CICS Transaction Server for z/OS
Version 5 Release 5

Using IBM MQ with CICS

Note

Before using this information and the product it supports, read the information in "Notices" on page 149.

This edition applies to the IBM® CICS® Transaction Server for z/OS® Version 5 Release 5 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF provides introductory and guidance information on configuring and using the CICS-MQ adapter and the CICS-MQ bridge, and on defining and maintaining your CICS-MQ environment.

For details of the terms and notation used, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on January 20th 2020.

Chapter 1. CICS and IBM MQ

CICS provides a number of ways to run work in CICS using IBM MQ messages. From Java applications, you can access IBM MQ through IBM MQ classes for Java or IBM MQ classes for JMS. From other applications, you can access IBM MQ through two interfaces that are supplied with CICS: the CICS-MQ adapter and the CICS-MQ bridge.

For a summary of the concepts and architecture of IBM MQ on z/OS, see IBM MQ for z/OS concepts in the IBM MQ product documentation in the IBM MQ documentation. To check which versions of IBM MQ work with CICS, see System requirements for IBM MQ and look under the z/OS entry for the relevant release of IBM MQ.

How CICS works with IBM MQ

CICS provides two resources in support for IBM MQ:

- MQCONN for defining the attributes of the CICS-MQ connection
- MQMONITOR for defining the attributes of an IBM MQ message consumer.

The resources are described in more detail in "CICS resources for MQ support: MQCONN and MQMONITOR" on page 12.

Accessing IBM MQ through the CICS-MQ adapter

The CICS-MQ adapter is supplied with CICS and communicates with IBM MQ as an external resource manager using the CICS Resource Manager Interface (RMI). It is often known as the *trigger monitor* because it allows you to initiate user transactions in CICS through the MQ message trigger mechanism. You can put a message to an application queue that is enabled for triggering. When the message releases the trigger, IBM MQ sends a trigger message to CICS to initiate the specified user transaction. The CICS-MQ adapter provides two main facilities:

- A set of control functions for use by system programmers and administrators to manage the adapter.
- Message Queue Interface (MQI) support for CICS applications.

The CICS-MQ adapter uses standard CICS command-level services where required; for example, **EXEC CICS ASSIGN** and **EXEC CICS ABEND**. Part of the CICS-MQ adapter runs under the control of the transaction that issues the messaging requests. Therefore, these calls for CICS services look as though they are issued by the transaction.

In a CICS multi-region operation or inter-system communication (ISC) environment, each CICS address space can have its own attachment to the queue manager subsystem. A single CICS address space can connect to only one queue manager at a time. However, each address space can connect to a queue manager subsystem.

For more information, see "How it works: the CICS-MQ adapter" on page 3.

Accessing IBM MQ through the CICS-MQ bridge

The CICS-MQ bridge is supplied with CICS. It allows direct access from IBM MQ applications to applications on your CICS system, by sending a message that contains the name of the target program to an IBM MQ queue. Unlike MQ triggering, the CICS-MQ bridge provides direct access to MQ-unaware CICS applications. When you use the CICS-MQ bridge to run a program or transaction on CICS, the CICS program contains no IBM MQ calls because the bridge enables implicit Message Queue Interface (MQI) support. Therefore, you can re-engineer legacy CICS applications to be controlled from other operating systems by IBM MQ messages, without having to rewrite, recompile, or relink them. The IBM MQ messages include the IBM MQ CICS information header (the MQCIH structure) to supply control options for the CICS applications.

The following types of CICS application are suitable for use with the CICS-MQ bridge:

- CICS programs that are called by the **EXEC CICS LINK** command, known as Distributed Program Link (DPL) programs. The programs must conform to the DPL subset of the CICS API; that is, they must not use CICS terminal or sync point facilities. You can use the CICS-MQ bridge to run a single CICS program, or a set of CICS programs that form a unit of work.
- CICS transactions that were designed to be run from a 3270 terminal, known as 3270 transactions. The transactions can use Basic Mapping Support (BMS) or terminal control commands. They can be conversational or part of a pseudoconversation. They are permitted to issue sync points.

For more information, see About the CICS-WebSphere MQ bridge.

Accessing IBM MQ from a Java application

A Java application that runs in CICS can access IBM MQ through two interfaces:

- Using IBM MQ classes for JMS.
- Using IBM MQ classes for Java

The Java application connects to IBM MQ in either of the following ways:

- In MQ client mode, as an IBM MQ MQI client by using TCP/IP
- In MQ bindings mode, connecting directly to IBM MQ by using the Java Native Interface (JNI).

For more information, see Accessing IBM MQ from Java applications. For details about the IBM MQ communication types, see Connection modes for IBM MQ classes for JMS in the IBM MQ product documentation.

IBM MQ classes for JMS

Java Message Service (JMS) is an API defined by the Java Enterprise Edition (Java EE) specification that allows applications to send and receive messages using reliable, asynchronous communication. It provides the ability to use a range of messaging providers including IBM MQ, the WebSphere Liberty-embedded JMS messaging provider or a third party messaging provider. IBM MQ classes for JMS implement the interfaces that are defined in the `javax.jms` package, and also provides two sets of extensions to the JMS API. Both Java™ Platform, Standard Edition (Java SE) and Java Platform, Enterprise Edition (Java EE) applications can use IBM MQ classes for JMS.

From Version 8.0, IBM MQ supports the JMS 2.0 version of the JMS standard. This implementation offers all the features of the classic API but requires fewer interfaces and is simpler to use. For more information, see The JMS model in the IBM MQ documentation and the JMS 2.0 specification at Java.net.

The IBM MQ classes for JMS can be used in:

- An OSGi JVM server, with restrictions.
- A CICS standard-mode Liberty JVM server when the JMS application connects to a queue manager, using either bindings mode or client mode transport.
- A CICS integrated-mode Liberty JVM server when the JMS application connects to a queue manager, using client mode transport.

