. Once any queued requests have completed, the duration of the end of batch request is measured. So if you get a large NETTIME value, check for unusual processing at the remote end.
- Check whether the channel is using message retry.

  If the receiver channel fails to put a message to a target queue, it might use message retry processing, rather than put the message to a dead-letter queue immediately. Retry processing can cause the batch to slow down. In between MQPUT attempts, the channel will have STATUS(PAUSED), indicating that it is waiting for the message retry interval to pass.

Determining whether the channel can process messages fast enough

If there messages are building up on the transmission queue, but you have found no processing problems, determine whether the channel can process messages fast enough.

Before you begin

Issue the following command repeatedly over a period of time to gather performance data about the channel:

```
DIS CHSTATUS(QM1.TO.QM2) ALL
```

About this task

Confirm that there are no uncommitted messages on the transmission queue, as described in "Checking that messages on the queue are available" on page 291, then check the XQTIME field in the output from the display channel status command. When the values of the XQTIME indicators are consistently high, or increase over the measurement period, the indication is that the channel is not keeping pace with the putting applications.

Perform the following tests:

Procedure

1. Check whether exits are processing.

If exits are used on the channel that is delivering these messages, they might add to the time spent processing messages. To identify if this is the case, do the following checks:

a) In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the EXITTIME field.

   If the time spent in exits is higher than expected, review the processing in your exits for any unnecessary loops or extra processing, especially in message, send, and receive exits. Such processing affects all messages moved across the channel.

b) In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the SUBSTATE field.

   If the channel has of one of the following substates for a significant time, review the processing in your exits:

   - SCYEXIT
   - RCVEXIT
   - SENDEXIT
   - MSGEXIT
   - MREXIT

2. Check whether the network is slow.

   If messages are not moving fast enough across a channel, it might be because the network is slow. To identify if this is the case, do the following checks:

   a) In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the NETTIME field.

      These indicators are measured when the sending channel asks its partner for a response. This happens at the end of each batch and, when a channel is idle during heartbeating.

   b) If this indicator shows that round trips are taking longer than expected, use other network monitoring tools to investigate the performance of your network.

3. Check whether the channel is using compression.

   If the channel is using compression, this adds to the time spent processing messages. If the channel is using only one compression algorithm, do the following checks:

   a) In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME field.

      These indicators show the time spent during compression or decompression.

   b) If the chosen compression is not reducing the amount of data to send by the expected amount, change the compression algorithm.

4. If the channel is using multiple compression algorithms, do the following checks:

   a) In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME, COMPHDR, and COMPMSG fields.

   b) Change the compression algorithms specified on the channel definition, or consider writing a message exit to override the channel's choice of compression algorithm for particular messages if the rate of compression, or choice of algorithm, is not providing the required compression or performance.

Solving problems with cluster channels

If you have a build up of messages on the SYSTEM.CLUSTER.TRANSMIT.QUEUE queue, the first step in diagnosing the problem is discovering which channel, or channels, are having a problem delivering messages.

About this task

To discover which channel, or channels, using the SYSTEM.CLUSTER.TRANSMIT.QUEUE are having a problem delivering messages. Perform the following checks:

Procedure

1. Issue the following command:

```
DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
```

Note: If you have a busy cluster that has many messages moving, consider issuing this command with a higher number to eliminate the channels that have only a few messages available to deliver.

2. Look through the output for the channel, or channels, that have large values in the field XQMSGSA. Determine why the channel is not moving messages, or is not moving them fast enough. Use the tasks outlined in "Monitoring channels" on page 293 to diagnose the problems with the channels found to be causing the build up.

The Windows performance monitor

In IBM WebSphere MQ 7.0 and earlier versions, it was possible to monitor the performance of local queues on Windows systems by using the Windows performance monitor. As of IBM WebSphere MQ 7.1, this method of performance monitoring is no longer available.

You can monitor queues on all supported platforms by using methods described in "Real-time monitoring" on page 287.

Monitoring clusters

Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Monitoring application messages in the cluster

Typically, all cluster messages that leave the queue manager pass through the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, irrespective of which cluster sender channel is being used to transmit the message. Each channel is draining messages targeted for that channel in parallel with all other cluster sender channels. A growing build-up of messages on this queue can indicate a problem with one or more channels and must be investigated:

- The depth of the queue must be monitored appropriately for the cluster design.
- The following command returns all channels that have more than one message that is waiting on the transmit queue:

  ```
  DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
  ```

  With all cluster messages on a single queue, it is not always easy to see which channel has problems when it begins to fill up. Using this command is an easy way to see which channel is responsible.

You can configure a cluster queue manager to have multiple transmission queues. If you change the queue manager attribute `DEFCLXQ` to `CHANNEL`, every cluster-sender channel is associated with a different cluster transmit queue. Alternatively you can configure separate transmission queues manually. To display all the cluster transmit queues that are associated with cluster-sender channels, run the command:

```
DISPLAY CLUSQMGR (qmgrName) XMITQ
```

Define cluster transmission queues so that they follow the pattern of having the fixed stem of the queue name on the left. You can then query the depth of all the cluster transmission queues returned by the **`DISPLAY CLUSMGR`** command, by using a generic queue name:

```
DISPLAY QUEUE (qname *) CURDEPTH
```

Monitoring control messages in the cluster

The SYSTEM.CLUSTER.COMMAND.QUEUE queue is used for processing all cluster control messages for a queue manager, either generated by the local queue manager or sent to this queue manager from other queue managers in the cluster. When a queue manager is correctly maintaining its cluster state, this queue tends toward zero. There are situations where the depth of messages on this queue can temporarily grow however:

- Having lots of messages on the queue indicates churn in the cluster state.
- When making significant changes, allow the queue to settle in between those changes. For example, when moving repositories, allow the queue to reach zero before moving the second repository.

While a backlog of messages exists on this queue, updates to the cluster state or cluster-related commands are not processed. If messages are not being removed from this queue for a long time, further investigation is required, initially through inspection of the queue manager error logs z/OS (or CHINIT logs on z/OS) which might explain the process that is causing this situation.

The SYSTEM.CLUSTER.REPOSITORY.QUEUE holds the cluster repository cache information as a number of messages. It is usual for messages to always exist on this queue, and more for larger clusters. Therefore, the depth of messages on this queue is not an issue for concern.

Monitoring logs

Problems that occur in the cluster might not show external symptoms to applications for many days (and even months) after the problem originally occurs due to the caching of information and the distributed nature of clustering. However, the original problem is often reported in the IBM MQ error logs z/OS (and CHINIT logs on z/OS). For this reason, it is vital to actively monitor these logs for any messages written that relate to clustering. These messages must be read and understood, with any action taken where necessary.

For example: A break in communications with a queue manager in a cluster can result in knowledge of certain cluster resources that are being deleted due to the way that clusters regularly revalidate the cluster resources by republishing the information. A warning of such an event potentially occurring is reported by the message AMQ9465 z/OS or CSQX465I on z/OS systems. This message indicates that the problem needs to be investigated.

Special considerations for load balancing

When the cluster load balances between two or more instances of a queue, consuming applications must be processing messages on each of the instances. If one or more of those consuming applications terminates or stops processing messages, it is possible that clustering might continue to send messages to those instances of the queue. In this situation, those messages are not processed until the applications are functioning correctly again. For this reason the monitoring of the applications is an important part of the solution and action must be taken to reroute messages in that situation. An example of a mechanism to automate such monitoring can be found in this sample: The Cluster Queue Monitoring sample program (AMQSCLM).

Related concepts

"Tuning distributed publish/subscribe networks" on page 343
Use the tuning tips in this section to help improve the performance of your IBM MQ distributed publish/subscribe clusters and hierarchies.

"Balancing producers and consumers in publish/subscribe networks" on page 349

An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Monitoring transmission queue switching

It is important that you monitor the process of cluster-sender channels switching transmission queues so that the impact on your enterprise is minimized. For example, you should not attempt this process when the workload is high or by switching many channels simultaneously.

The process of switching channels

The process used to switch channels is:

1. The channel opens the new transmission queue for input and starts getting messages from it (using get by correlation ID)
2. A background process is initiated by the queue manager to move any messages queued for the channel from its old transmission queue to its new transmission queue. While messages are being moved any new messages for the channel are queued to the old transmission queue to preserve sequencing. This process might take a while to complete if there are a large number of messages for the channel on its old transmission queue, or new messages are rapidly arriving.
3. When no committed or uncommitted messages remain queued for the channel on its old transmission queue then the switch is completed. New messages are now put directly to the new transmission queue.

To avoid the eventuality of numerous channels switching simultaneously IBM MQ provides the ability to switch the transmission queue of one or more channels that are not running. On:

- IBM MQ for Multiplatforms the command is called **`runswchl`**
- IBM MQ for z/OS the CSQUTIL utility can be used to process a SWITCH CHANNEL command instead

Monitoring the status of switch operations

To understand the status of switch operations administrators can perform the following actions:

- Monitor the queue manager error log (AMQERR01.LOG) where messages are output to indicate the following stages during the operation:
  - The switch operation has started
  - The moving of messages has started
  - Periodic updates on how many messages are left to move (if the switch operation does not complete quickly)
  - The moving of messages has completed
  - The switch operation has completed

  On z/OS, these messages are output to the queue manager job log, not the channel initiator job log, although a single message is output by a channel to the channel initiator job log if it initiates a switch when starting.
- Use the DISPLAY CLUSQMGR command to query the transmission queue that each cluster-sender channel is currently using.
- Run the **`runswchl`** command (or CSQUTIL on z/OS) in query mode to ascertain the switching status of one or more channels. The output of this command identifies the following for each channel:
  - Whether the channel has a switch operation pending
  - Which transmission queue the channel is switching from and to
  - How many messages remain on the old transmission queue

Each command is really useful, because in one invocation you can determine the status of every channel, the impact a configuration change has had and whether all switch operations have completed.

Potential issues that might occur

See Potential issues when switching transmission queues for a list of some issues that might be encountered when switching transmission queue, their causes, and most likely solutions.

Related concepts

"Tuning distributed publish/subscribe networks" on page 343
Use the tuning tips in this section to help improve the performance of your IBM MQ distributed publish/subscribe clusters and hierarchies.

"Balancing producers and consumers in publish/subscribe networks" on page 349
An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Monitoring performance and resource usage

Use this topic to understand the facilities available to monitor the performance, and resource usage of your IBM MQ for z/OS subsystems.

Related information

Configuring z/OS

Administering IBM MQ for z/OS

z/OS Introduction to monitoring

Use this topic as an overview of the monitoring facilities available for IBM MQ for z/OS. For example, obtaining snapshots, using IBM MQ trace, online monitoring, and events.

This topic describes how to monitor the performance and resource usage of IBM MQ.

- It outlines some of the information that you can retrieve and briefly describes a general approach to investigating performance problems. z/OS (You can find information about dealing with performance problems in the Problem determination on z/OS .)
- It describes how you can collect statistics about the performance of IBM MQ by using SMF records.
- It describes how to gather accounting data to enable you to charge your customers for their use of your IBM MQ systems.
- It describes how to use IBM MQ events (alerts) to monitor your systems.

Here are some of the tools you might use to monitor IBM MQ; they are described in the sections that follow:

- Tools provided by IBM MQ:
  - Using DISPLAY commands
  - "Using CICS adapter statistics" on page 304
  - "Using IBM MQ events" on page 306
- z/OS service aids:
  - "Using System Management Facility" on page 306
- Other IBM licensed programs:
  - Using the Resource Measurement Facility
  - Using Tivoli Decision Support for z/OS
  - Using the CICS monitoring facility

Information about interpreting the data gathered by the performance statistics trace is given in"Interpreting IBM MQ performance statistics" on page 309.

Information about interpreting the data gathered by the accounting trace is given in"Interpreting IBM MQ accounting data" on page 331.

Getting snapshots of IBM MQ using the DISPLAY commands

IBM MQ provides the MQSC facility which can give a snapshot of the performance, and resource usage using the DISPLAY commands.

You can get an idea of the current state of IBM MQ by using the DISPLAY commands and, for the CICS adapter, the CICS adapter panels.

Using DISPLAY commands

You can use the IBM MQ MQSC DISPLAY or PCF Inquire commands to obtain information about the current state of IBM MQ. They provide information about the status of the command server, process definitions, queues, the queue manager, and its associated components. These commands are:

MQSC command	PCF command
DISPLAY ARCHIVE	Inquire Archive
DISPLAY AUTHINFO	Inquire Authentication Information Object
DISPLAY CFSTATUS	Inquire CF Structure Status
DISPLAY CFSTRUCT	Inquire CF Structure
DISPLAY CHANNEL	Inquire Channel
DISPLAY CHINIT	Inquire Channel Initiator
DISPLAY CHSTATUS	Inquire Channel Status
DISPLAY CMDSERV	
DISPLAY CLUSQMGR	Inquire Cluster Queue Manager
DISPLAY CONN	Inquire Connection
DISPLAY GROUP	Inquire Group
DISPLAY LOG	Inquire Log
DISPLAY PROCESS	Inquire Process
DISPLAY QMGR	Inquire Queue Manager
DISPLAY QSTATUS	Inquire Queue Status
DISPLAY QUEUE	Inquire Queue
DISPLAY SECURITY	Inquire Security
DISPLAY STGCLASS	Inquire Storage Class
DISPLAY SYSTEM	Inquire System
DISPLAY TRACE	
DISPLAY USAGE	Inquire Usage

For the detailed syntax of each command, see MQSC commands or PCF commands. All of the functions of these commands (except DISPLAY CMDSERV and DISPLAY TRACE) are also available through the operations and control panels.

These commands provide a snapshot of the system *only* at the moment the command was processed. If you want to examine trends in the system, you must start an IBM MQ trace and analyze the results over a period of time.

Using CICS adapter statistics

If you are an authorized CICS user, you can use the CICS adapter control panels to display CICS adapter statistics dynamically.

These statistics provide a snapshot of information related to CICS thread usage and situations when all threads are busy. The display connection panel can be refreshed by pressing the Enter key. For more information, see "The CICS-IBM MQ Adapter" section in the CICS Transaction Server for z/OS Version 4.1 product documentation at: CICS Transaction Server for z/OS Version 4.1, The CICS-IBM MQ adapter.

Using IBM MQ trace

You can record performance statistics and accounting data for IBM MQ by using the IBM MQ trace facility. Use this topic to understand how to control IBM MQ trace.

The data generated by IBM MQ is sent to:

- The System Management Facility (SMF), specifically as SMF record type 115, subtypes 1 and 2 for the performance statistics trace
- The SMF, specifically as SMF record type 116, subtypes zero, 1, and 2 for the accounting trace.

If you prefer, the data generated by the IBM MQ accounting trace can also be sent to the generalized trace facility (GTF).

Starting IBM MQ trace

You can start the IBM MQ trace facility at any time by issuing the IBM MQ START TRACE command.

Accounting data can be lost if the accounting trace is started or stopped while applications are running. To collect accounting data successfully, the following conditions must apply:

- The accounting trace must be active when an application starts, and it must still be active when the application finishes.
- If the accounting trace is stopped, any accounting data collection that was active stops.

You can also start collecting some trace information automatically if you specify YES on the SMFSTAT (SMF STATISTICS) and SMFACCT (SMF ACCOUNTING) parameters of the CSQ6SYSP macro. z/OS These parameters are described in Using CSQ6SYSP.

You cannot use this method to start collecting class 3 accounting information (thread-level and queue-level accounting). You must use the START TRACE command to collect such information. However, you can include the command in your CSQINP2 input data set so that the trace is started automatically when you start your queue manager.

Before starting an IBM MQ trace, read “Using System Management Facility” on page 306.

Controlling IBM MQ trace

To control the IBM MQ trace data collection at start-up, specify values for the parameters in the CSQ6SYSP macro when you customize IBM MQ. z/OS See Using CSQ6SYSP for details.

You can control IBM MQ tracing when the queue manager is running with these commands:

- START TRACE
- ALTER TRACE
- STOP TRACE

You can choose the destination to which trace data is sent. Possible destinations are:

SMF
System Management Facility

GTF
Generalized Trace Facility (accounting trace only)

SRV
Serviceability routine for diagnostic use by IBM service personnel

For daily monitoring, information is sent to SMF (the default destination). SMF data sets typically contain information from other systems; this information is not available for reporting until the SMF data set is dumped.

You can also send accounting trace information to the GTF. This information has an event identifier of 5EE. z/OS The The MQI call and user parameter, and z/OS generalized trace facility (GTF) describes how to deal with IBM MQ trace information sent to the GTF.

For information about IBM MQ commands, see MQSC commands.

Effect of trace on IBM MQ performance

Using the IBM MQ trace facility can have a significant effect on IBM MQ and transaction performance. For example, if you start a global trace for class 1 or for all classes, it is likely to increase processor usage and transaction response times by approximately 50%. However, if you start a global trace for classes 2 - 4 alone, the increase in processor usage and transaction response times is likely to be less than 1% additional processor cost to the cost of IBM MQ calls. The same applies for a statistics or accounting trace.

Using IBM MQ online monitoring

You can collect monitoring data for queues and channels (including automatically defined cluster-server channels) by setting the MONQ, MONCHL, and MONACLS attributes.

Table 33 on page 305 summarizes the commands to set these attributes at different levels and to display the monitoring information.

Table 33. Setting and displaying attributes to control online monitoring

Attribute	Applicable at this level	Set using command	Display monitoring information using command
MONQ	Queue	DEFINE QLOCAL DEFINE QMODEL ALTER QLOCAL ALTER QMODEL	DISPLAY QSTATUS
	Queue manager	ALTER QMGR	
MONCHL	Channel	DEFINE CHANNEL ALTER CHANNEL	DISPLAY CHSTATUS
	Queue manager	ALTER QMGR	
MONACLS	Queue manager	ALTER QMGR	

For full details of these commands, see MQSC commands. For more information about online monitoring, see “Monitoring your IBM MQ network” on page 18.

Using IBM MQ events

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can monitor the operation of all your queue managers by incorporating these events into your own system management application.

IBM MQ instrumentation events fall into the following categories:

Queue manager events
: These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events
: These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached, or the queue was not serviced within a predefined time limit.

Channel events
: These events are reported by channels as a result of conditions detected during their operation. For example, a channel instance is stopped.

Configuration events
: These events are notifications that an object has been created, changed, or deleted.

When an event occurs, the queue manager puts an *event message* on the appropriate *event queue*, if defined. The event message contains information about the event that can be retrieved by a suitable IBM MQ application.

IBM MQ events can be enabled using the IBM MQ commands or the operations and control panels.

See “Event types” on page 21 for information about the IBM MQ events that generate messages, and for information about the format of these messages. See Event message reference for information about enabling the events.

z/OS *Using System Management Facility*

You can use SMF to collect statistics and accounting information. To use SMF, certain parameters must be set in z/OS and in IBM MQ.

System management facility (SMF) is a z/OS service aid used to collect information from various z/OS subsystems. This information is dumped and reported periodically, for example, hourly. You can use SMF with the IBM MQ trace facility to collect data from IBM MQ. In this way you can monitor *trends*, for example, in system utilization and performance, and collect accounting information about each user ID using IBM MQ.

To record performance statistics (record type 115) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB or with the SETSMF z/OS operator command.

```
SYS(TYPE(115))
```

To record accounting information (record type 116) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB or with the SETSMF z/OS operator command.

```
SYS(TYPE(116))
```

You can turn on or off the recording of accounting information at the queue or queue manager level using the ACCTQ parameter of the DEFINE QLOCAL, DEFINE QMODEL, ALTER QLOCAL, ALTER QMODEL, or ALTER QMGR commands. See MQSC commands for details of these commands.

To use the z/OS command SETSMF, either PROMPT(ALL) or PROMPT(LIST) must be specified in the SMFPRM *xx* member. See the z/OS MVS Initialization and Tuning Reference and the z/OS MVS System Management Facilities (SMF) manuals for more information.

You can start collecting some trace information automatically if you specify YES on the SMFSTAT (SMF STATISTICS) and SMFACCT (SMF ACCOUNTING) parameters of the CSQ6SYSP macro; this is described in Using CSQ6SYSP.

Specifying YES on the SMFSTAT and SMFACCT parameters enables you to collect trace information as a queue manager starts.

You can also start collection of the data on a queue manager by specifying START TRACE(A) or START TRACE(S).

You can specify the interval at which IBM MQ collects statistics and accounting data in one of two ways:

- You can collect statistics data and accounting data at the same interval by specifying a value for STATIME in your system parameters (described in Using CSQ6SYSP).
- You can collect statistics data and accounting data at the SMF global accounting interval by specifying zero for STATIME (described in the z/OS MVS Initialization and Tuning Reference).

SMF must be running before you can send data to it. For more information about SMF, see the z/OS MVS System Management Facilities (SMF) manual.

For the statistics and accounting data to be reset, at least one MQI call must be issued during the accounting interval.

Allocating additional SMF buffers

When you start a trace, you must ensure that you allocate adequate SMF buffers. Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. IBM MQ sends a CSQW133I message to the z/OS console when this occurs. IBM MQ treats the error as temporary and remains active even though SMF data can be lost. When the shortage has been alleviated and trace recording has resumed, IBM MQ sends a CSQW123I message to the z/OS console.

Reporting data in SMF

You can use the SMF program IFASMFDP (or IFASMFDL if logstreams are being used) to dump SMF records to a sequential data set so that they can be processed.

There are several ways to report on this data, for example:

- Write an application program to read and report information from the SMF data set. You can then tailor the report to fit your exact needs.
- Use Performance Reporter to process the records. For more information, see "Using other products with IBM MQ" on page 307.

Using other products with IBM MQ

You can use other products to help you to improve the presentation of, or to augment statistics related to, performance and accounting. For example, Resource Measurement Facility, Tivoli Decision Support, and CICS monitoring.

Using the Resource Measurement Facility

Resource Measurement Facility (RMF) is an IBM licensed program (program number 5685-029) that provides system-wide information about processor utilization, I/O activity, storage, and paging. You can use RMF to monitor the utilization of physical resources across the whole system dynamically. For more information, see the *MVS Resource Measurement Facility User's Guide*.

Using Tivoli Decision Support for z/OS

You can use Tivoli Decision Support for z/OS to interpret RMF and SMF records.

Tivoli Decision Support for z/OS is an IBM licensed program (program number 5698-B06) that enables you to manage the performance of your system by collecting performance data in a Db2® database and presenting the data in various formats for use in systems management. Tivoli Decision Support for can generate graphic and tabular reports using systems management data it stores in its Db2 database. It includes an administration dialog, a reporting dialog, and a log collector, all of which interact with a standard Db2 database.

This is described in the *Tivoli Decision Support Administrator's Guide*.

Using the CICS monitoring facility

The CICS monitoring facility provides performance information about each CICS transaction running. It can be used to investigate the resources used and the time spent processing transactions. For background information, see the *CICS Performance Guide* and the *CICS Customization Guide*.

Investigating performance problems

Performance problems can arise from various factors. For example, incorrect resource allocation, poor application design, and I/O restraints. Use this topic to investigate some of the possible causes of performance problems.

Performance can be adversely affected by:

- Buffer pools that are an incorrect size
- Lack of real storage
- I/O contention for page sets or logs
- Log buffer thresholds that are set incorrectly
- Incorrect setting of the number of log buffers
- Large messages
- Units of recovery that last a long time, incorporating many messages for each sync point
- Messages that remain on a queue for a long time
- RACF® auditing
- Unnecessary security checks
- Inefficient program design

When you analyze performance data, always start by looking at the overall system before you decide that you have a specific IBM MQ problem. Remember that almost all symptoms of reduced performance are magnified when there is contention. For example, if there is contention for DASD, transaction response times can increase. Also, the more transactions there are in the system, the greater the processor usage and greater the demand for both virtual and real storage.

In such situations, the system shows heavy use of *all* its resources. However, the system is actually experiencing normal system stress, and this stress might be hiding the cause of a performance reduction. To find the cause of such a loss of performance, you must consider all items that might be affecting your active tasks.

Investigating the overall system

Within IBM MQ, the performance problem is either increased response time or an unexpected and unexplained heavy use of resources. First check factors such as total processor usage, DASD activity, and paging. An IBM tool for checking total processor usage is resource management facility (RMF). In general, you must look at the system in some detail to see why tasks are progressing slowly, or why a specific resource is being heavily used.

Start by looking at general task activity, then focus on particular activities, such as specific tasks or a specific time interval.

Another possibility is that the system has limited real storage; therefore, because of paging interrupts, the tasks progress more slowly than expected.

Investigating individual tasks

You can use the accounting trace to gather information about IBM MQ tasks. These trace records tell you a great deal about the activity that the task has performed, and about how much time the task spent suspended, waiting for latches. The trace record also includes information about how much Db2 and coupling facility activity were performed by the task.

Interpreting IBM MQ accounting data is described in "Interpreting IBM MQ accounting data" on page 331.

Long running units of work can be identified by the presence of message CSQR026I in the job log. This message indicates that a task has existed for more than three queue manager checkpoints and its log records have been shunted. z/OS For a description of log record shunting, see The log files.

Interpreting IBM MQ performance statistics

Use this topic as an index to the different SMF records created by IBM MQ for z/OS.

IBM MQ performance statistics are written as SMF type 115 records. Statistics records are produced periodically at a time interval specified by the STATIME parameter of the CSQ6SYSP system parameter module, or at the SMF global accounting interval if you specify zero for STATIME. The information provided in the SMF records comes from the following components of IBM MQ:

Buffer manager	Manages the buffer pools in virtual storage and the writing of pages to page sets as the buffer pools become full. Also manages the reading of pages from page sets.
Coupling facility manager	Manages the interface with the coupling facility.
Data manager	Manages the links between messages and queues. It calls the buffer manager to process the pages with messages on them.
Db2 manager	Manages the interface with the Db2 database that is used as the shared repository.
Lock manager	Manages locks for IBM MQ for z/OS.
Log manager	Manages the writing of log records, which are essential for maintaining the integrity of the system if there is a back out request, or for recovery, if there is a system or media failure.
Message manager	Processes all IBM MQ API requests.
Storage manager	Manages storage for IBM MQ for z/OS, for example, storage pool allocation, expansion, and deallocation.
Topic manager	Manages the Topic and Subscription information for IBM MQ for z/OS.
Coupling facility SMDS manager	Manages the shared message data sets (SMDS) for large messages stored in the coupling facility.

IBM MQ statistics are written to SMF as SMF type 115 records. The following subtypes can be present:

1
System information, for example, related to the logs and storage.

2
Information about number of messages and paging information. Queue sharing group information related to the coupling facility and Db2.

5 and 6
Detailed information about internal storage usage in the queue manager address space. While you can view this information, some of it is intended only for IBM use.

7
Storage manager summary information. While you can view this information, some of it is intended only for IBM use.

215
Buffer pool information

231
System information for the channel initiator address space.

Note that:

- Subtype 1, 2, and 215 records are created with statistics trace class 1.
- Subtype 7 records are created with statistics trace class 2.
- Subtype 5 and 6 records are created with statistics trace class 3.
- Subtype 231 records are created with statistics trace class 4.

The subtype is specified in the SM115STF field (shown in Table 34 on page 310).

Layout of an SMF type 115 record

You can use this section as a reference for the format of an SMF type 115 record.

The standard layout for SMF records involves three parts:

SMF header
Provides format, identification, and time and date information about the record itself.

Self-defining section
Defines the location and size of the individual data records within the SMF record.

Data records
The actual data from IBM MQ that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

Related reference

"The SMF header" on page 310
Use this topic as a reference for the format of the SMF header.

"Self-defining sections" on page 311
Use this topic as a reference for format of the self-defining sections of the SMF record.

"Examples of SMF statistics records" on page 312
Use this topic to understand some example SMF records.

The SMF header

Use this topic as a reference for the format of the SMF header.

Table 34 on page 310 shows the format of SMF record header (SM115).

Table 34. SMF record 115 header description

Offset: Dec	Offset: Hex	Type	Len	Name	Description	Example
0	0	Structure	28	SM115	SMF record header.	
0	0	Integer	2	SM115LEN	SMF record length.	14A0
2	2		2		Reserved.	
4	4	Integer	1	SM115FLG	System indicator.	5E

Table 34. SMF record 115 header description (continued)						
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**	**Example**
5	5	Integer	1	SM115RTY	Record type. The SMF record type, for IBM MQ statistics records this is always 115 (X'73').	73
6	6	Integer	4	SM115TME	Time when SMF moved record.	00355575
10	A	Integer	4	SM115DTE	Date when SMF moved record.	0100223F
14	E	Character	4	SM115SID	z/OS subsystem ID. Defines the z/OS subsystem on which the records were collected.	D4E5F4F1 (MV41)
18	12	Character	4	SM115SSI	IBM MQ subsystem ID.	D4D8F0F7 (MQ07)
22	16	Integer	2	SM115STF	Record subtype.	0002
24	18	Character	3	SM115REL	IBM MQ version.	F6F0F0 (600)
27	1B		1		Reserved	
28	1C	Character	0	SM115END	End of SMF header and start of self-defining section.	

Self-defining sections

Use this topic as a reference for format of the self-defining sections of the SMF record.

A self-defining section of a type 115 SMF record tells you where to find a statistics record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow the header, at fixed offsets from the start of the SMF record. Each statistics record can be identified by an eye-catcher string.

The following types of self-defining section are available to users for type 115 records. Each self-defining section points to statistics data related to one of the IBM MQ components. Table 35 on page 311 summarizes the sources of the statistics, the eye-catcher strings, and the offsets of the self-defining sections from the start of the SMF record header.

Table 35. Offsets to self-defining sections				
Source of statistics	**Record subtype (SM115STF)**	**Offset of self-defining section**		**Eye-catcher of data**
		Dec	**Hex**	
Storage manager	1	100	X'64'	QSST
Log manager	1	116	X'74'	QJST
Message manager	2	36	X'24'	QMST
Data manager	2	44	X'2C'	QIST
No longer used. The self-defining section will be binary zeros.	2	52	X'34'	
Lock manager	2	60	X'3C'	QLST
Db2 manager	2	68	X'44'	Q5ST

Table 35. Offsets to self-defining sections (continued)				
Source of statistics	**Record subtype (SM115STF)**	**Offset of self-defining section**		**Eye-catcher of data**
		Dec	**Hex**	
Coupling Facility manager	2	76	X'4C'	QEST
Topic manager	2	84	X'54'	QTST
SMDS usage	2	92	X'5C'	QESD
Buffer manager - one for each buffer pool	215	36	X'24'	QPST
Channel initiator	231			QWSX
Storage manager	5	36	X'24'	QSPH
Storage manager	6	36	X'24'	QSGM
Storage manager	7	36	X'24'	QSRS

Note: Some of the storage manager information in subtype 5, 6 and 7 records is intended only for IBM use. Other self-defining sections that are not listed contain data for IBM use only.

Each self-defining section is two fullwords long and has this format:

```
ssssssssllllnnnn
```

where:

- *ssssssss* is a fullword containing the offset from the start of the SMF record.
- *llll* is a halfword giving the length of this data record.
- *nnnn* is a halfword giving the number of data records in this SMF record.

For more information see, "Examples of SMF statistics records" on page 312.

Note: Always use offsets in the self-defining sections to locate the statistics records.

Examples of SMF statistics records

Use this topic to understand some example SMF records.

Figure 26 on page 313 shows an example of part of the SMF record for subtype 1. Subtype 1 includes the storage manager and log manager statistics records. The SMF record header is shown underlined.

The self-defining section at offset X'64' refers to storage manager statistics and the self-defining section at offset X'74' refers to log manager statistics, both shown in **bold**.

The storage manager statistics record is located at offset X'0000011C' from the start of the header and is X'48' bytes long. There is one set of storage manager statistics, identified by the eye-catcher string QSST. The start of this statistics record is also shown in the example.

The log manager statistics record is located at offset X'00000164' from the start of the header and is X'78' bytes long. There is one set of log manager statistics, identified by the eye-catcher string QJST.

```
000000  02000000 5E730035 55750100 223FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70001 F6F0F000 000001DC  *41MQ07..600.....*
000020  00240001 00000000 00000000 00000000  *................*
000030  00000000 00000000 00000000 0000007C  *...............@*
000040  00400001 000000BC 00600001 00000000  *. ......-......*
000050  00000000 00000000 00000000 00000000  *................*
000060  00000000 0000011C 00480001 00000000  *................*
000070  00000000 00000164 00780001 00000000  *................*
000080  00000000 00000000 00000000 00000000  *................*
.
.
000110  00000000 00000000 00000000 003C0048  *................*
000120  D8E2E2E3 0000004F 00000003 00000002  *QSST...|........*
```

Figure 26. SMF record 115, subtype 1

Figure 27 on page 314 shows an example of part of the SMF record for subtype 2. Subtype 2 includes the statistics records for the message, data, lock, coupling facility, topic, and Db2 managers. The SMF record header is shown underlined; the self-defining sections are shown alternately **bold** and *italic*.