New applications should use the IBM MQ classes for JMS rather than IBM MQ classes for Java.

See “How it works: IBM MQ classes for JMS with CICS” on page 9 for more details, including the types of communication and the restrictions.

IBM MQ classes for Java

IBM MQ classes for Java provide a Java variant of the IBM Message Queue Interface (MQI) for use by Java applications that run in CICS. A call request to IBM MQ is transformed into an MQI call, and is processed as normal by the existing CICS-MQ adapter. The converted requests flow into the CICS-MQ adapter in the same way as MQI requests from any other program (for example, a COBOL program). So there are no operational differences between Java programs and other programs that access IBM MQ.

The IBM MQ classes for Java can be used only in an OSGi JVM server. Although existing applications that use the IBM MQ classes for Java continue to be fully supported, new applications should use the IBM MQ classes for JMS.

For more information, see Using IBM MQ classes for Java in an OSGi JVM server .

How it works: the CICS-MQ adapter

CICS and the CICS-MQ adapter share the same address space. The IBM MQ queue manager runs in its own address space.

Figure 1 on page 3 shows the relationship between CICS, the CICS-MQ adapter, and IBM MQ.

Part of the adapter is a CICS task-related user exit that communicates with the IBM MQ message manager. CICS management modules call the exit directly; application programs call it through the supplied API stub program (CSQCSTUB). For more information about task-related user exits and stub programs, see Task-related user exit programs.

Each CKTI transaction is usually in an `MQGET WAIT` state, ready to respond to any trigger messages that are placed on its initiation queue.

The adapter management interface provides the operation and control functions described in Administering the CICS-MQ adapter.

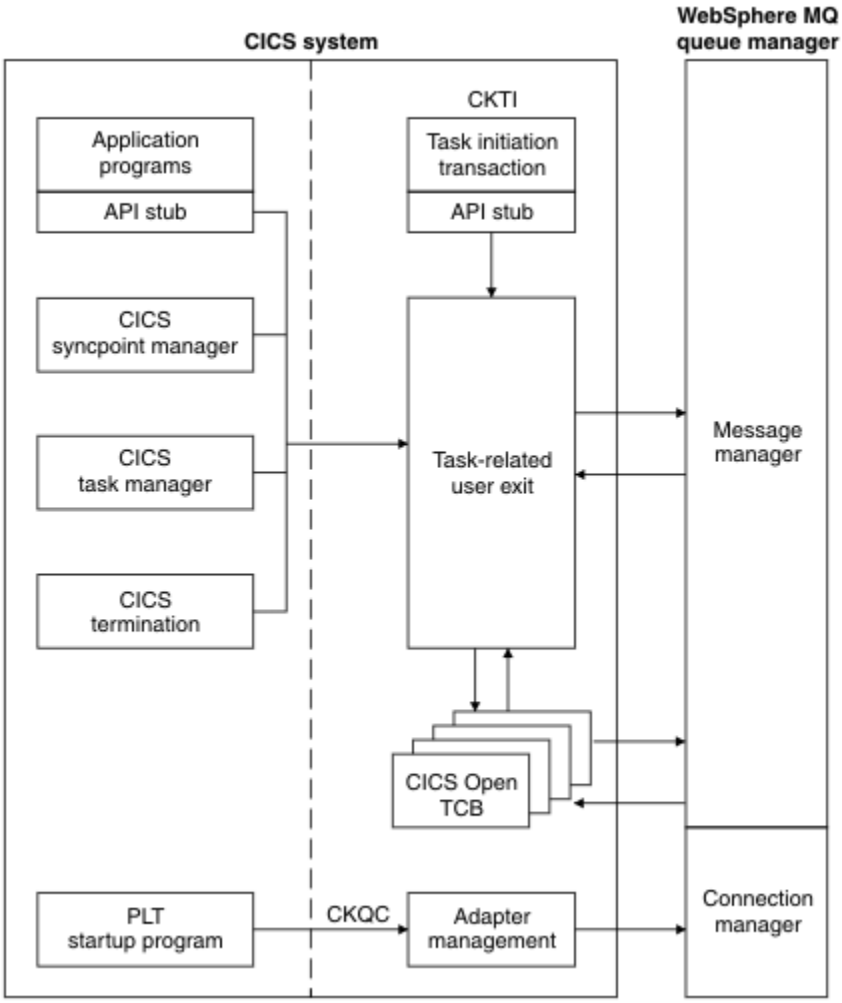


Figure 1. How CICS, the CICS-MQ adapter, and an IBM MQ queue manager are related

The CICS-MQ adapter display function uses two main temporary storage queues for each calling task to store the output data for browsing. The names of the queues are *tttt*CKRT and *tttt*CKDP, where *tttt* is the terminal identifier of the terminal from which the display function is requested. Do not try to access these queues.

The CICS-MQ adapter uses the name DFH.`genericapplid(8)`.QMGR to issue CICS ENQ and CICS DEQ calls during processing; for example, starting and stopping the connection. Do not use similar names for CICS ENQ or DEQ purposes.

Alert monitor (CKAM)

The alert monitor transaction, CKAM, performs two functions: handling pending events that occur as a result of connect or disconnect requests to instances of IBM MQ, and calculating the maximum number of MQGET calls that an MQMONITOR can issue per second.

CKAM and pending events

When pending events occur, the alert monitor generates messages that are sent to the system console.

Pending events are of two kinds:

Deferred connection
A pending event (called a deferred connection) is activated if CICS tries to connect to IBM MQ before a suitable queue manager is started. A suitable queue manager could be either a single named queue manager or a member of an IBM MQ queue-sharing group, depending on the setting for the connection. When a suitable queue manager is available, the CICS-MQ adapter issues a connection request, a connection is made, and the pending event is canceled.

You can have multiple deferred connections, one of which is connected when a suitable queue manager is started.

If you are using an IBM MQ queue-sharing group for the connection, and all the queue managers in the group are unavailable, the CICS-MQ adapter initiates connection to each queue manager in turn, resulting in multiple deferred connections. When one of the queue managers is started, the CICS-MQ adapter makes the connection. If more than one queue manager is started on the same LPAR, the CICS-MQ adapter connects to one of them.

Termination notification
When a connection is successfully made to IBM MQ, a pending event called a termination notification is created. This pending event expires when one of the following occurs:

- The queue manager to which CICS is connected shuts down normally with MODE(QUIESCE). The alert monitor issues a quiesce request on the connection.
- The queue manager shuts down with MODE(FORCE) or ends abnormally.
- The connection is shut down from the CKQC transaction.

The maximum number of pending events that CICS can handle is 99. If this limit is reached, no more events can be created until at least one current event expires.

The alert monitor stops itself when all pending events have expired. It is subsequently restarted automatically by any new connect request.

CKAM and MQMONITORs

If the z/OS Workload Manager health service is active in a CICS region, changes in the region's health state can have an effect on MQMONITORs, in which CKAM plays a part as follows:

- When the region's z/OS WLM health value is less than 100, CKAM calculates the maximum number of MQGET calls that an MQMONITOR can issue per second. For details on the formula that is used to calculate this throttle and the effect of the z/OS Workload Manager health service can have on MQMONITORs, see Effect of z/OS Workload Manager Health service on MQMONITORs.

- When the z/OS WLM health open status is OPENING and the region's health value is greater than zero (for example, during a CICS warm-up process), CKAM starts all MQMONITORs that have been defined with AUTOSTART(YES).
- When the z/OS WLM health open status is CLOSED and the region's health value is zero (for example, by the end of a CICS cool-down process, or after **SET WLMHEALTH IMMCLOSE** is issued), CKAM stops all MQMONITORs.

If CICS encounters an MXT condition, CKAM calculates, as shown in the following formula, the maximum number of MQGET calls that an MQMONITOR can issue per second while this condition exists. This restriction limits the number of tasks being started by MQMONITORs when CICS is at MXT. When the condition no longer exists, CKAM removes the restriction.

Formula: The maximum number of MQGET calls is MXT plus 10%.

To determine whether MXT gating has taken place, see Interpreting transaction statistics.

Task initiator or trigger monitor (CKTI)

CKTI starts a CICS transaction when an MQ trigger message is read; for example, when a message is put onto a specific queue.

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message, containing user-defined data, known as a trigger message, to the initiation queue that has been specified for that message queue. In a CICS environment, you can set up an instance of CKTI to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. CKTI starts another CICS transaction, specified using the DEFINE PROCESS command, which typically reads the message from the application message queue and then processes it. The process must be named on the application queue definition, not the initiation queue.

Each copy of CKTI services a single initiation queue. You can have more than one instance of CKTI monitoring an MQ initiation queue.

How to set up CKTI

An instance of CKTI can be set up in any of the following methods:

Using MQCONN and MQMONITOR resources

You can specify the name of the default initiation queue in the MQCONN resource definition for the CICS region. When you install the MQCONN resource definition, CICS dynamically creates and installs an MQMONITOR resource with the reserved name of DFHMQINI representing the default initiation queue. You can install more MQMONITOR resources to monitor other initiation queues.

An MQMONITOR resource allows the associated transaction that services an MQ queue to restart automatically when the connection to the MQ queue manager is established. You can use CEMT or CICS APIs to manually start or stop an MQMONITOR.

You can change the name of the default initiation queue by changing and reinstalling the MQCONN resource definition to create a new MQMONITOR resource definition. You can also name an alternative default initiation queue if you start the CICS-MQ connection manually.

Using a terminal from the CKQC transaction or a user-written program that links to DFHMQSSQ

The CKQC transaction gives you access to the control functions of the CICS-MQ adapter to initiate, manage, and view the connection between CICS and IBM MQ. To start or stop an instance of CKTI, you can use the **CKQC STARTCKTI** or **CKQC STOPCKTI** command. You can start CKTI from a terminal from the CKQC transaction or from a CICS program that links to the adapter task initiation program, DFHMQSSQ.

To start or stop a copy of CKTI, you must supply the name of the queue that this CKTI is to serve or is now serving.

If you issue **CKQC STARTCKTI** or **CKQC STOPCKTI** commands without specifying an initiation queue, these commands are automatically interpreted as referring to the default initiation queue for the CICS region, which is specified in the INITQNAME attribute of the MQCONN resource, and the request will fail if the implicit MQMONITOR is already installed and started. When you install the MQCONN

resource definition, CICS creates and installs an implicit MQMONITOR resource definition to represent the default initiation queue. You can change the name of the default initiation queue by changing and reinstalling the MQCONN resource definition to create a new implicit MQMONITOR resource definition. You can also name an alternative default initiation queue if you start the CICS-MQ connection manually.

CKQC does not control MQMONITOR resources.

CAUTION: Using the CKQC transaction and MQMONITORs at the same time to manage instances of the CKTI transaction can lead to confusing statistics and STAT of an MQMONITOR because CKQC is not aware of MQMONITORs and because MQMONITORs are not aware of CKTI transactions managed by using CKQC.

Note: When **CKQC STOPCKTI** is issued, all transactions monitoring the initiation queue are stopped, including those associated with MQMONITORs.

Supplied resource definitions for the CICS-MQ adapter

CICS supplies CSD definitions for the CICS-MQ adapter in group DFHMQ as part of DFHLIST. The definitions for sample application programs are provided by IBM MQ in the CSQ4SAMP group.

DFHMQ contains these definitions:

- The supplied adapter programs
- The supplied adapter management transactions
- The supplied sets of BMS maps, required for the adapter panels

Program names have the format DFHMQxxx (and some CSQCxxxx aliases provided for compatibility), and transaction names have the format CKxx.