- The self-defining section at offset X'24' refers to message manager statistics. The message manager statistics record is located at offset X'00000064' from the start of the header and is X'48' bytes long. There is one set of these statistics, identified by the eye-catcher string QMST.
- The self-defining section at offset X'2C' refers to data manager statistics. The data manager statistics record is located at offset X'000000AC' from the start of the header and is X'50' bytes long. There is one set of these statistics, identified by the eye-catcher string QIST.
- The self-defining section at offset X'34' refers to buffer manager statistics. As this SMF record was taken from a queue manager that has OPMODE(NEWFUNC,800) set in its system parameters, the buffer manager self-defining section is set to zeros to indicate that there are no buffer manager statistics. Instead, these statistics are in SMF 115 subtype 215 records.
- The self-defining section at offset X'3C' refers to lock manager statistics. The lock manager statistics record is located at offset X'000000FC' from the start of the header and is X'20' bytes long. There is one set of these statistics, identified by the eye-catcher string QLST.
- The self-defining section at offset X'44' refers to Db2 manager statistics. The Db2 manager statistics record is located at offset X'0000011C' from the start of the header and is X'2A0' bytes long. There is one set of these statistics, identified by the eye-catcher string Q5ST.
- The self-defining section at offset X'4C' refers to coupling facility manager statistics. The coupling facility manager statistics record is located at offset X'000003BC' from the start of the header and is X'1008' bytes long. There is one set of these statistics, identified by the eye-catcher string QEST.
- The self-defining section at offset X'54' refers to topic manager statistics. The topic manager statistics record is located at offset X'000013C4' from the start of the header and is X'64' bytes long. There is one set of these statistics, identified by the eye-catcher string QTST.
- The self-defining section at offset X'5C' is for SMDS statistics. This self defining section is set to zeros indicating that SMDS is not being used.

```
000000 09F40000 5E730033 4DBE0113 142FD4E5 *.4..;...(.....MV*
000010 F4F1D4D8 F2F10002 F8F0F000 00001428 *41MQ21..800.....*
000020 00240001 00000064 00480001 000000AC *................*
000030 00500001 00000000 00000000 000000FC *................*
000040 00200001 0000011C 02A00001 000003BC *................*
000050 10080001 000013C4 00640001 00000000 *.......D........*
000060 00000000 D40F0048 D8D4E2E3 00000000 *....M...QMST....*
000080 00000000 00000000 00000000 00000000 *................*
000090 00000000 00000000 00000000 00000000 *................*
0000A0 00000000 00000000 00000000 C90F0050 *............I..&*
0000B0 D8C9E2E3 00000000 00000000 00000000 *QIST............*
0000C0 00000000 00000000 00000000 00000000 *................*
0000D0 00000000 00000000 00000000 00000000 *................*
0000E0 00000000 00000000 00000000 00000000 *................*
0000F0 00000000 00000000 00000000 D30F0020 *............L...*
000100 D8D3E2E3 00000000 00000000 00000000 *QLST............*
000110 00000000 00000000 00000000 F50F02A0 *............5...*
000120 D8F5E2E3 00000008 00000000 00000000 *Q5ST............*
.
.
```

Figure 27. SMF record 115, subtype 2

Processing type 115 SMF records

Use this topic as a reference for processing type 115 SMF records.

You must process any data you collect from SMF to extract useful information. When you process the data, verify that the records are from IBM MQ and that they are the records you are expecting.

Validate the values of the following fields:

- SM115RTY, the SMF record number, must be X'73' (115)
- SM115STF, the record subtype, must be 0001, 0002, 0005, 0006, 0007, 0215, or 0231

Reading from the active SMF data sets (or SMF logstreams) is not supported. You must use the SMF program IFASMFDP (or IFASMFDL if logstreams are being used) to dump SMF records to a sequential data set so that they can be processed. For more information see "Using System Management Facility" on page 306.

There is a C sample program called CSQ4SMFD which prints the contents of SMF type 115 and 116 records from the sequential data set. The program is provided as source in thlqual.SCSQC37S and in executable format in thlqual.SCSQLOAD. Sample JCL is provided in thlqual.SCSQPROC(CSQ4SMFJ).

Storage manager data records

Use this topic as a reference for storage manager data records.

The format of the storage manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQSST).

The data contains information about the number of fixed and variable storage pools that the queue manager has allocated, expanded, contracted, and deleted during the statistics interval, plus the number of GETMAIN, FREEMAIN, and STORAGE requests to z/OS, including a count of those requests that were unsuccessful. Additional information includes a count of the number of times the short-on-storage condition was detected and a count of the number of abends that occurred as a result of that condition.

Additional data about storage usage in the queue manager is produced by class 2 and class 3 statistics trace. While you can view this information, some of it is intended only for IBM use.

- The format of the storage manager pool header statistics record, which is present in subtype 5 records, is described in assembler macro thlqual.SCSQMACS(CSQDQSPH).
- The format of the storage manager getmain statistics record, which is present in subtype 6 records, is described in assembler macro thlqual.SCSQMACS(CSQDQSGM).

- The format of the storage manager region summary record, which is present in subtype 7 records, is described in assembler macro thlqual.SCSQMACS(CSQDQSRS).

Log manager data records

Use this topic as a reference for format of log manager data records.

The format of the log manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQJST).

In the statistics, these counts are important:

1. The total number of log write requests:

   $$N_{logwrite} = QJSTWRW + QJSTWRNW + QJSTWRF$$

2. The total number of log read requests:

   $$N_{logread} = QJSTRBUF + QJSTRACT + QJSTRARH$$

The problem symptoms that can be examined using log manager statistics are described in the following table.

Symptom 1
QJSTWTB is nonzero.

Reason
Tasks are being suspended while the in-storage buffer is being written to the active log.

There might be problems writing to the active log.

The OUTBUFF parameter within CSQ6LOGP is too small.

Action
Investigate the problems writing to the active log.

Increase the value of the OUTBUFF parameter within CSQ6LOGP.

Symptom 2
The ratio: $QJSTWTL/N_{logread}$ is greater than 1%.

Reason
Log reads were initiated that had to read from an archive log, but IBM MQ could not allocate a data set because MAXRTU data sets were already allocated.

Action
Increase MAXRTU.

Symptom 3
The ratio: $\mathrm{QJSTRARH}/N_{\mathrm{logread}}$ is larger than normal.

Reason
Most log read requests should come from the output buffer or the active log. To satisfy requests for back out, unit-of-recovery records are read from the in-storage buffer, the active log, and the archived logs.

A long-running unit of recovery, extending over a period of many minutes, might have log records spread across many different logs. This degrades performance because extra work has to be done to recover the log records.

Action
Change the application to reduce the length of a unit of recovery. Also, consider increasing the size of the active log to reduce the possibility of a single unit of recovery being spread out over more than one log.

Other pointers
The ratio $N_{\mathrm{logread}}/N_{\mathrm{logwrite}}$ gives an indication of how much work has to be backed out.

Symptom 4
QJSTLLCP is more than 10 an hour.

Reason
On a busy system, you would expect to see typically 10 checkpoints an hour. If the QJSTLLCP value is larger than this, it indicates a problem in the setup of the queue manager.

The most likely reason for this is that the LOGLOAD parameter in CSQ6SYSP is too small. The other event that causes a checkpoint is when an active log fills up and switches to the next active log data set. If your logs are too small, this can cause frequent checkpoints. The QJSTLLCP counter is not incremented for log switch induced checkpoints; you must look in the JES logs for the queue managers to determine if the rate log files are switched.

Action
Increase the LOGLOAD parameter, or increase the size of your log data sets as required.

Symptom 5
QJSTCmpFail > 0 or QJSTCmpComp not much less than QJSTCmpUncmp

Reason

The queue manager is unable to significantly compress log records.

QJSTCmpFail is the number of times the queue manager was unable to achieve any reduction in record length. You should compare the number to QJSTCmpReq (number of compression requests) to see if the number of failures is significant.

QJSTCmpComp is the total of compressed bytes written to the log and QJSTCmpUncmp is the total bytes before compression. Neither total contains bytes written for log records that were not eligible for compression. If the numbers are similar then compression has achieved little benefit.

Action
Turn off log compression. Issue the SET LOG COMPLOG(NONE) command. See the SET LOG command for details.

Note: In the first set of statistics produced after system startup, there might be significant log activity due to the resolution of in-flight units of recovery.

Message manager data records

Use this topic as a reference for message manager data records.

The format of the message manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQMST).

The data gives you counts of different IBM MQ API requests.

Data manager data records

Use this topic as a reference for the format of the Data Manager data records.

The format of the data manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQIST).

The data gives you counts of different object requests.

Buffer manager data records

Use this topic as a reference for the format of buffer manager data records.

The format of the buffer manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQPST).

Note: Buffer manager statistics records will only be created for buffer pools that are defined. If a buffer pool is defined but not used then no values will be set and its buffer manager statistics record will not contain any data.

For information about efficiently managing your buffer pools, see "Managing your buffer pools" on page 318.

When interpreting the statistics, you are recommended to consider the following factors because the values of these fields can be used to improve the performance of your system:

1. If QPSTSOS, QPSTDMC, or QPSTIMW is greater than zero, you should either increase the size of the buffer pool or reallocate the page sets to different buffer pools.
   - QPSTSOS is the number of times that there were no buffers available for page get requests. If QPSTSOS ever becomes nonzero, it shows that IBM MQ is under severe stress. The buffer pool size should be increased. If increasing the buffer pool size does not make the value of QPSTSOS zero, there might be I/O contention on the DASD page sets.
   - QPSTDMC is the number of updates that were performed synchronously because there was either more than 95% of the pages in the buffer pool waiting for write I/O, or there was less than 5% of the buffer pool available for read requests. If this number is not zero, the buffer pool might be too small and should be enlarged. If increasing the buffer pool size does not reduce QPSTDMC to zero, there might be I/O contention on the DASD page sets.
   - QPSTIMW is a count of the number of times pages were written out synchronously. If QPSTDMC is zero, QPSTIMW is the number of times pages were found on the queue waiting for write I/O that had been there for at least two checkpoints.
2. For buffer pool zero and buffer pools that contain short-lived messages:
   - QPSTDWT should be zero, and the percentage QPSTCBSL/QPSTNBUF should be greater than 15%.

     QPSTDWT is the number of times the asynchronous write processor was started because there was either more than 85% of the pages in the buffer pool waiting for write I/O, or there was less than 15% of the buffer pool available for read requests. Increasing the buffer pool size should reduce this value. If it does not, the pattern of access is one of long delays between puts and gets.
   - QPSTTPW might be greater than zero due to checkpointing activity.
   - QPSTRIO should be zero unless messages are being read from a page set after the queue manager is restarted.

     The ratio of QPSTRIO to QPSTGETP shows the efficiency of page retrieval within the buffer pool. Increasing the buffer pool size should decrease this ratio and, therefore, increase the page retrieval efficiency. If this does not happen, it indicates that pages are not being frequently reaccessed. This implies a transaction pattern where there is a long delay between messages being put and then later retrieved.

     The ratio of QPSTGETN to QPSTGETP indicates the number of times an empty page, as opposed to a non-empty page, has been requested. This ratio is more an indication of transaction pattern, than a value that can be used to tune the system.

- If QPSTSTL has a value greater than zero, this indicates that pages that have not been used before are now being used. This might be caused by an increased message rate, messages not being processed as fast as they were previously (leading to a buildup of messages), or larger messages being used.

  QPSTSTL is a count of the number of times a page access request did not find the page already in the buffer pool. Again, the lower the ratio of QPSTSTL to (QPSTGETP + QPSTGETN) is, the higher the page retrieval efficiency. Increasing the buffer pool size should decrease this ratio but, if it does not, it is an indication that there are long delays between puts and gets.
- You are recommended to have sufficient buffers to handle your peak message rate.

3. For buffer pools with long-lived messages, where there are more messages than can fit into the buffer pool:
   - (QPSTRIO+QPSTWIO)/Statistics interval is the I/O rate to page sets. If this value is high, you should consider using multiple page sets on different volumes to allow I/O to be carried out in parallel.
   - Over the period of time that the messages are processed (for example, if messages are written to a queue during the day and processed overnight) the number of read I/Os (QPSTRIO) should be approximately the total number of pages written (QPSTTPW). This shows that one page is read for every page written.

     If QPSTRIO is much larger than QPSTTPW, this shows that pages are being read in multiple times. This might be a result of the application using MQGET by *MsgId* or *CorrelId* when the queue is not indexed, or browsing messages on the queue using get next.

     The following actions might relieve this problem:

     a. Increase the size of the buffer pool so that there are enough pages to hold the queue, in addition to any changed pages.
     b. Use the INDXTYPE queue attribute, which allows a queue to be indexed by *MsgId* or *CorrelId* and eliminates the need for a sequential scan of the queue.
     c. Change the design of the application to eliminate the use of MQGET with *MsgId* or *CorrelId*, or the get next with browse option.

        Note: Applications using long-lived messages typically process the first available message and do not use MQGET with *MsgId* or *CorrelId*, and they might browse only the first available message.
     d. Move page sets to a different buffer pool to reduce contention between messages from different applications.

Managing your buffer pools
To manage your buffer pools efficiently, you must consider the factors that affect the buffer pool I/O operations and also the statistics associated with the buffer pools.

The following factors affect buffer pool I/O operations.

- If a page containing the required data is not found in the buffer pool, it is read in synchronously to an available buffer from its DASD page set.
- Whenever a page is updated, it is put on an internal queue of pages to be (potentially) written out to DASD. This means that the buffer used by that page is unavailable for use by any other page until the buffer has been written to DASD.
- If the number of pages queued to be written to DASD exceeds 85% of the total number of buffers in the pool, an asynchronous write processor is started to put the buffers to DASD.

  Similarly, if the number of buffers available for page get requests become less than 15% of the total number of buffers in the pool, the asynchronous write processor is started to perform the write I/O operations.

  The write processor stops when the number of pages queued to be written to DASD has fallen to 75% of the total number of buffers in the pool.
- If the number of pages queued for writing to DASD exceeds 95% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.

Similarly, if the number of buffers available for page get requests becomes less than 5% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.

- If the number of buffers available for page get requests ever reaches zero, a transaction that encounters this condition is suspended until the asynchronous write processor has finished.
- If a page is frequently updated, the page spends most of its time on the queue of pages waiting to be written to DASD. Because this queue is in least recently used order, it is possible that a frequently updated page placed on this least recently used queue is never written out to DASD. For this reason, at the time of update, if the page is found to have been waiting on the write operation to DASD queue for at least two checkpoints, it is synchronously written to DASD. Updating occurs at checkpoint time and is suspended until the asynchronous write processor has finished.

  The aim of this algorithm is to maximize the time pages spend in buffer pool memory while allowing the system to function if the system load puts the buffer pool usage under stress.

Lock manager data records

Use this topic as a reference to the format of the lock manager data records.

The format of the lock manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQLST).

The records contain data about the following information:

- The number of lock get requests and lock release requests.
- The number of times a lock get request determined that the requested lock was already held.

Db2 manager data records

Use this topic as a reference to the format of the Db2 manager data records.