When the CICS-MQ adapter was shipped with the IBM MQ product, IBM MQ supplied the CSQCAT1 and CSQCKB CSD groups. The CSQCAT1 and CSQCKB groups must not be installed on CICS TS for z/OS, Version 5.5. For information on how to handle these groups, see Setting up the CICS-MQ adapter.

You can install the CSQ4SAMP sample, which contains the definitions for sample application programs, into your CICS CSD. For JCL to install this sample, see Setting up the CICS-MQ adapter. If you want to use CICS program autoinstall rather than define resources in the CICS CSD, you must ensure that the autoinstalled definitions map to those supplied in CICS CSD DFHMQ.

How it works: the CICS-MQ bridge

The CICS-MQ bridge enables an application that is not running in a CICS environment to use IBM MQ messages to run a program or transaction on CICS and get a response.

A non-CICS application can use the CICS-MQ bridge from any environment that has access to a IBM MQ network that encompasses IBM MQ for z/OS. The non-CICS application may wait for a response to come back before it runs the next CICS program (synchronous processing), or it may request one or more CICS programs to run, but not wait for a response (asynchronous processing).

When you use the CICS-MQ bridge to run a program or transaction on CICS, the CICS program contains no IBM MQ calls, because the bridge enables implicit Message Queue Interface (MQI) support. You can therefore reengineer legacy CICS applications to be controlled from other platforms by IBM MQ messages, without having to rewrite, recompile, or relink them. The IBM MQ messages include the IBM MQ CICS information header (the MQCIH structure) to supply control options for the CICS applications.

The following types of CICS application are suitable for use with the CICS-MQ bridge:

- CICS programs that are called using the **EXEC CICS LINK** command, known as DPL programs. The programs must conform to the DPL subset of the CICS API; that is, they must not use CICS terminal or sync point facilities. You can use the CICS-MQ bridge to run a single CICS program, or a set of CICS programs that form a unit of work.

- CICS transactions that were designed to be run from a 3270 terminal, known as 3270 transactions. The transactions can use Basic Mapping Support (BMS) or terminal control commands. They can be conversational or part of a pseudoconversation. They are permitted to issue sync points.

Typically, more complex application programming is required to run a 3270 transaction through the CICS-MQ bridge, because the non-CICS application must interact with the internal logic and flow of control in the CICS transaction. It is preferable to run a DPL program that contains the business logic of the CICS application. However, some CICS applications are not structured with the business logic of the application separated from the presentation logic, so the CICS-MQ bridge lets you communicate with either type of application.

The CICS-MQ bridge uses standard CICS and IBM MQ security features. You can configure it to authenticate, trust, or ignore the user ID of the requestor. The MQCIH structure in the IBM MQ messages provides data for security checking.

The CICS-MQ bridge uses a bridge monitor, which is a transaction with the default name CKBR, to browse a IBM MQ request queue for new requests to run CICS applications. The request queue must reside on the local z/OS queue manager that is connected to the CICS-MQ adapter. The CICS-MQ bridge tasks typically run in the same CICS region as the bridge monitor. The user programs can be in the same or a different CICS region, using CICS transaction routing as necessary. If you need a request to be processed by a specific CICS region, you can name the CICS region in the MQCIH header.

How CICS DPL programs run under the CICS-MQ bridge

An IBM MQ message provides the data needed to run a CICS DPL program. The CICS-MQ bridge builds a COMMAREA or container from this data, and runs the program using **`EXEC CICS LINK`**.

Figure 2 on page 7 shows the sequence of actions taken to process a single message to run a CICS DPL program.

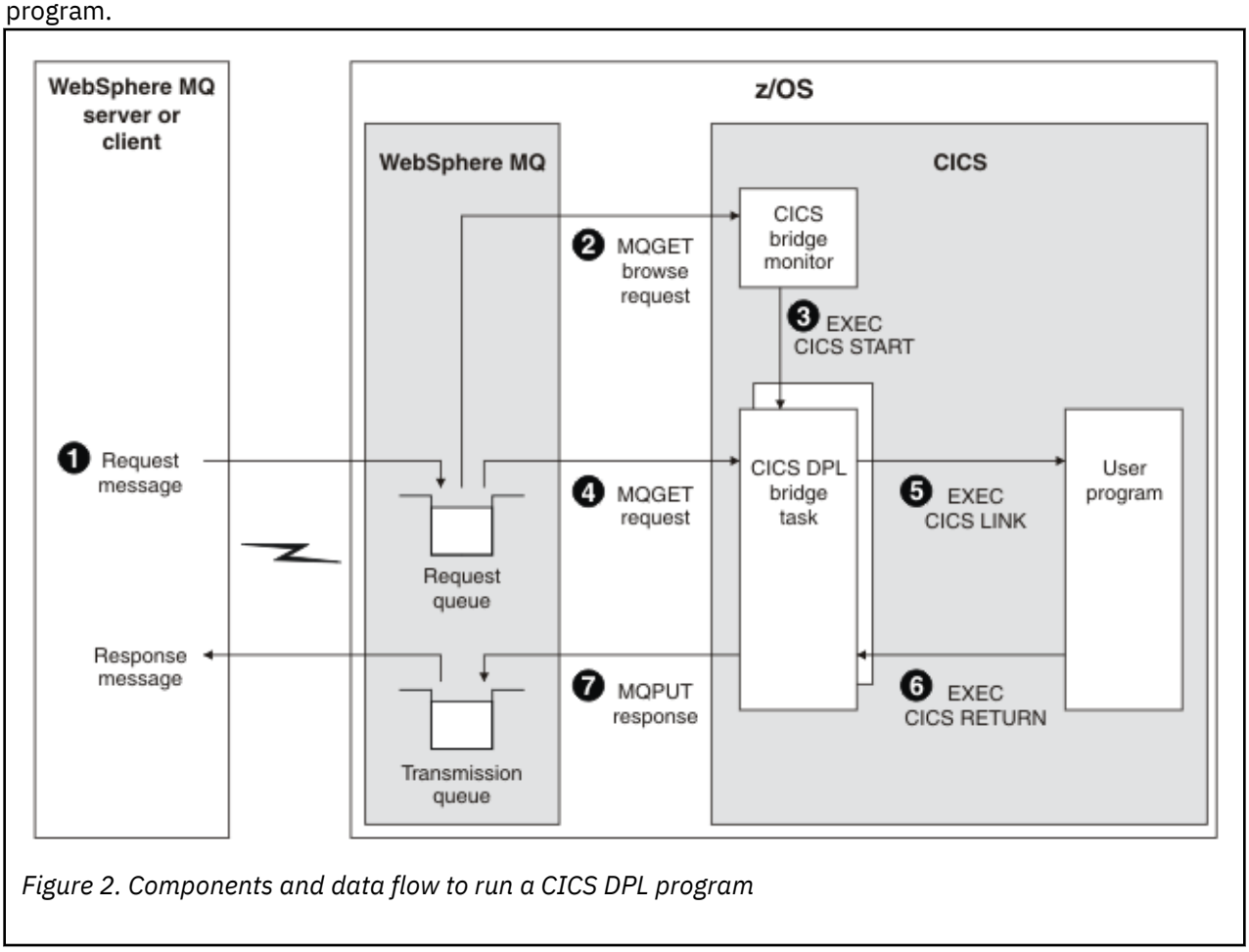


Figure 2. Components and data flow to run a CICS DPL program

1. A message, with a request to run a CICS program, arrives on the request queue.

2. The bridge monitor task, which is constantly browsing the queue, examines the message.
3. The bridge monitor task starts the CICS DPL bridge task using either your specified transaction, or CKBC (if you chose to use a container), or you can use the default CKBP transaction (if you chose to use a COMMAREA). CKBC uses channels and containers, instead of a COMMAREA. The program used is DFHMQBP3. CKBP uses the COMMAREA, and the program used is DFHMQBP0. If you use your own specified transaction, you can select the relevant program you chose to use.
4. The CICS DPL bridge task, for either program DFHMQBP0 or DFHMQBP3, receives the message from the request queue.
5. The DFHMQBP3 program puts the message payload into container DFHREQUEST in channel DFHMQBR_CHANNEL, and issues EXEC CICS LINK to the specified program. The DFHMQBP0 program puts the message payload into a COMMAREA and issues **EXEC CICS LINK** to the specified program.
6. The DFHMQBP3 program expects the application to place its response message payload in container DFHRESPONSE in channel DFHMQBR_CHANNEL. The DFHMQBP0 program expects the application to place its response message payload in the COMMAREA used by the request.

   If the application does not provide a payload for the response message, it should still return a container; in this case it is an empty container. Note that a reply is still sent with just the headers and with no payload.

   For a client to assert that no reply should be sent, omit the reply-to queue name (**ReplytoQueue**) in the request message MQMD.
7. The DFHMQBP3 program puts the content of the DFHRESPONSE container into the reply message payload, and also puts the reply message to the reply-to queue. The DFHMQBP0 program puts the content of the COMMAREA into the reply message payload, and also puts the reply message to the reply-to queue.

The user transaction ends or requests more input. If this flow is the last one in the pseudo conversation, the transaction ends. If the message is not the last one, the transaction waits until the next message is received or the specified timeout interval expires.

Multiple CICS programs

A unit of work can be just a single user program, or it can be multiple user programs. The number of messages that you can send to build up a unit of work is not limited.

In this scenario, a unit of work built from many messages works in the same way, with the exception that the bridge task waits for the next request message in the final step unless it is the last message in the unit of work.

How CICS 3270 transactions run under the CICS-MQ bridge

An IBM MQ message provides the data needed to run a CICS 3270 transaction. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more IBM MQ messages to communicate between the CICS transaction and the IBM MQ application.

The CICS-MQ bridge uses the CICS Link3270 mechanism to access the CICS transaction. Unlike traditional 3270 emulators, the CICS-MQ bridge does not work by replacing the z/OS Communications Server flows with IBM MQ messages. Instead, the message consists of a number of parts called vectors, each of which corresponds to an **EXEC CICS** request. The application is therefore talking directly to the CICS transaction, rather than through an emulator, using the data used by the transaction (known as application data structures or ADSs).

The components and data flow to run a CICS 3270 transaction with the CICS-MQ bridge are as follows:

1. A message, with a request to run a CICS transaction, is put on the request queue.
2. The CICS-MQ bridge monitor task, which is constantly browsing the queue, recognizes that a ‘start unit of work' message is waiting (CorrelId=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS 3270 bridge task is started with the appropriate authority, with a particular user ID (depending on the options used to start the bridge monitor).

4. The CICS-MQ bridge removes the message from the queue and changes task to run a user transaction.
5. Vectors in the message provide data to answer all terminal-related input **EXEC CICS** requests in the transaction.
6. Terminal-related output **EXEC CICS** requests result in output vectors being built.
7. The CICS-MQ bridge builds all the output vectors into a single message and puts it on the reply-to queue.
8. The CICS 3270 bridge task ends. If this flow is the last one in the transaction, the transaction ends. If this message is not the last one, the transaction waits until the next message is received or the specified timeout interval expires.

Multiple CICS transactions

A CICS application often consists of one or more transactions linked together as a pseudoconversation. In general, each transaction is started by the 3270 terminal user entering data onto the screen and pressing an AID key. This model of application can be emulated by a non-CICS application that uses the CICS-MQ bridge.

In this model, a message is built for the first transaction, containing information about the transaction, and input vectors. This message is put on the queue. The reply message consists of the output vectors, the name of the next transaction to be run, and a token that is used to represent the pseudoconversation. The IBM MQQ application builds a new input message, with the transaction name set to the next transaction, and the facility token and remote system ID set to the value returned on the previous message. Vectors for this second transaction are added to the message, and the message is put on the queue. This process is continued until the application ends.