The format of the Db2 manager statistics record is described in the following table and in assembler macro thlqual.SCSQMACS(CSQDQ5ST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example q5st, q5stid.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Table 36. Db2 statistics record (Q5ST)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
0	0	Structure	668	Q5ST	Db2 manager statistics
0	0	Bitstring	2	Q5STID	Control block identifier
2	2	Integer	2	Q5STLL	Control block length
4	4	Character	4	Q5STEYEC	Control block eye catcher
8	8	Character	660	Q5STZERO	QMST part cleared on occasion
8	8	Integer	4	NUMTASK	Number of server tasks
12	C	Integer	4	ACTTASK	Number of active server tasks
16	10	Integer	4	CONNCNT	Number of connect requests
20	14	Integer	4	DISCCNT	Number of disconnect requests
24	18	Integer	4	DHIGMAX	Max. request queue depth
28	1C	Integer	4	ABNDCNT	Number of Db2SRV task abends
32	20	Integer	4	REQUCNT	Number of requests requeued
36	24	Integer	4	DEADCNT	Number of deadlock timeouts

Offset: Dec	Offset: Hex	Type	Len	Name	Description
40	28	Integer	4	DELECNT	Number of delete requests
44	2C	Integer	4	LISTCNT	Number of list requests
48	30	Integer	4	READCNT	Number of read requests
52	34	Integer	4	UPDTCNT	Number of update requests
56	38	Integer	4	WRITCNT	Number of write requests
60	3C	Integer	4	SCSSEL	SCST (shared-channel-status) selects
64	40	Integer	4	SCSINS	SCST inserts
68	44	Integer	4	SCSUPD	SCST updates
72	48	Integer	4	SCSDEL	SCST deletes
76	4C	Integer	4	SSKSEL	SSKT (shared-sync-key) selects
80	50	Integer	4	SSKINS	SSKT inserts
84	54	Integer	4	SSKDEL	SSKT deletes
88	58	Integer	4	SCSBFTS	SCST number of times buffer too small
92	5C	Integer	4	SCSMAXR	SCST maximum rows on query
96	60	Integer	4	* (2)	Reserved
104	68	Character	8	DELETCUW	Cumulative STCK difference - Thread delete
112	70	Character	8	DELETMXW	Maximum STCK difference - Thread delete
120	78	Character	8	DELESCUW	Cumulative STCK difference - SQL delete
128	80	Character	8	DELESMXW	Maximum STCK difference - SQL delete
136	88	Character	8	LISTTCUW	Cumulative STCK difference - Thread list
144	90	Character	8	LISTTMXW	Maximum STCK difference - Thread list
152	98	Character	8	LISTSCUW	Cumulative STCK difference - SQL list
160	A0	Character	8	LISTSMXW	Maximum STCK difference - SQL list
168	A8	Character	8	READTCUW	Cumulative STCK difference - Thread read
17 6	B0	Character	8	READTMXW	Maximum STCK difference - Thread read
184	B8	Character	8	READSCUW	Cumulative STCK difference - SQL read
192	C0	Character	8	READSMXW	Maximum STCK difference - SQL read
200	C8	Character	8	UPDTTCUW	Cumulative STCK difference - Thread update
208	D0	Character	8	UPDTTMXW	Maximum STCK difference - Thread update
216	D8	Character	8	UPDTSCUW	Cumulative STCK difference - SQL update
224	E0	Character	8	UPDTSMXW	Maximum STCK difference - SQL update
232	E8	Character	8	WRITTCUW	Cumulative STCK difference - Thread write
240	F0	Character	8	WRITTMXW	Maximum STCK difference - Thread write
248	F8	Character	8	WRITSCUW	Cumulative STCK difference - SQL write

Table 36. Db2 statistics record (Q5ST) (continued)

Table 36. Db2 statistics record (Q5ST) (continued)					
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**
256	100	Character	8	WRITSMXW	Maximum STCK difference - SQL write
264	108	Character	8	SCSSTCUW	Cumulative STCK difference - Thread select
272	110	Character	8	SCSSTMXW	Maximum STCK difference - Thread select
280	118	Character	8	SCSSSCUW	Cumulative STCK difference - SQL select
288	120	Character	8	SCSSSMXW	Maximum STCK difference - SQL select
296	128	Character	8	SCSITCUW	Cumulative STCK difference - Thread insert
304	130	Character	8	SCSITMXW	Maximum STCK difference - Thread insert
312	138	Character	8	SCSISCUW	Cumulative STCK difference - SQL insert
320	140	Character	8	SCSISMXW	Maximum STCK difference - SQL insert
328	148	Character	8	SCSUTCUW	Cumulative STCK difference - Thread update
336	150	Character	8	SCSUTMXW	Maximum STCK difference - Thread update
344	158	Character	8	SCSUSCUW	Cumulative STCK difference - SQL update
352	160	Character	8	SCSUSMXW	Maximum STCK difference - SQL update
360	168	Character	8	SCSDTCUW	Cumulative STCK difference - Thread delete
368	170	Character	8	SCSDTMXW	Maximum STCK difference - Thread delete
376	178	Character	8	SCSDSCUW	Cumulative STCK difference - SQL delete
384	180	Character	8	SCSDSMXW	Maximum STCK difference - SQL delete
392	188	Character	8	SSKSTCUW	Cumulative STCK difference - Thread select
400	190	Character	8	SSKSTMXW	Maximum STCK difference - Thread select
408	198	Character	8	SSKSSCUW	Cumulative STCK difference - SQL select
416	1A0	Character	8	SSKSSMXW	Maximum STCK difference - SQL select
424	1A8	Character	8	SSKITCUW	Cumulative STCK difference - Thread insert
432	1B0	Character	8	SSKITMXW	Maximum STCK difference - Thread insert
440	1B8	Character	8	SSKISCUW	Cumulative STCK difference - SQL insert
448	1C0	Character	8	SSKISMXW	Maximum STCK difference - SQL insert
456	1C8	Character	8	SSKDTCUW	Cumulative STCK difference - Thread delete
464	1D0	Character	8	SSKDTMXW	Maximum STCK difference - Thread delete
472	1D8	Character	8	SSKDSCUW	Cumulative STCK difference - SQL delete
480	1E0	Character	8	SSKDSMXW	Maximum STCK difference - SQL delete
488	1E8	Integer	4	LMSSEL	Number of Db2 BLOB read requests
492	1EC	Integer	4	LMSINS	Number of Db2 BLOB insert requests
496	1F0	Integer	4	LMSUPD	Number of Db2 BLOB update requests
500	1F4	Integer	4	LMSDEL	Number of Db2 BLOB delete requests
504	1F8	Integer	4	LMSLIS	Number of Db2 BLOB list requests

Table 36. Db2 statistics record (Q5ST) (continued)					
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**
508	IFC	64 bit integer	8	LMSSTCUW	Total elapsed time for all thread read BLOB requests
516	204	64 bit integer	8	LMSSTMXW	Maximum elapsed time for a thread read BLOB request
524	20C	64 bit integer	8	LMSSSCUW	Total elapsed time for all SQL read BLOB requests
532	214	64 bit integer	8	LMSSSMXW	Maximum elapsed time for an SQL read BLOB request
540	21C	64 bit integer	8	LMSITCUW	Total elapsed time for all thread insert BLOB requests
548	224	64 bit integer	8	LMSITMXW	Maximum elapsed time for a thread insert BLOB request
556	22C	64 bit integer	8	LMSISCUW	Total elapsed time for all SQL insert BLOB requests
564	234	64 bit integer	8	LMSISMXW	Maximum elapsed time for an SQL insert BLOB request
572	23C	64 bit integer	8	LMSUTCUW	Total elapsed time for all thread update BLOB requests
580	244	64 bit integer	8	LMSUTMXW	Maximum elapsed time for a thread update BLOB request
588	24C	64 bit integer	8	LMSUSCUW	Total elapsed time for all SQL update BLOB requests
596	254	64 bit integer	8	LMSUSMXW	Maximum elapsed time for an SQL update BLOB request
604	25C	64 bit integer	8	LMSDTCUW	Total elapsed time for all thread delete BLOB requests
612	264	64 bit integer	8	LMSDTMXW	Maximum elapsed time for a thread delete BLOB request
620	26C	64 bit integer	8	LMSDSCUW	Total elapsed time for all SQL delete BLOB requests
628	274	64 bit integer	8	LMSDSMXW	Maximum elapsed time for an SQL delete BLOB request
636	27C	64 bit integer	8	LMSLTCUW	Total elapsed time for all thread list BLOB requests
644	284	64 bit integer	8	LMSLTMXW	Maximum elapsed time for a thread list BLOB request
652	28C	64 bit integer	8	LMSLSCUW	Total elapsed time for all SQL list BLOB requests
660	294	64 bit integer	8	LMSLSMXW	Maximum elapsed time for an SQL list BLOB request

The data contains counts for each request type that the Db2 resource manager supports. For these request types, maximum and cumulative elapse times are kept for the following:

- The time spent in the Db2 resource manager as a whole (called the thread time).
- The time that was spent performing the RRSAF and SQL parts of the request (a subset of the thread time called the SQL time).

Information is also provided for:

- The number of server tasks attached.
- The maximum overall request depth against any of the server tasks.
- The number of times any of the server task requests terminated abnormally.

If the abnormal termination count is not zero, a requeue count is provided indicating the number of queued requests that were requeued to other server tasks as a result of the abnormal termination.

If the average thread time is significantly greater that the average SQL time, this might indicate that thread requests are spending an excessive amount of time waiting for a server task to process the SQL part of the request. If this is the case, examine the DHIGMAX field and, if the value is greater than one, consider increasing the number of Db2 server tasks specified in the QSGDATA parameter of the CSQ6SYSP system parameter macro.

Coupling facility manager data records

Use this topic as a reference to the format of the coupling facility manager data records.

The format of the coupling facility manager statistics record is described in the following table and in assembler macro thlqual. SCSQMACS(CSQDQEST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qest, qestid.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Table 37. Coupling facility statistics record (QEST)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
0	0	Structure	4104	QEST	CF manager statistics
0	0	Bitstring	2	QESTID	Control block identifier
2	2	Integer	2	QESTLL	Control block length
4	4	Character	4	QESTEYEC	Control block eye catcher
8	8	Character	4096	QESTZERO	QEST part cleared on occasion
8	8	Character	64	QESTSTUC (0:63)	Array (one entry per structure)
8	8	Character	12	QESTSTR	Structure name
20	14	Integer	4	QESTSTRN	Structure number
24	18	Integer	4	QESTCSEC	Number of IXLLSTE calls
28	1C	Integer	4	QESTCMEC	Number of IXLLSTM calls
32	20	Character	8	QESTSSTC	Time spent doing IXLLSTE calls
40	28	Character	8	QESTMSTC	Time spent doing IXLLSTM calls
48	30	Integer	4	QESTRSEC	Number of IXLLSTE redrives
52	34	Integer	4	QESTRMEC	Number of IXLLSTM redrives

Table 37. Coupling facility statistics record (QEST) (continued)					
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**
56	38	Integer	4	QESTSFUL	Number of structure fulls
60	3C	Integer	4	QESTMNUS	Maximum number of entries in use
64	40	Integer	4	QESTMLUS	Maximum number of elements in use
68	44	Character	4	*	Reserved
4104	1008	Character	0	*	End of control block

The data contains information for each coupling facility list structure, including the CSQ_ADMIN structure, that the queue manager could connect to during the statistics interval. The information for each structure includes the following:

- The number of and cumulative elapsed times for IXLLSTE and IXLLSTM requests.
- The number of times a request had to be retried because of a timeout.
- The number of times a 'structure full' condition occurred.

Topic manager data records

Use this topic as a reference to the format of the topic manager data records.

The format of the Topic manager statistics record is described in the following table and in assembler macro thlqual.SCSQMACS(CSQDQTST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qtst, qtstid.

Table 38. Topic manager statistics record (QTST)					
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**
0	0	Structure	96	QTST	Topic manager statistics
0	0	Bitstring	2	QTSTID	Control block identifier
2	2	Integer	2	QTSTLL	Control block length
4	4	Character	4	TESTEYEC	Control block eye catcher
8	8	Character	88	QTSTZERO	QTST part cleared on occasion
8	8	Integer	4	QTSTSTOT	Total subscription requests
12	0C	Integer	4	QTSTSDUR	Durable subscription requests
16	10	Integer	4	QTSTSHIG (1:3)	Subscription high water mark array (API, ADMIN, PROXY)
28	1C	Integer	4	QTSTSLOW (1:3)	Subscription low water mark array (API, ADMIN, PROXY)
40	28	Integer	4	QTSTSEXP	Subscriptions expired
44	2C	Integer	4	QTSTTMSG	Total messages put to Sub queue
48	30	Integer	4	QTSTSPHW	Single publish subscriber high water mark
52	34	Integer	4	QTSTPTOT (1:3)	Total Publication requests (API, ADMIN, PROXY)

Table 38. Topic manager statistics record (QTST) (continued)					
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**
64	40	Integer	4	QTSTPTHI	Total publish high water mark
68	44	Integer	4	QTSTPTLO	Total publish low water mark
72	48	Integer	4	QTSTPNOS	Count of publishes to no subscriber
76	4C	Integer	4	*	Reserved
80	50	Bitstring	8	QTSTETHW	Elapse time HW on publish
88	58	Bitstring	8	QTSTETTO	Elapse time total on publish

Coupling facility manager SMDS data records

Use this topic as a reference to the format of the coupling facility manager shared message data set (SMDS) data records.

The format of the coupling facility manager shared message data set (SMDS) statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQESD), C header file thlqual.SCSQC370(CSQDSMFC) and in IBM MQ SupportPac MP1B.

The statistics provide information about the utilization of the owned shared message data set, I/O activity for the group of shared message data sets, and SMDS buffer utilization.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Layout of SMF records for the channel initiator

The layouts of channel accounting data (SMF type 116, subtype 10) and channel initiator statistics data (SMF type 115 , subtype 231 records) are described in this topic.

Processing the SMF data for the CHINIT

The data written to SMF is in the standard triplet format.

Accounting data SMF type 116, subtype 10

There is the standard SMF header.

The triplets are mapped by qws5 in csqdsmfc.h and csqdqws5.macro, and have the following layout:

4 bytes offset to the QWHS
2 bytes length of the QWHS
2 bytes count of the number of instances of QWHS
4 bytes offset to the QCST
2 bytes length of the QCST
2 bytes count of the number of instances of QCST

The QWHS mapped is mapped by csqdqwhs.macro and csqdsmfc.h, and has the following key fields:

- qwhsnsda 1 byte, count of the number of self defining section.
- qwhssmfc 1 bit. If this is on there are multiple SMF records containing information for this interval. If this is off, this is the last or only record.
- Qwhstime 8 bytes in STCK format. The local time of the start of the interval.
- qwhsdurn 8 bytes in STCK format. The duration from the start of the interval to the end of the interval.
- Qwhsstck 8 bytes STCK format. The end of the interval in GMT.

The QCST is mapped by csqdsmfc.h and csqdqcst.macro.

Statistics data SMF type 115, subtype 231

There is the standard SMF header.

The triplets are mapped by qwsx in csqdsmfc.h and csqdqwsx.macro, and have the following layout:

4 bytes offset to the QWHS
2 bytes length of the QWHS
2 bytes count of the number of instances of QWHS

CHINIT Control Information, number of channels. and so on is mapped by csqdsmfc.h and csqdqcct.macro:

4 bytes offset to the QCCT
2 bytes length of the QCCT
2 bytes count of the number of instances of the QCCT

Dispatcher tasks are mapped by csqdsmfc.h and the QCT_DSP structure in the CSQDQCTA macro:

4 bytes offset to the QCT_DSP
2 bytes length of the QCT_DSP
2 bytes count of the number of instances of the QCT_DSP

Adapter tasks are mapped by csqdsmfc.h and the QCT_ADP structure in the CSQDQCTA macro:

4 bytes offset to the QCT_ADP
2 bytes length of the QCT_ADP
2 bytes count of the number of instances of QCT_ADP

SSL tasks are mapped by csqdsmfc.h and the QCT_SSL structure in the CSQDQCTA macro:

4 bytes offset to the QCT_SSL
2 bytes length of the QCT_SSL
2 bytes count of the number of instances of QCT_SSL

DNS task is mapped by csqdsmfc.h and the QCT_DNS structure in the CSQDQCTA macro:

4 bytes offset to the QCT_DNS
2 bytes length of the QCT_DNS
2 bytes count of the number of instances of QCT_DNS

Typically one record contains all the data. If there are a large number of dispatchers, adapters, or SSL tasks, the data is split over more than one record.

If this happens, the count of instances of the dispatchers can be zero, and information about a group of TCBs can be spread across multiple records. For example the number of instances can look like this:

Table 39. Example data

Count	First record	Last record
QWHS	1	1
QCCT	1	0
QDSP	50	5
QADP	0	10
QSSL	0	3
QDNS	0	1

This example shows that there were 55 dispatcher TCBs within the SMF interval.

The field `qwhs.qwhssmfc` indicates a continuation. If this bit is on, there are multiple SMF records containing information for this interval. If this bit is off, this is the last or only record.

Channel initiator statistics data records

Use this topic as a reference for channel initiator statistics data records.

The format of the channel initiator statistics data record contains two parts:

- The first part is the channel initiator control information block, described in assembler macro `thlqual.SCSQMACS(CSQDQCCT)`. For further information, see "Channel initiator control information block" on page 327.
- The second part is the channel initiator task block, described in assembler macro `thlqual.SCSQMACS(CSQDQCTA)`.

  The channel initiator task block contains information about the four types of task within the CHINIT. For further information, see:

  - "Dispatcher tasks" on page 328
  - "Adapter tasks" on page 329
  - "Domain Name Server (DNS) task" on page 330
  - "SSL tasks" on page 330

  Each task includes:

  - The elapsed time the task was active in the interval (*qcteltm*)
  - How much CPU time was using in the interval (*qctcptm*)
  - Total wait time of this task in the interval (*qctwttm*)
  - The number of requests in the interval (*qctreqn*)

  You can use this information to see how busy the task was, and determine whether you need to add more tasks based on the analysis.

  For SSL and DNS tasks, the duration of the longest request (*qctlgdu*, *qctlsdu*) and the time of day when this occurred (*qctlgdm*, *qctlsdm*) are also included.

  These can be useful to identify when channel requests took a long time. For example, a DNS lookup request going to a server outside of your enterprise taking seconds rather than milliseconds.