You can include all the IBM MQ messages for multiple transactions in the same bridge session, which reduces monitoring overheads and improves performance.

An alternative approach to writing CICS applications is the conversational model. In this model, the original message might not contain all the data to run the transaction. If the transaction issues a request that cannot be answered by any of the vectors in the message, a message is put onto the reply-to queue requesting more data. The IBM MQ application gets this message and puts a new message back to the queue with a vector to satisfy the request.

How it works: IBM MQ classes for JMS with CICS

IBM MQ classes for JMS are the preferred interfaces to IBM MQ from a Java application that runs in CICS. (The IBM MQ classes for Java continue to be supported but newer applications should use IBM classes for JMS.) The IBM MQ classes for JMS can run in an OSGi JVM server, with restrictions, or in a Liberty JVM server.

On the Java EE platform, IBM MQ classes for JMS supports two types of communication between a component of an application and an IBM MQ queue manager:

Outbound communication

Using the JMS API directly, an application component creates a connection to a queue manager, and then sends and receives messages.

For example, the application component can be an application client, a servlet, a JavaServer Page (JSP), an enterprise Java bean (EJB), or a message driven bean (MDB). In this type of communication, the application server container provides only low-level functions in support of messaging operations, such as connection pooling and thread management.

Inbound communication

A message arriving at a destination is delivered to an MDB, which then processes the message.

Java EE applications use MDBs to process messages asynchronously. An MDB acts as a JMS message listener and is implemented by an `onMessage()` method, which defines how a message is processed. An MDB is deployed in the EJB container of an application server. The precise way in which an MDB is configured depends on which application server you are using, but the configuration

information must specify which queue manager to connect to, how to connect to the queue manager, which destination to monitor for messages, and the transactional behavior of the MDB. This information is then used by the EJB container. When a message satisfying the selection criteria of the MDB arrives at the specified destination, the EJB container uses IBM MQ classes for JMS to retrieve the message from the queue manager, and then delivers the message to the MDB by calling its `onMessage()` method.

IBM MQ classes for JMS in an OSGi JVM server

The following restrictions apply:

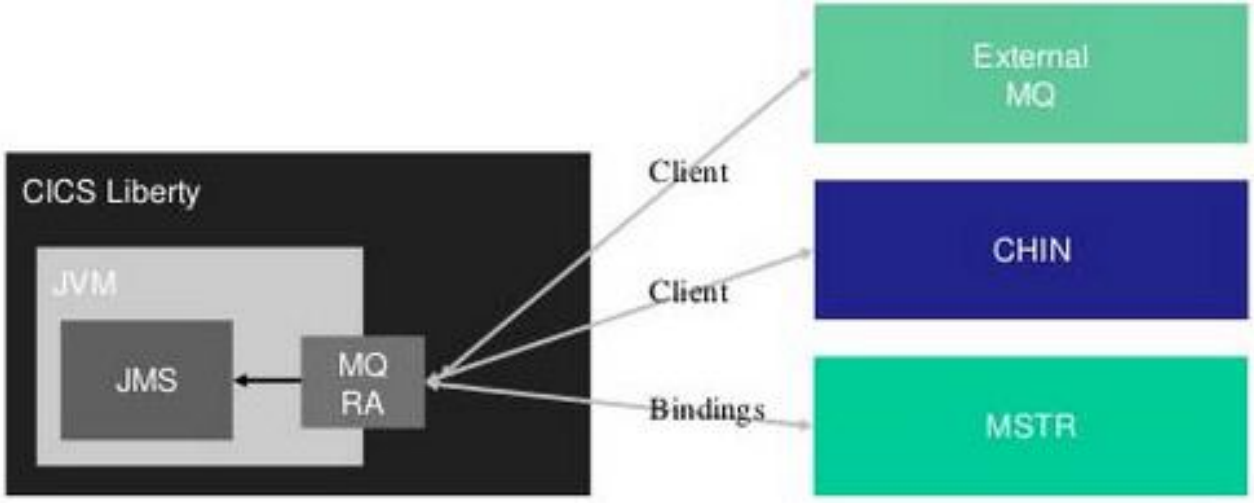
- Using IBM MQ classes for JMS in a CICS OSGi JVM server is only supported in CICS Version 5.2 or later.
- Client mode connections are not supported.
- Connections are only supported to queue managers in IBM MQ Version 7.1 or IBM MQ Version 8.0 or later. The PROVIDERVERSION attribute on the connection factory must be either unspecified, or a value greater than, or equal to, 7.
- Using any of the XA connection factories, for example `com.ibm.mq.jms.MQXAConnectionFactory`, is not supported.

IBM MQ classes for JMS in a CICS standard-mode Liberty JVM server

A CICS standard-mode Liberty JVM server when the JMS application connects to a queue manager using either bindings mode or client mode connections.

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in client mode
- The IBM MQ resource adapter can connect to any in-service version of IBM MQ for z/OS in bindings mode when there is no CICS connection (active CICS MQCONN resource definition) to the same queue manager from the same CICS region.

To connect to IBM MQ from Liberty, you need the IBM MQ resource adapter at Version 9.0.1 or later. Liberty does not contain the IBM MQ resource adapter so you must get it from Fix Central (see Installing the resource adapter in Liberty).

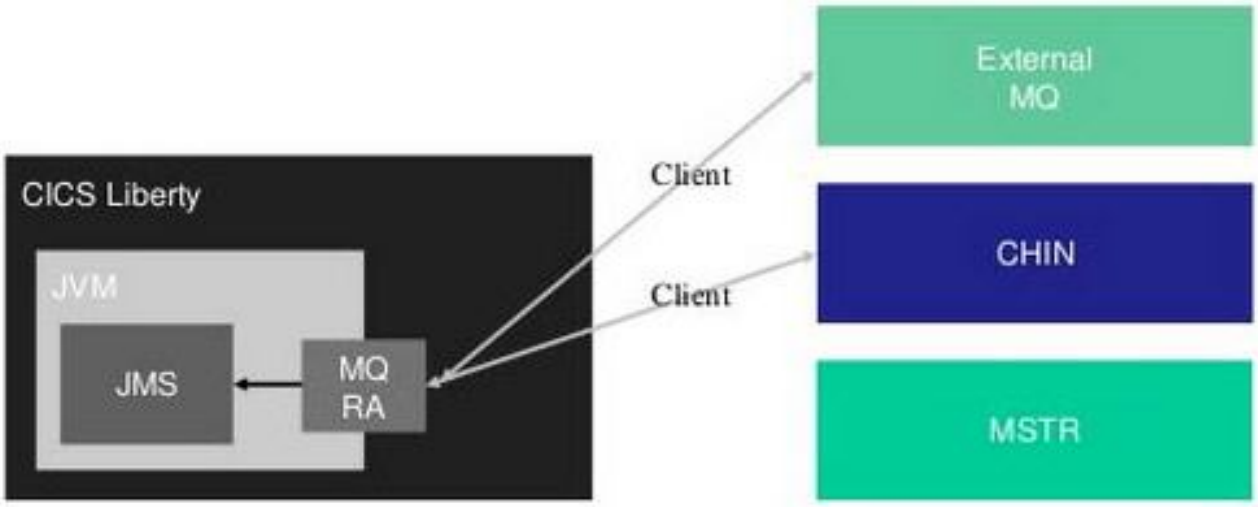


IBM MQ classes for JMS in a CICS integrated-mode Liberty JVM server

CICS integrated-mode Liberty JVM server when the JMS application connects to a queue manager using only client mode connections.

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in client mode.
- Bindings mode connection is not supported.

To connect to IBM MQ from Liberty, you need the IBM MQ resource adapter at Version 9.0.1 or later. Liberty does not contain the IBM MQ resource adapter so you must get it from Fix Central (see Installing the resource adapter in Liberty).



Where next?

For information about developing applications with IBM MQ classes for JMS with CICS, see Using IBM MQ classes for JMS in an OSGi JVM server or Using IBM MQ classes for Java in a Liberty JVM server in the CICS documentation. For information about setting up the JVM server for the applications, see Configuring a Liberty JVM server to support JMS

For information about IBM MQ classes, see Using IBM MQ classes for JMS in the IBM MQ documentation.

CICS-MQ transaction tracking support

CICS transaction tracking support includes tasks that are initiated by both IBM MQ trigger and bridge monitors.

CICS adapter tracking support includes tasks that are initiated by IBM MQ trigger or bridge monitors.

IBM MQ trigger or bridge monitor tasks are supported by CICS transaction tracking by setting adapter fields in the origin data of the task that they start within a CICS region. The following adapter fields are intended to be used in hierarchical order. The adapter identifier specifies the general information, and the adapter data 3 specifies the most specific information. This method provides a uniform manner to identify and isolate tasks.

1. Originating Adapter ID

   The product identifier for the version of IBM MQ queue that is being used. For example, ID=IBM WebSphere® MQ for z/OS V7.0.1
2. Originating Adapter Data 1

   The name of the IBM MQ queue manager. For example, QMGR=RQ38.
3. Originating Adapter Data 2

   The name of the initiation queue from which a trigger monitor retrieved the trigger message and started this task. For example, INITQ=ALIINIT1.

   The name of the initiation queue is not applicable for the CICS-MQ Bridge scenario, so Adapter Data 2 sets to the default value. For example, INITQ=Not Applicable.
4. Originating Adapter Data 3

The name of the application message queue on which a message is put, as a result the task is started. For example, QNAME=ALITRIG1.

For remote transactions, the adapter data that applies to the mirror transaction when it is first attached is used for all subsequent START commands that use that mirror, regardless of what is set on their individual START commands.

CICS resources for MQ support: MQCONN and MQMONITOR

MQCONN is a CICS resource that defines the attributes of the connection between CICS and IBM MQ. MQMONITOR is a CICS resource that defines the attributes of an MQ message consumer, such as the trigger monitor transaction CKTI.

About MQCONN

You must install an MQCONN resource before you start the connection between CICS and IBM MQ.

Only one MQCONN resource definition can be installed at a time in a CICS region. You can install an MQCONN resource only when CICS is not connected to IBM MQ, so if you change your MQCONN definition and reinstall it, you must stop the connection first.

The MQCONN resource specifies the following settings for the connection between CICS and IBM MQ:

- The name of either a single MQ queue manager or a queue-sharing group of MQ queue managers, to which CICS connects (MQNAME attribute).
- The resynchronization strategy that CICS adopts if the connection ends with units of work outstanding (RESYNCMEMBER attribute).
- The name of the default initiation queue for the connection (INITQNAME attribute).

You can use the CICS Explorer®, CEDA, CEMT, or SPI commands to manage the MQCONN resource, and you can also manage it using CICSPlex® SM.

About MQMONITOR

The MQMONITOR resource specifies the following settings for a CICS MQ monitor:

- The automatic start strategy of the MQMONITOR, which enables the MQMONITOR to start automatically when the connection to an MQ queue manager is established (AUTOSTART attribute).
- The status of the resource (ENABLESTATUS attribute).
- Data, if any, to be passed to the transaction monitoring the queue, for example, the CICS-MQ bridge (MONDATA attribute).
- The name of the MQ queue to be monitored (QNAME attribute).
- The user ID that the task monitoring the MQ queue will use (MONUSERID attribute).
- The state of the MQ monitor (MONSTATUS attribute)
- The transaction ID of the task monitoring the MQ queue (TRANSACTION attribute).
- The user ID to be used by default for starting application transactions if no suitable user ID is available from any other source (USERID attribute).

You can install any number of MQMONITOR resources to process incoming messages on MQ queues including initiation queues. You can have more than one MQMONITOR resource processing incoming messages on an MQ queue.