  The example accounting data in the following tasks has been formatted using IBM MQ SupportPac MP1B.

Both of the parts are also described in the C programming language header file `thlqual.SCSQC370(CSQDSMFC)`. Note that the field names in C are all in lowercase, for example, *qcct*, *qct_adp*.

Channel initiator control information block

Use this topic as a reference for the channel initiator control information block.

The channel initiator control information block contains basic information for this CHINIT, including:

- CHINIT job name (*qcctjobn*)
- QSG name if it is in a QSG (*qcctqsgn*)
- Peak number used of current channels (*qcctnocc*)
- Peak number used of active channels (*qcctnoac*)
- MAXCHL - maximum permitted current channels (*qcctmxcc*)
- ACTCHL - maximum permitted active channels (*qcctmxac*)
- TCPCHL - maximum permitted TCP/IP channels (*qcctmxtp*)
- LU62CHL - maximum permitted LU62 channels (*qcctmxlu*)

- Storage used by CHINIT in the extended private region (*qcctstus*). This information is also provided by the CSQX004I message in the CHINIT job log.

You can use this information to see if the number of active channels is approaching the configured maximum value. Note that the number of current and active channels are the values when the record was created. So, between the two intervals there might have been more than this number of channels active.

Channel information from SMF data

Here is an example of channel information from SMF data:

```
MVCA,MQPV,2014/03/18,13:00:00,VRM:800,
From 2014/03/18,12:45:00.015222 to 2014/03/18,13:00:00.083630 duration 900.068408 seconds
Peak number used of current channels..........  1
Peak number used of active channels ..........  1
MAXCHL. Max allowed current channels..........9999
ACTCHL. Max allowed active channels...........9999
TCPCHL. Max allowed TCP/IP channels...........9999
LU62CHL. Max allowed LU62 channels............ 200
Storage used by Chinit........................ 436MB
```

You can monitor the storage usage and see whether the value is trending upwards. If the total used is approaching the total storage available, you might be running out of storage, and so might not be able to support many more channels.

If the numbers of active current channels are tending towards the maximum number of channels, you might need to increase the maximum number of channels.

Dispatcher tasks
Example data for the dispatcher tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET"
    ,     ,         ,       ,   Seconds,      ,  uSeconds, uSeconds
   0, DISP,    26587,    0.4,  0.592463,   0.1,        22,      127
   1, DISP,    26963,    0.3,  0.588092,   0.1,        22,      112
   2, DISP,   864329,    2.7,  2.545668,   0.3,         3,       28
   3, DISP,    26875,    0.4,  0.590825,   0.1,        22,      120
   4, DISP,    26874,    0.4,  0.603285,   0.1,        22,      123
Summ, DISP,   971628,    0.8,  4.920332,   0.1,         5,       38
```

The example data shows that there were five dispatchers. A channel is associated with a dispatcher, and the work is distributed across all the dispatchers. This example shows that one dispatcher is processing more requests than other dispatchers. This is normal, as some channels might stop, so the dispatcher is processing fewer channels, and some channels can be busier than others.

- 4.9 seconds of CPU were used by the dispatchers.
- The average request used 5 microseconds of CPU and took 38 microseconds elapsed time.
- A dispatcher is used to send and receive data over a communications network, and this is not usually dependent on external events. The average elapsed time should, therefore, be close to the average CPU time used. If the CHINIT is delayed due to lack of CPU, then the ratio of average Elapsed Time to average CPU time is much larger, compared to when the CHINIT is not delayed for CPU.
- The average CPU used per request depends on the message traffic, for example, bigger messages use more CPU than smaller messages.

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm

- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn

Usually, the number of dispatchers should be less than, or equal to, the number of processors in the LPAR. If you have more dispatchers than processors in the LPAR they might compete for CPU resources. For more information about tuning your system, see SupportPac MP16.

Channels have an affinity to a dispatcher, so you might find that some dispatchers process many more requests than another dispatcher.

You can use the ALTER QMGR CHIDISPS() command to change the number of dispatchers used. Any change comes into effect the next time the CHINIT is started.

Adapter tasks
Example data for the adapter tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET"
    ,     ,         ,       ,   Seconds,      ,  uSeconds, uSeconds
   0, ADAP,   470297,   10.2, 41.290670,   4.6,        88,      194
   1, ADAP,    13907,    0.6,  1.589428,   0.2,       114,      365
   2, ADAP,     2517,    0.2,  0.185325,   0.0,        74,      746
   3, ADAP,     1095,    0.1,  0.085774,   0.0,        78,      907
   4, ADAP,      535,    0.1,  0.040743,   0.0,        76,      947
   5, ADAP,      220,    0.0,  0.016228,   0.0,        74,     1175
   6, ADAP,       82,    0.0,  0.005521,   0.0,        67,     1786
   7, ADAP,       80,    0.0,  0.004248,   0.0,        53,     1160
Summ, ADAP,   488733,    1.4, 43.217938,   0.6,        88,      205
```

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests: qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn average
- ET: qcteltm and qctreqn

This example shows that there were eight adapter tasks.

Adapter number 0

- Processed the majority of the requests (470297 out of 488733)
- Was busy 10.2% of the interval
- Used 41.3 seconds of CPU

Overall
The average CPU per request was 88 microseconds of CPU and took 205 microseconds

The adapters process IBM MQ requests. Some of these requests might wait, for example, for log I/O during a commit, so the average Elapsed Time per request has little meaning.

When an IBM MQ request is made the first free adapter task is used.

- If there is at least one adapter that has been little used (less than 1%) busy, you have enough adapters.
- If at least one adapter was not used, you have enough adapters defined.
- If all the adapters were used, you might need to allocate more adapters.

- If all of the adapters were used, and they were all busy for most of the interval, you need to allocate more adapters.

You can use the ALTER QMGR CHIADAPS() command to change the number of adapters used. Any changes come into effect the next time the CHINIT is started.

Attention: If there are too many adapters acting on a small set of queues, you might get contention within the queue manager.

Related information

ALTER QMGR

Domain Name Server (DNS) task
Example data for the DNS tasks, and information about how to interpret the data.

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET",  longest,
date,            time
    ,     ,         ,       ,   Seconds,      ,  uSeconds, uSeconds, uSeconds,           ,
   0,  DNS,    14002,    0.0,  0.122578,   0.0,         9,       11,      463, 2014/03/18,
12:56:33.987671
Summ,  DNS,    14002,    0.0,  0.122578,   0.0,         9,       11,      463, 2014/03/18,
12:56:33.987671
```

The CHINIT uses a single DNS task. The example shows that the task processed 14002 requests and on average the request used 9 microseconds of CPU and took 11 microseconds of elapsed time.

The longest DNS request took 463 microseconds elapsed time, and this occurred at 12:56:33 local time.

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn
- Longest: qctlgdu
- Longest at: qctlgtm

The DNS task can go out of your enterprise to look up the IP address associated with a name. If the average Elapsed time is significantly more than the average CPU time used, you might have some long requests.

If the value of the longest request time is unacceptable you should work with your network team to investigate why you are having long requests. It might be that you have an invalid name in your connections.

If the DNS task is busy for 25% of the duration, consider investigating the cause further.

Note: There are requests to the DNS task that are not DNS lookups, so you might have the number of requests being greater than zero - but no longest request information.

SSL tasks
Example data for the SSL tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET",  longest,
date,            time
```

```
    ,     ,          ,        ,    Seconds,        ,  uSeconds, uSeconds, uSeconds,              ,
   0,  SSL,     3112,    1.2,  0.248538,   0.3,        80,      362,     8864, 2014/03/18,
12:46:40.237697
   1,  SSL,     3070,    1.2,  0.245433,   0.3,        80,      359,     4714, 2014/03/18,
12:46:18.938022
   2,  SSL,     3170,    1.2,  0.255557,   0.3,        81,      362,     7273, 2014/03/18,
12:46:35.358145
   3,  SSL,     3060,    1.2,  0.246542,   0.3,        81,      365,    13164, 2014/03/18,
12:46:44.514045
   4,  SSL,     3120,    1.3,  0.251927,   0.3,        81,      373,    22438, 2014/03/18,
12:46:22.134123
Summ,  SSL,    15532,    1.2,  1.247998,   0.3,        80,      364,    22438, 2014/03/18,
12:46:22.134123
```

This example data shows that the average request took 364 microseconds. The longest request was for SSL task 4, took 22,438 microseconds, and occurred at 12:46:22.134123 local time.

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn
- Longest: qctlsdu longest at: qctlstm

A running channel is associated with an SSL task, in a similar way that a channel is associated with a dispatcher. The SSL tasks can use the cryptographic coprocessors available to the LPAR. So, the elapsed time can include time spent on a coprocessor. You should monitor the average elapsed time throughout the day. If this time increases significantly during peak periods you should work with your MVS systems programmers, as your coprocessors might be over utilized.

If the SSL tasks are busy for a significant proportion of the interval, increasing the number of SSL tasks might help. If the SSL tasks are waiting for external resources such as a coprocesor, increasing the number of SSL tasks has little effect.

You can use the ALTER QMGR SSLTASKS() command to change the number of SSL tasks used. Any changes come into effect the next time the CHINIT is started.

Related information
ALTER QMGR

Interpreting IBM MQ accounting data

IBM MQ accounting data is written as SMF type 116 records. Use this topic as a reference to the different types of accounting data records.

IBM MQ accounting information can be collected for the following subtypes:

0
: Message manager accounting records (how much processor time was spent processing IBM MQ API calls and the number of `MQPUT` and `MQGET` calls). This information is produced when a named task disconnects from IBM MQ, and so the information contained within the record might cover many hours.

1
: Accounting data for each task, at thread and queue level.

2
: Additional queue-level accounting data (if the task used more queues than could fit in the subtype 1 record).

10
Accounting data for channels.

Note: Accounting information for specific channels can be enabled or suppressed by the STATCHL channel attribute, and the STATACLS queue manager attribute.

Note that:

- Subtype 0 is produced with trace class(1)
- Subtypes 1 and 2 are produced with trace class(3)
- Subtype 10 is produced with accounting trace class(4)

Subtype

Layout of an SMF type 116 record

Use this topic as a reference to the format of an SMF type record.

The standard layout for SMF records involves three parts:

SMF header
Provides format, identification, and time and date information about the record itself.

Self-defining section
Defines the location and size of the individual data records within the SMF record.

Data records
The actual data from IBM MQ that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

The SMF header

Table 40 on page 332 shows the format of SMF record header (SM116).

Table 40. SMF record header description

Offset : Dec	Offset: Hex	Type	Len	Name	Description	Example
0	0	Structure	28	SM116	SMF record header.	
0	0	Integer	2	SM116LEN	SMF record length.	01A4
2	2		2		Reserved.	
4	4	Integer	1	SM116FLG	System indicator.	5E
5	5	Integer	1	SM116RTY	Record type. The SMF record type, for IBM MQ accounting records this is always 116 (X'74').	74
6	6	Integer	4	SM116TME	Time when SMF moved record.	00356124
10	A	Integer	4	SM116DTE	Date when SMF moved record.	0100223F
14	E	Character	4	SM116SID	z/OS subsystem ID. Defines the z/OS subsystem on which the records were collected.	D4E5F4F1 (MV41)
18	12	Character	4	SM116SSI	IBM MQ subsystem ID.	D4D8F0F7 (MQ07)
22	16	Integer	2	SM116STF	Record subtype.	0000

Table 40. SMF record header description (continued)						
Offset: Dec	**Offset: Hex**	**Type**	**Len**	**Name**	**Description**	**Example**
24	18	Character	3	SM116REL	IBM MQ version.	F6F0F0 (600)
27	1B		1		Reserved.	
28	1C	Character	0	SM116END	End of SMF header and start of self-defining section.	
Note: The (hexadecimal) values in the right-hand column relate to Figure 28 on page 334.						

Self-defining sections

A self-defining section of an SMF record tells you where to find an accounting record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow the header, at a fixed offset from the start of the SMF record.

Each self-defining section points to accounting related data. Table 41 on page 333 summarizes the offsets from the start of the SMF record header.

Table 41. Offsets to self-defining sections

Record subtype (SMF116STF)	Source of accounting data	Offset of self-defining section		See...
		Dec	Hex	
All	Common header	28	X'1C'	"Common IBM MQ SMF header" on page 335
0	Message manager	44	X'2C'	"Message manager data records" on page 337
1	Thread identification record	36	X'24'	"Thread-level and queue-level data records" on page 338
1	Thread-level accounting	44	X'2C'	"Thread-level and queue-level data records" on page 338
1	Queue-level accounting	52	X'34'	"Thread-level and queue-level data records" on page 338. This section is present only if the WTASWQCT field in the task-related information (WTAS) structure is non-zero.
2	Thread identification record	36	X'24'	"Thread-level and queue-level data records" on page 338
2	Queue-level accounting	44	X'2C'	"Thread-level and queue-level data records" on page 338

Table 41. Offsets to self-defining sections (continued)				
Record subtype (SMF116STF)	**Source of accounting data**	**Offset of self-defining section**		**See...**
		Dec	**Hex**	
10	Channel accounting			"Channel accounting data records" on page 340

Note: Other self-defining sections refer to data for IBM use only.

Each self-defining section is two fullwords long and has this format:

```
ssssssssllllnnnn
```

where:

ssssssss
Fullword containing the offset from start of the SMF record.

llll
Halfword giving the length of this data record.

nnnn
Halfword giving the number of data records in this SMF record.

Figure 28 on page 334 shows an example of part of an SMF type 116 record. The numbers in the left-hand column represent the offset, in hexadecimal, from the start of the record. Each line corresponds to sixteen bytes of data, where each byte is two hexadecimal characters, for example 0C. The characters in the right-hand column represent the printable characters for each byte. Non-printable characters are shown by a period (.) character.

In this example, alternate fields in the SMF header are underlined to help you to see them; refer to Table 40 on page 332 to identify them. The self defining section for one of the message manager accounting data records (at the offset given in Table 41 on page 333) is shown in **bold**.

```
000000 01A40000 5E740035 61240100 223FD4E5  *....;.../.....MV*
000000 F4F1D4D8 F0F70000 F6F0F000 00000134  *41MQ07..600.....*
000000 00700001 00000054 00B00001 00000104  *................*
000000 00300001 00000000 00000000 00000000  *................*
000000 00000000 00000000 00000000 00000000  *................*
```

Figure 28. Part of an SMF record 116 showing the header and self-defining sections

The self-defining section for the type of message manager accounting data is located at offset X'2C' from the start of the SMF record and contains this information:

- The offset of the message manager accounting data is located X'00000104' bytes from the start of the SMF record.
- This message manager record is X'0030' bytes long.
- There is one record (X'0001').

Note: Always use offsets in the self-defining sections to locate the accounting records.

z/OS Processing type 116 SMF records

Use this topic as a reference to the format of the processing type accounting record.

Any accounting data you collect from SMF must be processed to extract useful information. When you process the data, verify that the records are from IBM MQ and that they are the records you are expecting.

Validate the value of the following fields:

- SM116RTY, the SMF record number = X'74' (116)
- SM116STF, the record subtype, must be 0000, 0001, 0002, or 0010

Reading from the active SMF data sets (or SMF logstreams) is not supported. You must use the SMF program IFASMFDP (or IFASMFDL if logstreams are being used) to dump SMF records to a sequential data set so that they can be processed. For more information see "Using System Management Facility" on page 306.

There is a C sample program called CSQ4SMFD which prints the contents of SMF type 115 and 116 records from the sequential data set. The program is provided as source in thlqual.SCSQC37S and in executable format in thlqual.SCSQLOAD. Sample JCL is provided in thlqual.SCSQPROC(CSQ4SMFJ).

You need to update the SMFIN DD card with the name of the SMF data set. Use the z/OS command '/D SMF' to show the name of the data set, and you need to update the DUMPOUT DD card with the name for the output data set.

You also need to specify the START and END times that you require.

The following sample JCL extracts SMF records from SMF data sets and dumps them to the SMFOUT data set:

```
//SMFDUMP EXEC PGM=IFASMFDP,REGION=0M
//SYSPRINT DD SYSOUT=
//SMFIN  DD DSN=xxxxxx.MANA,DISP=SHR
//SMFOUT  DD DSN=xxxxxx.SMFOUT,SPACE=(CYL,(1,1)),DISP=(NEW,CATLG)
//SYSIN DD *
INDD(SMFIN,OPTIONS(DUMP))
OUTDD(SMFOUT,TYPE(116))
OUTDD(SMFOUT,TYPE(115))
START(1159) END(1210)
/*
```

The following sample JCL extracts SMF records from the SMF log stream named in LSNAME and dumps them to the SMFOUT data set:

```
//SMFDUMP EXEC PGM=IFASMFDL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SMFOUT  DD DSN=xxxxxx.SMFOUT,SPACE=(CYL,(1,1)),DISP=(NEW,CATLG)
//SYSIN DD *
LSNAME(IFASMF.MQ,OPTIONS(DUMP))
OUTDD(SMFOUT,TYPE(116))
OUTDD(SMFOUT,TYPE(115))
START(1159) END(1210)
/*
```

Common IBM MQ SMF header

Use this topic as a reference to the common IBM MQ SMF header type accounting record.

The format of this record is described in Table 42 on page 335 and in assembler macros thlqual.SCSQMACS(CSQDQWHS) and thlqual.SCSQMACS(CSQDQWHC), and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qwhs, qwhsnsda.

Details of the structures and fields can be found in IBM MQ supportpac MP1B.

The QWHS data includes the subsystem name. For subtype 1 records, it also shows whether there are queue-level accounting records present. If the QWHSNSDA field is 3 or less, there are not, and the corresponding self-defining section (at offset X'34') is not set.

The QWHC data gives you information about the user (for example, the user ID (QWHCAID) and the type of application (QWHCATYP)). The QWHC section is completed only for subtype 0 records. The equivalent information is present in the thread identification record for subtype 1 and 2 records.

Table 42. Structure of the common IBM MQ SMF header record QWHS

Offset: Dec	Offset : Hex	Type	Length	Name	Description
0	0	Structure	128	QWHS	

Table 42. Structure of the common IBM MQ SMF header record QWHS (continued)					
Offset: Dec	**Offset : Hex**	**Type**	**Length**	**Name**	**Description**
0	0		6		Reserved
6	6	Character	1	QWHSNSDA	Number of self defining sections in the SMF records
7	7		5		Reserved
12	C	Character	4	QWHSSSID	Subsystem name
16	10		24		Reserved
40	28	Character	8	QWHCAID	User ID associated with the z/OS job
48	30	Character	12	QWHCCV	Thread cross reference
60	3C	Character	8	QWHCCN	Connection name
68	44		8		Reserved
76	4C	Character	8	QWHCOPID	User ID associated with the transaction
84	54	Integer	4	QWHCATYP	Type of connecting system (1=CICS, 2=Batch or TSO, 3=IMS control region, 4=IMS MPP or BMP, 5=Command server, 6=Channel initiator, 7=RRS Batch)
88	58	Character	22	QWHCTOKN	Accounting token set to the z/OS accounting information for the user
110	6E	Character	16	QWHCNID	Network identifier
126	7E		2		Reserved

Combining CICS and IBM MQ performance data

Use this topic as a reference to the combination of IBM MQ and CICS performance data.

The common IBM MQ SMF header type accounting record section, QWHCTOKN, is used to correlate CICS type 110 SMF records with IBM MQ type 116 SMF records.

CICS generates an LU6.2 unit-of-work token, for each CICS task. The token is used to generate an accounting token that is written to QWHCTOKN in the correlation header of subtype zero records.

Details are also written to the WTIDACCT section in subtype 1 and 2 records. The accounting token enables correlation between CICS and IBM MQ performance data for a transaction.

Thread cross reference data

Use this topic as a reference to the format of the thread cross reference type accounting record.

The interpretation of the data in the thread cross reference (QWHCCV) field varies. This depends on what the data relates to:

- CICS connections (QWHCATYP=1) - see Table 43 on page 337
- IMS connections (QWHCATYP=3 or 4) - see Table 44 on page 337
- Batch connections (QWHCATYP=2 or 7) - this field consists of binary zeros
- Others - no meaningful data

Table 43. Structure of the thread cross reference for a CICS system				
Offset: Dec	**Offset: Hex**	**Type**	**Length**	**Description**
48	30	Character	4	CICS thread number.
52	34	Character	4	CICS transaction name.
56	38	Integer	4	CICS task number.

Some entries contain blank characters. These apply to the task, rather than to a specific transaction.

Table 44. Structure of the thread cross reference for an IMS system				
Offset: Dec	**Offset: Hex**	**Type**	**Length**	**Description**
48	30	Character	4	IMS partition specification table (PST) region identifier.
52	34	Character	8	IMS program specification block (PSB) name.

Message manager data records

Use this topic as a reference to the format of the message manager accounting records.

The message manager is the component of IBM MQ that processes all API requests. The format of the message manager accounting records is described in assembler macro thlqual.SCSQMACS(CSQDQMAC).

The QMAC data gives you information about the processor time spent processing IBM MQ calls, and counts of the number of `MQPUT` and `MQGET` requests for messages of different sizes.

Note: A single IMS application might write two SMF records. In this case, add the figures from both records to provide the correct totals for the IMS application.

Records containing zero processor time

Records are sometimes produced that contain zero processor time in the QMACCPUT field. These records occur when long running tasks identified to IBM MQ either terminate or are prompted to output accounting records by accounting trace being stopped. Such tasks exist in the CICS adapter and in the channel initiator (for distributed queuing). The number of these tasks with zero processor time depends upon how much activity there has been in the system:

- For the CICS adapter, this can result in up to nine records with zero processor time.
- For the channel initiator, the number of records with zero processor time can be up to the sum of `Adapters + Dispatchers + 6`, as defined in the queue manager attributes.

These records reflect the amount of work done under the task, and can be ignored.

Sample subtype zero accounting record

Use this topic as a reference to the format of the subtype zero accounting records.

Figure 29 on page 338 shows a type 116, subtype zero SMF record. In this figure, the SMF record header and the QMAC accounting data record are underlined. The self-defining sections are in bold.

```
000000  01A40000 5E740035 61240100 223FD4E5   *....;.../.....MV*
000010  F4F1D4D8 F0F70000 F6F0F000 00000134    *41MQ07..600.....*
000020  00700001 00000054 00B00001 00000104   *................*
000030  00300001 00000000 00000000 00000000   *................*
000040  00000000 00000000 00000000 00000000   *................*
000050  00000000 B478AB43 9C6C2280 B478AB47   *.........%......*
000060  9DB47E02 00000000 04C0F631 00000001   *..=......{6.....*
000070  9880E72D 00000000 014D9540 00000000   *..X......(. ....*
000080  08480C80 00000010 40404040 40404040   *........        *
000090  00000000 00000000 00000051 00000000   *................*
0000A0  00000000 00000000 00000000 00000000   *................*
0000B0  00000000 00000000 00000000 00000000   *................*
0000C0  00000000 00000000 00000000 00000000   *................*
0000D0  00000000 00000000 00000000 00000000   *................*
0000E0  00000000 00000000 00000000 00000000   *................*
0000F0  00000000 00000000 00000000 00000000   *................*
000100  00000000 D4140030 D8D4C1C3 00000000   *....M...QMAC....*
000110  689C738D 00000050 00000000 00000050   *.......&.......&*
000120  0000000A 00000000 00000000 00000000   *................*
000130  00000000 0024011A 00030710 02DAACF0   *...............0*
```

Figure 29. Example SMF type 116, subtype zero record

Thread-level and queue-level data records

Use this topic as a reference to the format of the thread-level and queue-level accounting records.

Thread level accounting records are collected for each task using IBM MQ. For each task, a thread-level accounting data record is written to the SMF when the task finishes. For a long running task, data is also written at the statistics interval set by the STATIME parameter of the CSQ6SYSP system parameter macro (or by the system SMF statistics broadcast), provided that the task was running the previous time statistics were gathered. In addition, accounting information is gathered about each queue that the task opens. A queue-level accounting record is written for each queue that the task has used since the thread-level accounting record was last written.

Thread-level and queue-level accounting records are produced if you specify class 3 when you start the accounting trace. For example, use the following command:

```
START TRACE(ACCTG) DEST(SMF) CLASS(3)
```

The thread level accounting information is written to an SMF type 116, subtype 1 record, and is followed by queue-level records. If the task opened many queues, further queue information is written to one or more SMF type 116 subtype 2 records. A thread identification control block is included in each subtype 1 and 2 record to enable you to relate each record to the correct task. Typically, the maximum number of queue-level records in each SMF record is about 45.

The format of the thread-level accounting record is described in assembler macro thlqual.SCSQMACS(CSQDWTAS). The format of the queue-level accounting record is described in assembler macro thlqual.SCSQMACS(CSQDWQ). The format of the thread identification record is described in assembler macro thlqual.SCSQMACS(CSQDWTID). All these records are also described in C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example wtas, wtasshex.

Meaning of the channel names

Use this topic as a reference to the meaning of channel names.

The channel name in the WTID is constructed as shown in the following example. In this example a sender channel exists from queue manager QM1 to queue manager QM2.

Table 45. Meaning of channel names

Field name	Meaning	Example
For queue manager QM1 the sender channel has the following fields set:		

Table 45. Meaning of channel names (continued)		
Field name	**Meaning**	**Example**
WTIDCCN	The job name	QM1CHIN
WTIDCHL	The channel name	QM1.QM2
WTIDCHLC	This is defined in the CONNAME of the channel	WINMVS2B(2162)
For queue manager QM2 the receiver channel has the following fields set:		
WTIDCCN	The job name	QM2CHIN
WTIDCHL	The channel name	QM1.QM2
WTIDCHLC	Where the channel came from	9.20.101.14

Sample subtype 1 and subtype 2 records
Use this topic as a reference to the format of the subtype 1 and subtype 2 accounting records.

Figure 30 on page 339 and Figure 31 on page 339 show examples of SMF type 116, subtype 1 and subtype 2 records. These two accounting records were created for a batch job that opened 80 queues. Because many queues were opened, a subtype 2 record was required to contain all the information produced.

```
000000  703C0000 5E74002D 983B0100 229FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70001 F6F0F000 00006FCC  *41MQ07..600...?.*
000020  00700001 0000003C 00D00001 0000010C  *.........}......*
000030  02C00001 000003CC 02400030 F70000D0  *.{....... ..7..}*
000040  E6E3C9C4 00000000 00000000 00000040  *WTID........... *
.
.
.
000100  00000000 00000000 7F4A4BB8 F70102C0  *........"...7..{*
000110  E6E3C1E2 B4802373 0BF07885 7F4AE718  *WTAS.....0..".X.*
```

Figure 30. Example SMF type 116, subtype 1 record

The first self-defining section starts at X'24' and is **bold** in the example; X'0000003C' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'0000010C' is the offset to the WTAS data record, X'02C0' is the length of the WTAS record, and X'0001' is the number of WTAS records.

The third self-defining section starts at X'34' and is **bold** in the example; X'000003CC' is the offset to the first WQST data record, X'0240' is the length of the WQST record, and X'0030' is the number of WQST records.

Figure 31 on page 339 shows an example of an SMF type 116, subtype 2 record.

```
000000  49740000 5E74002D 983B0100 229FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70002 F6F0F000 00004904  *41MQ07..600.....*
000020  00700001 00000034 00D00001 00000104  *.........}......*
000030  02400020 F70000D0 E6E3C9C4 00000002  *. ..7..}WTID....*
.
.
.
000100  7F4A4BB8 F7020240 E6D8E2E3 00000001  *"...7.. WQST....*
```

Figure 31. Example SMF type 116, subtype 2 record

The first self-defining section starts at X'24' and is **bold** in the example; X'00000034' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'00000104' is the offset to the first WQST data record, X'0240' is the length of the WQST record, and X'0020' is the number of WQST records.

Figure 32 on page 340 shows an example of an SMF type 116, subtype 1 record where no queues have been opened and there are consequently no self-defining sections for WQST records..

```
000000          5E740039 4E9B0104 344FD4E5  *    .........|MV*
000010 F4F1D4D8 F0F70001 F6F0F000 000003DC  *41MQ07..600.....*
000020 00800001 00000034 00D00001 00000104  *................*
000030 02D80001 F70000D0 E6E3C9C4 00000002  *.Q..7...WTID....*
000040 C1F8C5C1 C4C5D740 C1F8C5C1 C4C54040  *A8EADEP A8EADE  *
000050 40404040 40404040 00000000 00000000  *        ........*
000060 40404040 40404040 4040               *          *
```

Figure 32. Example SMF type 116, subtype 1 record with no WQST data records

The first self-defining section starts at X'24' and is **bold** in the example; X'00000034' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'0000010C' is the offset to the WTAS data record, X'02D8' is the length of the WTAS record, and X'0001' is the number of WTAS records.

There is no self-defining section describing a WQST data record, equivalent to the third self-defining section in Figure 30 on page 339.

Channel accounting data records

Use this topic as a reference for channel accounting data records.

The format of the channel accounting data record is described in assembler macro `thlqual.SCSQMACS(CSQDQCST)`. The format is also described in the C programming language header file `thlqual.SCSQC370(CSQDSMFC)`. Note that the field names in C are all in lowercase, for example, *qcst*.

The channel accounting data gives you information about the status and statistics of each channel instance, including:

- Average network time (*qcstntav*)
- Average time on exit (*qcstetav*)
- Channel batch data limit (*qcstcbdl*)
- Channel batch interval (*qcstcbit*)
- Channel batch size (*qcstcbsz*)
- Channel dispatcher number (*qcstdspn*)
- Channel disposition (*qcstchdp*)
- Channel name (*qcstchnm*)
- Channel state (*qcstchst*)
- Channel started time (*qcststrt*)
- Channel status collected time (*qcstcltm*)
- Channel stopped time (*qcstludt*)
- Channel type (*qcstchty*)
- Common name (CN) from SSLCERTI (*qcstslcn*)
- Compression rate (*qcstcpra*)
- Connection name (*qcstcnnm*)

- Current shared conversations (*qcstcscv*)
- DNS resolution time (*qcstdnrt*)
- Effective value of STATCHL parameter (*qcststcl*)
- Last message time (*qcstlmst*)
- Maximum network time (qcstntmx)
- Maximum time on exit (*qcstetmx*)
- Minimum network time (qcstntmn)
- Minimum time on exit (*qcstetmn*)
- Name of the remote queue manager or application (*qcstrqmn*)
- Number of batches (*qcstbatc*)
- Number of bytes for message data (*qcstnbyt*)
- Number of bytes for persistent message data (*qcstnpby*)
- Number of bytes received for both message data and control information (*qcstbyrc*)
- Number of bytes sent for both message data and control information (*qcstbyst*)
- Number of full batches (*qcstfuba*)
- Number of messages, or number of MQI calls (*qcstnmsg*)
- Number of persistent messages (*qcstnpmg*)
- Number of put retries (*qcstptrc*)
- Number of transmission queue becoming empty (*qcstqetc*)
- Number of transmission buffers received (**qcstbfrc**)
- Number of transmission buffers sent (*qcstbfst*)
- Serial number from SSLPEER (*qcstslsn*)
- SSL CipherSpec (zero means SSL not used) (*qcstslcs*)
- The date and time of maximum network time (*qcstntdt*)
- The date and time of maximum time on exit (*qcstetdt*)

Note, that for the channel accounting field *qcstetmn* (Minimum time on exit) and *qcstntmn* (Minimum network time) these two fields will be initialized to the hexadecimal value of 8FFFFFFF when unused.

You can use this information to see the throughput of a channel, if the actual batches are approaching the limit, the latency of the network, information about the remote end, performance of user exit, and so on.

Here is an example of the channel accounting data which has been formatted with IBM MQ SupportPac MP1B.

The fields available are based on the display channel status command (DIS CHS) and channel statistics by IBM MQ on platforms except z/OS, with some additional fields.

```
The data and time of the start and end of the record in local time, and the duration
SMF interval start        2014/03/26,02:30:00
SMF interval end          2014/03/26,02:45:00
SMF interval duration     899.997759 seconds

Information about the channel

Connection name        9.20.4.159
Channel disp           PRIVATE
Channel type           RECEIVER
Channel status         CLOSING
Channel STATCHL        HIGH


Start date & time            2014/03/26,02:44:58
Channel status collect time  2014/03/26,02:45:00
Last status changed          1900/01/01,00:00:00
Last msg time                2014/03/26,02:44:59
```

```
Batch size                        50
Messages/batch                    3.3
Number of messages                1,102
Number of persistent messages     1,102
Number of batches                 335
Number of full batches            0
Number of partial batches         335
Buffers sent                      337
Buffers received                  1,272
Message data                      5,038,344   4 MB
Persistent message data           5,038,344   4 MB
Non persistent message data       0   0 B
Total bytes sent                  9,852   9 KB
Total bytes received              5,043,520   4 MB
Bytes received/Batch              15,055  14 KB
Bytes sent/Batch                  29  29 B
Batches/Second                    1
Bytes received/message            4,576   4 KB
Bytes sent/message                8   8 B
Bytes received/second             28,019  27 KB/sec
Bytes sent/second                 54  54 B/sec
Compression rate                  0

The name of the queue manager at the remote end of the connection
Remote qmgr/app              MQPH
Put retry count              0
```

Tuning your IBM MQ network

Use the tuning tips in this section to help improve the performance of your queue manager network.