If you want to use an MQMONITOR resource to process incoming messages on an MQ initiation queue, you should define the MQMONITOR resource with the TRANSACTION attribute set to CKTI (or leave blank) and the QNAME attribute set to the name of the MQ initiation queue.

When a user-written MQ monitor program, rather than the default CKTI, is to be used with an MQMONITOR resource, it is the responsibility of the user-written program to get messages directly from application queues and perform the required logic. When you code your program, ensure to follow the

guidelines described in Developing and using user-written MQ trigger monitors and MQ message consumers.

You can use the CICS Explorer, CEDA, CEMT, or SPI commands to manage the MQMONITOR resource, and you can also manage it using CICSPlex SM.

Be aware that when security checking is active, CICS performs security checks on the user ID associated with the transaction that attempts to set the MQ monitor state to started. You must ensure that the user ID is a surrogate of the user ID defined in MONUSERID and is authorized to start transactions associated with the MONUSERID. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).

What affects MQMONITORs

Several aspects of your CICS region can affect the number of MQGET calls that an MQMONITOR can issue:

Active z/OS Workload Manager Health service

If z/OS Workload Manager Health service is active in the CICS region, MQMONITORs react to the region's health state and might be subject to the setting of the z/OS Workload Manager Health service open status. For details, see Effect of z/OS Workload Manager Health service on MQMONITORs.

CICS under MXT condition

If CICS encounters an MXT condition, a restriction is imposed on the number of MQGET calls an MQMONITOR can issue per second when this condition exists. For more information, see “Alert monitor (CKAM)” on page 4.

About the dynamically installed MQMONITOR resource DFHMQINI

When you install an MQCONN resource that specifies a value in the **INITQNAME** attribute, CICS also dynamically creates and installs an MQMONITOR resource with the reserved name of DFHMQINI representing the default initiation queue.

DFHMQINI has the following attributes:

QNAME
Specifies the initiation queue name.

TRANSACTION
CKTI (by default)

MONUSERID
This attribute is obtained as follows:

- From the **PLTPIUSR** system initialization parameter, if available
- Otherwise, from the CICS region user ID

USERID
The value is the CICS default user ID.

DFHMQINI is automatically started when the MQ connection is established, and the user ID associated with the CKTI transaction is obtained from **MONUSERID** if security checking is active, or from the user ID of the transaction that set the state of the MQMONITOR resource to started if security checking is disabled.

You can use the **EXEC CICS INQUIRE MQMONITOR** or **CEMT INQUIRE MQMONITOR** command to inquire on the **QNAME** attribute of the dynamically created MQMONITOR resource. If you want to change this **QNAME** value, you must first change the **INITQNAME** attribute of the MQCONN resource and then reinstall the MQCONN resource. When you discard an MQCONN resource that includes a setting for the **INITQNAME** attribute, the dynamically created MQMONITOR resource and any user-defined MQMONITOR resources are also discarded.

CAUTION: Using the CKQC transaction and MQMONITORs at the same time to manage instances of the CKTI transaction can lead to confusing statistics and STAT of an MQMONITOR because CKQC is not aware of MQMONITORs and because MQMONITORs are not aware of CKTI transactions managed by using CKQC.

Effect of z/OS Workload Manager health service on MQMONITORs

If the z/OS Workload Manager health service is active in a CICS region, CICS can have a warm-up process after the completion of system initialization and can have a cool-down process before shutdown. During either system warm-up or cool-down, CICS adjusts the region's health value to control flow of work into the region. Changes in the region's health state can have an effect on MQMONITORs; for example, when MQMONITORs with attribute AUTOSTART(YES) are started or stopped. When the health value is less than 100, all started MQMONITORs are subject to a throttle limiting the messages that a single MQMONITOR can process.

Effect on when MQMONITORs are started

If MQMONITORs are set to be started automatically, they will be started when the WLMHEALTH open status is OPEN or OPENING and the region's health value is greater than zero.

For example, when system initialization is completed, CICS will start to call the z/OS WLM Health API (IWM4HLTH) at preset intervals to increment the region's health value from zero until it reaches 100%. So, after the first interval, the health value will be greater than zero, and then MQMONITORs with attribute AUTOSTART(YES) are started. The interval and adjustment value is specified in the **WLMHEALTH** system initialization parameter.

Note: If the z/OS Workload Manager health service is disabled in a CICS region (that is, **WLMHEALTH** is set to OFF), such MQMONITORs will be started immediately after control is given to CICS at the end of system initialization.

Effect on when MQMONITORs are stopped

MQMONITORs with attribute AUTOSTART(YES) will be stopped when the WLMHEALTH open status is CLOSED and the region's health value is zero. For example, if **SET WLMHEALTH OPENSTATUS(CLOSE)** is issued, the region's health value is decremented at intervals, and when it is zero, such MQMONITORs in the region are stopped.

MQMONITORs with attribute AUTOSTART(YES) will be stopped immediately when **SET WLMHEALTH OPENSTATUS(IMMCLOSE)** is issued.

Throttle on the number of MQGET calls per second

When the region's z/OS WLM health value is less than 100, there is a throttle on the number of MQGET calls that an MQMONITOR can issue per second, thereby controlling how many trigger tasks are started. The throttle affects all started MQMONITORs in the region. When the region's health value reaches 100, the throttle on MQGET calls is removed. The throttle is calculated by CKAM as follows:

The maximum number of MQGET calls per second is double the value of the **MXT** value times the region's z/OS WLM health value.

Suppose `WLMHEALTH=(20,25)` and `MXT=400` are specified for the region. The following example illustrates how the amount of maximum MQGET calls per second is changed after each interval during the whole system warm-up process.

Table 1. Example

Time	Health value (%)	Maximum MQGET calls per second
0 (End of system initialization)	0	0
20	25	200

Table 1. Example (continued)		
Time	**Health value (%)**	**Maximum MQGET calls per second**
40	50	400
60	75	600
80 (End of system warm-up)	100	Throttle removed

Likewise, during the system cool