Tuning client and server connection channels

The default settings for client and server connection channels changed in Version 7.0 to use shared conversations. Performance enhancements for distributed severs were then introduced in Version 8.0. To benefit from the new features that were introduced alongside shared conversations, without the performance impact on the distributed server, set **SHARECNV** to 1 on your Version 8.0 server connection channels.

From Version 7.0, each channel is defined by default to run up to 10 client conversations per channel instance. Before Version 7.0, each conversation was allocated to a different channel instance. The enhancements added in Version 7.0 also include the following features:

- Bi-directional heartbeats
- Administrator stop-quiesce
- Read-ahead
- Asynchronous-consume by client applications

For some configurations, using shared conversations brings significant benefits. However, for distributed servers, processing messages on channels that use the default configuration of 10 shared conversations is on average 15% slower than on channels that do not use shared conversations. On an MQI channel instance that is sharing conversations, all of the conversations on a socket are received by the same thread. If the conversations sharing a socket are all busy, the conversational threads contend with one another to use the receiving thread. The contention causes delays, and in this situation using a smaller number of shared conversations is better.

You use the **SHARECNV** parameter to specify the maximum number of conversations to be shared over a particular TCP/IP client channel instance. For details of all possible values, and of the new features added in Version 7.0, see MQI client: Default behavior of client-connection and server-connection. If you do not need shared conversations, there are two settings that give best performance in Version 8.0:

- `SHARECNV(1)`. Use this setting whenever possible. It eliminates contention to use the receiving thread, and your client applications can take advantage of the new features added in Version 7.0. For this setting, distributed server performance is significantly improved in Version 8.0. The performance improvements apply to Version 8.0 client applications that issue non read ahead synchronous get wait calls; for example C client MQGET wait calls. When these client applications are connected, the distributed server uses less threads and less memory and the throughput is increased.

- `SHARECNV(0)`. The channel instance behaves exactly as if it was a Version 6.0 server or client connection channel, and you do not get the extra features such as bi-directional heartbeats that are available when you set **SHARECNV** to `1` or greater. Use a value of `0` only if you have existing client applications that do not run correctly when you set **SHARECNV** to `1` or greater.

Note: If a server has clients connected to it that are sharing conversations over a socket, and you decrease the shared conversations setting from `SHARECNV(10)` to `SHARECNV(1)`, this has the following effects:

- Increased socket usage on the server.
- Increased channel instances on the server.

In this case, you might also choose to increase the settings for **`MaxChannels`** and **`MaxActiveChannels`**.

For consistency with previous releases the default SVRCONN channel has not been updated, so you need explicitly to set **SHARECNV** to `1` or `0`.

Related information

MQI client: Default behavior of client-connection and server-connection

Tuning distributed publish/subscribe networks

Use the tuning tips in this section to help improve the performance of your IBM MQ distributed publish/subscribe clusters and hierarchies.

Related concepts

"Monitoring clusters" on page 299

Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Direct routed publish/subscribe cluster performance

In direct routed publish/subscribe clusters, information such as clustered topics and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. To reduce the effect of cluster management on performance you can perform updates at off-peak times, define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster, or switch to using topic host routing.

There are two sources of workload on a queue manager in a publish/subscribe cluster:

- Directly handling messages for application programs.
- Handling messages and channels needed to manage the cluster.

In a typical point-to-point cluster, the cluster system workload is largely limited to information explicitly requested by members of the cluster as required. Therefore in anything other than a very large point-to-point cluster, for example one which contains thousands of queue managers, you can largely discount the performance effect of managing the cluster. However, in a direct routed publish/subscribe cluster, information such as clustered topics, queue manager membership and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This can create a significant additional load on the system. Therefore you need to consider the effect of cluster management on queue manager performance, both in its timing, and its size.

Performance characteristics of direct routed clusters

Compare a point-to-point cluster with a direct routed publish/subscribe cluster in respect of the core management tasks.

First, a point to point cluster:

1. When a new cluster queue is defined, the destination information is pushed to the full repository queue managers, and only sent to other cluster members when they first reference a cluster queue (for example, when an application attempts to open it). This information is then cached locally by the queue manager to remove the need to remotely retrieve the information each time the queue is accessed.
2. Adding a queue manager to a cluster does not directly affect the load on other queue managers. Information about the new queue manager is pushed to the full repositories, but channels to the new queue manager from other queue managers in the cluster are only created and started when traffic begins to flow to or from the new queue manager.

In summary, the load on a queue manager in a point-to-point cluster is related to the message traffic it handles for application programs and is not directly related to the size of the cluster.

Second, a direct routed publish/subscribe cluster:

1. When a new cluster topic is defined, the information is pushed to the full repository queue managers, and from there directly to all members of the cluster, causing channels to be started to each member of the cluster from the full repositories if not already started. If this is the first direct clustered topic, each queue manager member is sent information about all other queue manager members in the cluster.
2. When a subscription is created to a cluster topic on a new topic string, the information is pushed directly from that queue manager to all other members of the cluster immediately, causing channels to be started to each member of the cluster from that queue manager if not already started.
3. When a new queue manager joins an existing cluster, information about all clustered topics (and all queue manager members if a direct cluster topic is defined) is pushed to the new queue manager from the full repository queue managers. The new queue manager then synchronizes knowledge of all subscriptions to cluster topics in the cluster with all members of the cluster.

In summary, cluster management load at any queue manager in a direct routed publish/subscribe cluster grows with the number of queue managers, clustered topics, and changes to subscriptions on different topic strings within the cluster, irrespective of the local use of those cluster topics on each queue manager.

In a large cluster, or one where the rate of change of subscriptions is high, this level of cluster management can be a significant overhead across all queue managers.

Reducing the effect of direct routed publish/subscribe on performance

To reduce the effect of cluster management on the performance of a direct routed publish/subscribe cluster, consider the following options:

- Perform cluster, topic, and subscription updates at off-peak times of the day.
- Define a much smaller subset of queue managers involved in publish/subscribe, and make that an "overlapping" cluster. This cluster is then the cluster where cluster topics are defined. Although some queue managers are now in two clusters, the overall effect of publish/subscribe is reduced:
  - The size of the publish/subscribe cluster is smaller.
  - Queue managers not in the publish/subscribe cluster are much less affected by cluster management traffic.

If the previous options do not adequately resolve your performance issues, consider using a *topic host routed* publish/subscribe cluster instead. For a detailed comparison of direct routing and topic host routing in publish/subscribe clusters, see Designing publish/subscribe clusters.

Related concepts

Topic host routed publish/subscribe cluster performance
A topic host routed publish/subscribe cluster gives you precise control over which queue managers host each topic. These topic hosts become the *routing* queue managers for that branch of the topic tree. Moreover, queue managers without subscriptions or publishers have no need to connect with the topic

hosts. This configuration can significantly reduce the number of connections between queue managers in the cluster, and the amount of information that is passed between queue managers.

Balancing producers and consumers in publish/subscribe networks
An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Subscription performance in publish/subscribe networks
Distributed publish/subscribe in IBM MQ works by propagating knowledge of where subscriptions to different topic strings have been created in the queue manager network. This enables the queue manager on which a message is published to identify which other queue managers require a copy of the published message, to match their subscriptions.

Topic host routed publish/subscribe cluster performance

A topic host routed publish/subscribe cluster gives you precise control over which queue managers host each topic. These topic hosts become the *routing* queue managers for that branch of the topic tree. Moreover, queue managers without subscriptions or publishers have no need to connect with the topic hosts. This configuration can significantly reduce the number of connections between queue managers in the cluster, and the amount of information that is passed between queue managers.

A topic host routed publish/subscribe cluster behaves as follows:

- Topics are manually defined on individual *topic host* queue managers in the cluster.
- When a subscription is made on a cluster queue manager, proxy subscriptions are created only on the topic hosts.
- When an application publishes information to a topic, the receiving queue manager forwards the publication to a queue manager that hosts the topic. The topic host then sends the publication to all queue managers in the cluster that have valid subscriptions to the topic.

For a more detailed introduction to topic host routing, see Topic host routing in clusters.

For many configurations, topic host routing is a more appropriate topology than *direct routing* because it provides the following benefits:

- Improved scalability of larger clusters. Only the topic host queue managers need to be able to connect to all other queue managers in the cluster. Therefore, there are fewer channels between queue managers, and there is less inter-queue manager publish/subscribe administrative traffic than for direct routing. When subscriptions change on a queue manager, only the topic host queue managers need to be informed.
- More control over the physical configuration. With direct routing, all queue managers assume all roles, and therefore all need to be equally capable. With topic host routing, you explicitly choose the topic host queue managers. Therefore, you can ensure that those queue managers are running on adequate equipment, and you can use less powerful systems for the other queue managers.

However, topic host routing also imposes certain constraints upon your system:

- System configuration and maintenance require more planning than for direct routing. You need to decide which points to cluster in the topic tree, and the location of the topic definitions in the cluster.
- Just as for direct routed topics, when a new topic host routed topic is defined, the information is pushed to the full repository queue managers, and from there direct to all members of the cluster. This event causes channels to be started to each member of the cluster from the full repositories if not already started.
- Publications are always sent to a host queue manager from a non-host queue manager, even if there are no subscriptions in the cluster. Therefore, you should use routed topics when subscriptions are typically expected to exist, or when the overhead of global connectivity and knowledge is greater than the risk of extra publication traffic.

- Messages that are published on non-host queue managers do not go direct to the queue manager that hosts the subscription, they are always routed through a topic host queue manager. This approach can increase the total overhead to the cluster, and increase message latency and reduce performance.

  Note: For certain configurations, you can usefully remove this constraint as described in Topic host routing using centralized publishers or subscribers.
- Using a single topic host queue manager introduces a single point of failure for all messages that are published to a topic. You can remove this single point of failure by defining multiple topic hosts. However, having multiple hosts affects the order of published messages as received by subscriptions.
- Extra message load is incurred by topic host queue managers, because publication traffic from multiple queue managers needs to be processed by them. This load can be lessened: Either use multiple topic hosts for a single topic (in which case message ordering is not maintained), or use different queue managers to host routed topics for different branches of the topic tree.

Topic host routing with centralized publishers or subscribers

To remove the extra "hop" incurred when publications are always routed to subscriptions through a topic host queue manager, configure the publishers or the subscriptions on the same queue manager that hosts the topic. This approach brings maximum performance benefits in the following two cases:

- Topics with many publishers and few subscriptions. In this case, host the subscriptions on the topic host queue manager.
- Topics with few publishers and many subscriptions. In this case, host the publishers on the topic host queue manager.

The following figure shows a topic host queue manager that also hosts the subscriptions. This approach removes the extra "hop" between the publisher and the subscriber, and reduces unnecessary sharing of subscription knowledge across all members of the cluster:

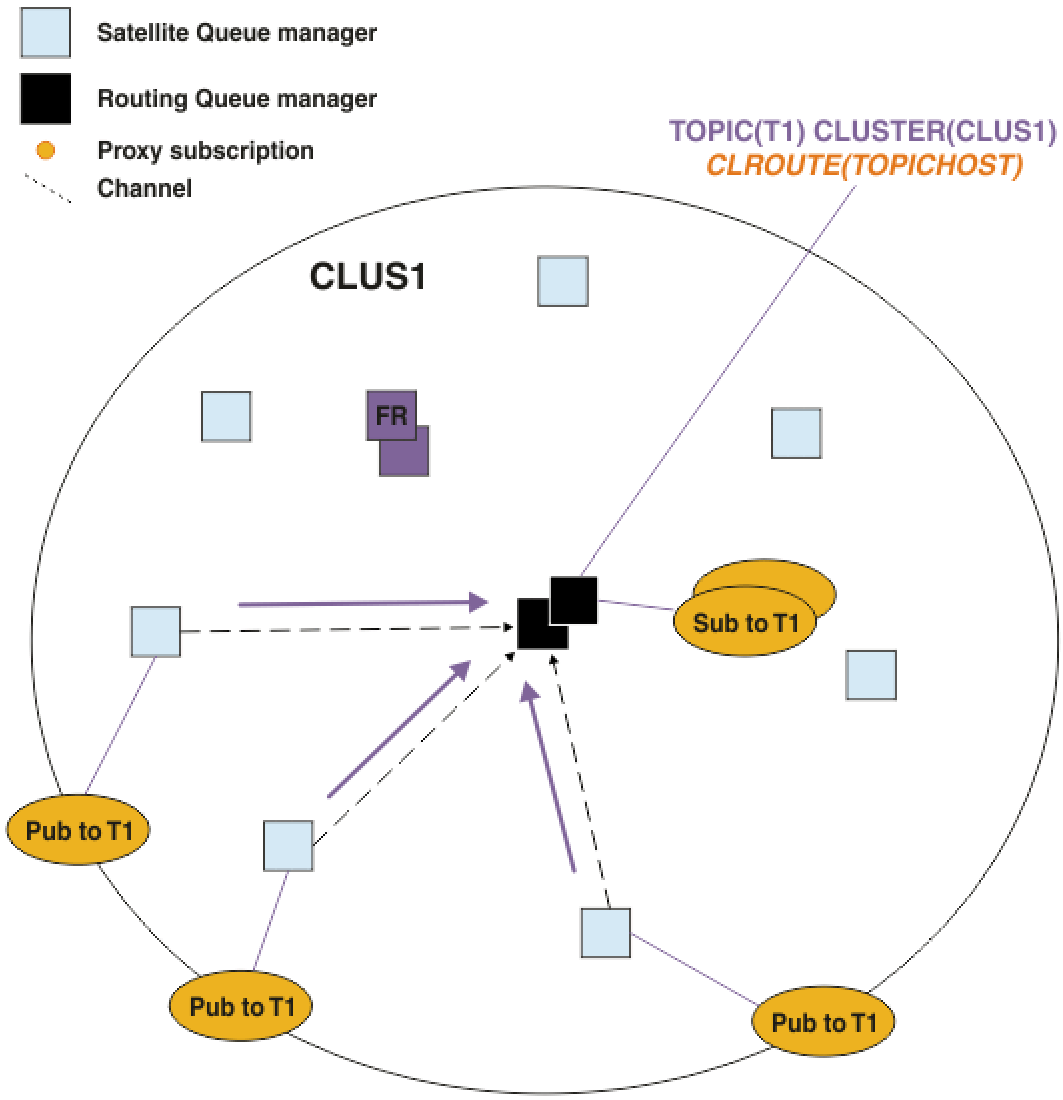


Figure 33. Hosting subscriptions on a topic host queue manager

The following figure shows a topic host queue manager that also hosts the publishers. This approach removes the extra "hop" between the publisher and the subscriber, and reduces unnecessary sharing of subscription knowledge across all members of the cluster:

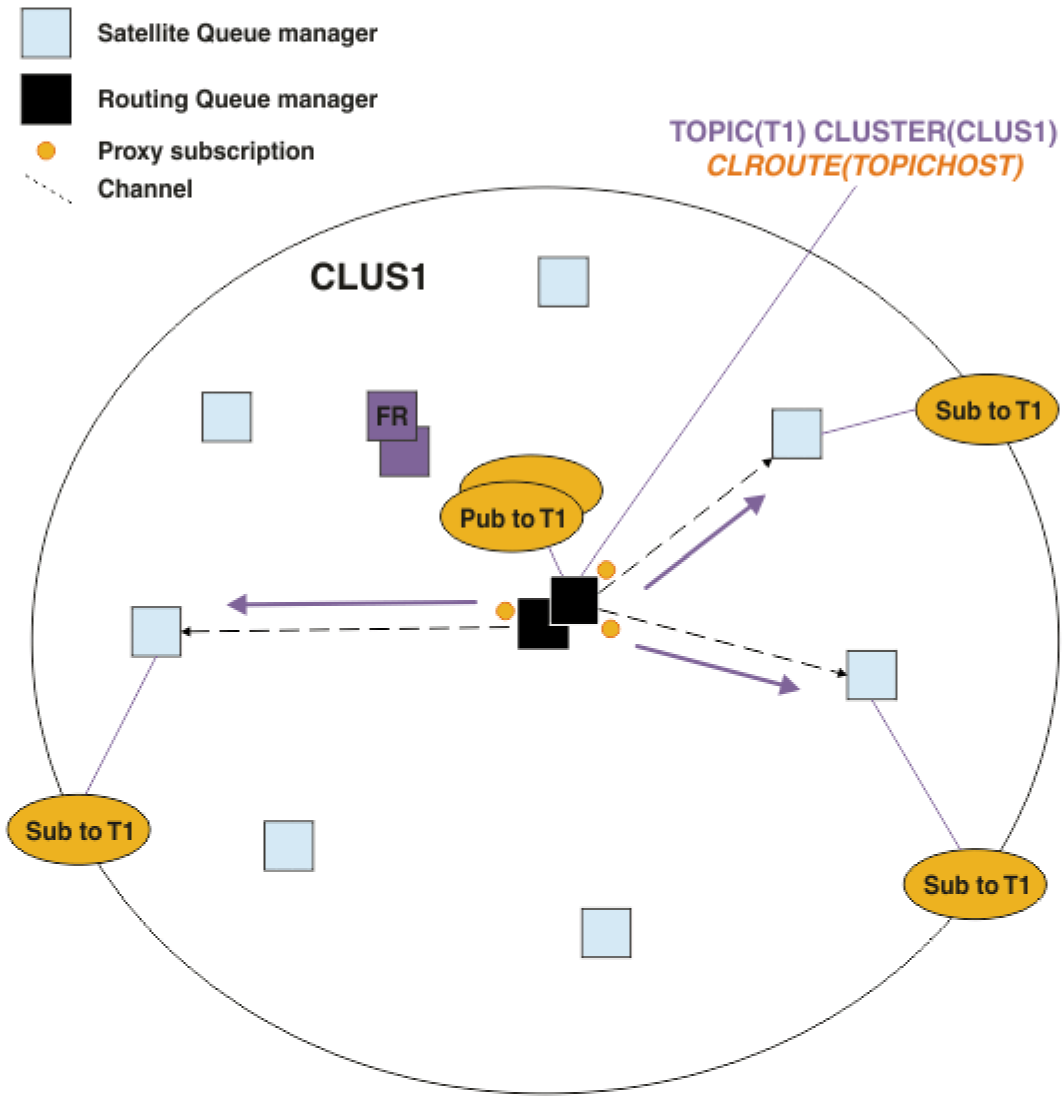


Figure 34. Hosting publications on a topic host queue manager

Related concepts

Direct routed publish/subscribe cluster performance
In direct routed publish/subscribe clusters, information such as clustered topics and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. To reduce the effect of cluster management on performance you can perform updates at off-peak times, define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster, or switch to using topic host routing.

Balancing producers and consumers in publish/subscribe networks
An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Subscription performance in publish/subscribe networks
Distributed publish/subscribe in IBM MQ works by propagating knowledge of where subscriptions to different topic strings have been created in the queue manager network. This enables the queue manager

on which a message is published to identify which other queue managers require a copy of the published message, to match their subscriptions.

Balancing producers and consumers in publish/subscribe networks

An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

In a point-to-point messaging topology, the relationship between message consumers and message producers is readily understood. You can get estimates of message production and consumption, queue by queue, channel by channel. If there is a lack of balance, the bottlenecks are readily identified and then remedied.

It is harder to work out whether publishers and subscribers are balanced in a publish/subscribe topology. Start from each subscription, and work back to the queue managers having publishers on the topic. Calculate the number of publications flowing to each subscriber from each queue manager.

Each publication that matches a subscription on a remote queue manager (based on proxy subscriptions) is put to a transmission queue. If multiple remote queue managers have proxy subscriptions for that publication, multiple copies of the message are put to a transmission queue, each targeted for a different sender channel.

In a publish/subscribe cluster, those publications are targeted at the `SYSTEM.INTER.QMGR.PUBS` queue on the remote queue managers that host the subscriptions. In a hierarchy, each publication is targeted at the `SYSTEM.BROKER.DEFAULT.STREAM` queue, or any other stream queues listed in the `SYSTEM.QPUBSUB.QUEUE.NAMELIST` on the remote queue managers. Each queue manager processes messages arriving on that queue and delivers them to the correct subscriptions on that queue manager.

For this reason, monitor the load at the following points where bottlenecks might arise:

- Monitor the load at the individual subscription queues.
  - This bottleneck implies that the subscribing application is not consuming the publications as quick as they are being published.
- Monitor the load at the `SYSTEM.INTER.QMGR.PUBS` queue or the stream queues.
  - This bottleneck implies that the queue manager is receiving publications from one or more remote queue managers faster than it can distribute them to the local subscriptions.
  - When seen on a topic host queue manager when using topic host routing in a cluster, consider making additional queue managers topic hosts, allowing the publication workload to be balanced across them. However, this will affect the message ordering across publications. See Topic host routing using multiple topic hosts for a single topic.
- Monitor the load at the channels between the publishing queue manager and the subscribing queue managers, which are fed by the transmission queues on the publishing queue manager.
  - This bottleneck implies that either one or more channels is not running, or messages are being published to the local queue manager faster than the channels can deliver them to the remote queue manager.
  - When you use a publish/subscribe cluster, consider defining additional cluster receiver channels on the target queue manager. This allows the publication workload to be balanced across them. However, this affects the message ordering across publications. Also consider moving to a multiple cluster transmission queue configuration, because this can improve performance in certain circumstances.
- If the publishing application is using a queued publish/subscribe interface, monitor the load at (a) the `SYSTEM.BROKER.DEFAULT.STREAM` queue, and any other stream queues listed in the `SYSTEM.QPUBSUB.QUEUE.NAMELIST` ; and (b) the `SYSTEM.BROKER.DEFAULT.SUBPOINT` queue, and any other subpoint queues listed in the `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST` .
  - This bottleneck implies that messages are being put by local publishing applications faster than the local queue manager can process the messages.

Related concepts

Direct routed publish/subscribe cluster performance
In direct routed publish/subscribe clusters, information such as clustered topics and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. To reduce the effect of cluster management on performance you can perform updates at off-peak times, define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster, or switch to using topic host routing.

Topic host routed publish/subscribe cluster performance
A topic host routed publish/subscribe cluster gives you precise control over which queue managers host each topic. These topic hosts become the *routing* queue managers for that branch of the topic tree. Moreover, queue managers without subscriptions or publishers have no need to connect with the topic hosts. This configuration can significantly reduce the number of connections between queue managers in the cluster, and the amount of information that is passed between queue managers.

Subscription performance in publish/subscribe networks
Distributed publish/subscribe in IBM MQ works by propagating knowledge of where subscriptions to different topic strings have been created in the queue manager network. This enables the queue manager on which a message is published to identify which other queue managers require a copy of the published message, to match their subscriptions.

“Monitoring clusters” on page 299
Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Subscription performance in publish/subscribe networks

Distributed publish/subscribe in IBM MQ works by propagating knowledge of where subscriptions to different topic strings have been created in the queue manager network. This enables the queue manager on which a message is published to identify which other queue managers require a copy of the published message, to match their subscriptions.

This approach minimizes the sending of published messages to queue managers on which no matching subscriptions exist. However, the propagation of the subscription knowledge can become a significant overhead, when the number of topic strings being subscribed to is high and constantly changing through frequent subscription creation and deletion.

You can affect performance by adjusting how publications and subscriptions are flowed around your publish/subscribe network. If your network traffic has few publications, and rapid subscription creation, deletion, or change, you can stop subscription information being flowed to all queue managers, and instead forward all publications to all queue managers in the network. You can also restrict the flow of proxy subscriptions and publications for a given topic between connected queue managers, restrict the flow of proxy subscriptions containing wildcards, and reduce the number and transient nature of topic strings.

Individual subscription propagation and *publish everywhere*

Publish everywhere is an alternative to individual subscription propagation. With individual propagation, only publications that have a matching subscription on a queue manager are forwarded to that queue manager. With *publish everywhere*, all publications are forwarded to all queue managers in the network. The receiving queue managers then deliver those publications that match local subscriptions.

Individual subscription propagation

This mechanism results in the least amount of inter-queue manager publication traffic, because only those publications that match subscriptions on a queue manager are sent.

However:

- For each individual topic string that is subscribed to, a proxy subscription is sent to other queue managers in the publish/subscribe topology. The set of queue managers depends on the routing model being used, as described in Planning your distributed publish/subscribe network.

- This messaging overhead can be significant if there are many thousands of subscriptions to create or delete (for example, recreating all non-durable subscriptions after a restart of a queue manager), or if the set of subscriptions is changing rapidly, and each is to a different topic string.
- The number of queue managers to which the proxy subscription is propagated also affects the scale of the overhead.

- Proxy subscriptions are flowed to other queue managers using asynchronous messaging. This has the following effect:
  - There is a delay between the creation of a subscription, and the creation, delivery, and processing of the proxy subscription by the other queue managers.
  - Messages that are published at those queue managers in that interval are not delivered to the remote subscription.

Publish everywhere

With this mechanism there is no per topic string proxy subscription overhead on the system. This means that rapid subscription creation, deletion, or change does not result in increased network load and processing.

There is also no delay between creating a subscription and publications being flowed to a queue manager, because all publications are flowed to all queue managers. Therefore there is no window in which publications are not delivered to newly-created remote subscriptions.

However:

- Sending all publications to all queue managers in the publish/subscribe topology, can result in excessive network traffic where publications do not have matching subscriptions on each queue manager.
  - The greater the number of queue managers in the topology, the greater the overhead.

You should consider using the *publish everywhere* mechanism when you expect a publication to be subscribed to from a significant proportion of your queue managers, or where the proxy subscription overheads are too great because of the frequency of subscription changes. You should use individual proxy subscription forwarding in cases where you experience increased messaging traffic when publications are sent to all queue managers, rather than to the queue managers with matching subscriptions.

You can set *publish everywhere* behavior at any level within the topic tree. To enable *publish everywhere*, you set the **PROXYSUB** parameter to `FORCE` for a high-level topic object. This results in a single wildcard proxy subscription that matches all topics below this topic object in the topic tree. When set it on a clustered topic object, the **PROXYSUB(FORCE)** attribute is propagated to every queue manager in the network, not just the queue manager that the topic was defined on.

Note: When used in a hierarchy, you set **PROXYSUB(FORCE)** individually on each queue manager, so the topology mechanism naturally limits the number of channels. However, when used in a cluster, many additional channels might be started:

- In a topic host routed cluster, channels are started from each queue manager to each topic host queue manager.
- In a direct routed cluster, channels are started from every queue manager to every other queue manager.

The overhead of starting many channels is most pronounced in a direct routed cluster, and can cause performance issues. See "Direct routed publish/subscribe cluster performance" on page 343.

Other ways of restricting the flow of proxy subscriptions and publications between connected queue managers

Consolidate topic strings

The use of many distinct, transient, topic strings introduces some level of management overhead on each queue manager in the system where publishers or subscriptions are attached. You should periodically assess the use of topic strings to see whether they can be consolidated. Reducing the number and transient nature of topic strings, and therefore the publishers and subscriptions to them, reduces the impact on the system.

Restrict publication and subscription scope
For a given topic, you can use the Publication scope and Subscription scope settings to keep publications and subscriptions local to the queue manager on which they are defined.

Block subscriptions made to wildcarded topics
You can restrict the flow of proxy subscriptions containing wildcards by setting the **Topic** attribute WILDCARD to BLOCK. See Wildcards in proxy subscriptions.

See also "Balancing producers and consumers in publish/subscribe networks" on page 349

Monitoring proxy subscription traffic in clusters

When considering the load on the system from the proxy subscription traffic, in addition to monitoring the queues listed in"Balancing producers and consumers in publish/subscribe networks" on page 349, monitor the following cluster queues:

- The SYSTEM.INTER.QMGR.FANREQ queue on the subscriber queue manager.
- The SYSTEM.INTER.QMGR.CONTROL queue on all other queue managers in the cluster.

Any significant message backlog on these queues implies that either the rate of subscription change is too great for the system, or a queue manager is not correctly functioning in the cluster. If you suspect the problem lies with a specific queue manager, check that publish/subscribe support is not disabled for that queue manager. See **PSMODE** in ALTER QMGR.

Related concepts

Direct routed publish/subscribe cluster performance
In direct routed publish/subscribe clusters, information such as clustered topics and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. To reduce the effect of cluster management on performance you can perform updates at off-peak times, define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster, or switch to using topic host routing.

Topic host routed publish/subscribe cluster performance
A topic host routed publish/subscribe cluster gives you precise control over which queue managers host each topic. These topic hosts become the *routing* queue managers for that branch of the topic tree. Moreover, queue managers without subscriptions or publishers have no need to connect with the topic hosts. This configuration can significantly reduce the number of connections between queue managers in the cluster, and the amount of information that is passed between queue managers.

Balancing producers and consumers in publish/subscribe networks
An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Related information

Proxy subscriptions in a publish/subscribe network

Reducing the number of unwanted topics in the topic tree

The performance of a publish/subscribe system is improved by reducing the number of unwanted topics in the topic tree. What is an unwanted topic and how do you remove them?

You can create large numbers of topics without affecting performance adversely. However, some ways of using publish/subscribe result in continually expanding topic trees. An exceptionally large number of topics are created once and never used again. The growing number of topics might become a performance problem.

How can you avoid designs that lead to a large and growing number of unwanted topics? What can you do to help the queue manager remove unwanted topics from the topic tree?

The queue manager recognizes an unwanted topic because it has been unused for 30 minutes. The queue manager removes unused topics from the topic tree for you. The 30 minute duration can be changed by altering the queue manager attribute, **TREELIFE**. You can help the queue manager to remove unwanted topics by making sure that the topic appears to the queue manager to be unused. The section, "What is an unused topic?" on page 353 explains what an unused topic is.

A programmer, designing any application, and especially designing a long running application, considers its resource usage: how much resource the program requires, are there any unbounded demands, and any resource leaks? Topics are a resource that publish/subscribe programs use. Scrutinize the use of topics just like any other resource a program uses.

What is an unused topic?

Before defining what an unused topic is, what exactly counts as a topic?

When a topic string, such as `USA/Alabama/Auburn`, is converted into a topic, the topic is added to the topic tree. Additional topic nodes, and their corresponding topics, are created in the tree, if necessary. The topic string `USA/Alabama/Auburn` is converted into a tree with three topics.

- `USA`
- `USA/Alabama`
- `USA/Alabama/Auburn`

To display all the topics in the topic tree, use the **runmqsc** command `DISPLAY TPSTATUS('#') TYPE(TOPIC)`.

An unused topic in the topic tree has the following properties.

It is not associated with a topic object

An administrative topic object has a topic string that associates it with a topic. When you define the topic object `Alabama`, if the topic, `USA/Alabama`, it is to be associated with does not exist, the topic is created from the topic string. If the topic does exist, the topic object and the topic are associated together using the topic string.

It does not have a retained publication

A topic with a retained publication results from a publisher putting a message to a topic with the option `MQPMO_RETAIN`.

Use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama') RETAINED` to check if `USA/Alabama` has a retained publication. The response is `YES` or `NO`.

Use the **runmqsc** command `CLEAR TOPICSTR('USA/Alabama') CLTRTYPE(RETAINED)` to remove a retained publication from `USA/Alabama`.

It has no child topics

`USA/Alabama/Auburn` is a topic with no child topics. `USA/Alabama/Auburn` is the direct child topic of `USA/Alabama`.

Display the direct children of `USA/Alabama` with the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama/+')`.

There are no active publishers to the node

An active publisher to a node is an application that has the topic open for output.

For example, an application opens the topic object named **Alabama** with open options `MQOO_OUTPUT`.

To display active publishers to `USA/Alabama` and all its children, use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama/#') TYPE(PUB) ACTCONN`.

There are no active subscribers to the node

An active subscriber can either be a durable subscription, or an application that has registered a subscription to a topic with `MQSUB`, and not closed it.

To display active subscriptions to USA/Alabama, use the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama') TYPE(SUB) ACTCONN.

To display active subscriptions to USA/Alabama and all its children, use the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama/#') TYPE(SUB) ACTCONN.

Managing the number of topics in a topic tree

In summary, there are a number of ways to manage the number of topics in a topic tree.

Display TPCOUNT

Use the **runmqsc** command DISPLAY PUBSUB ALL periodically to display the **TPCOUNT** property. This is the number of topic nodes in the topic tree. If the number is growing it might indicate that a shorter TREELIFE is required, or that a redesign of the topics themselves is required.

Modify TREELIFE

An unused topic has a lifetime of 30 minutes by default. You can make the lifetime of an unused topic smaller.

For example, The **runmqsc** command, ALTER QMGR TREELIFE(900), reduces lifetime of an unused topic from 30 minutes to 15 minutes.

Exceptionally, restart the queue manager

When the queue manager is restarted, the topic tree is reinitialized from topic objects, nodes with retained publications, and durable subscriptions. Topics that had been created by the operation of publisher and subscriber programs are eliminated.

As a last resort, if the growth in unwanted topics has been the cause of performance problems in the past, restart the queue manager.

Related information

Topic trees

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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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.

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 WebSphere 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

Part Number:

(1P) P/N: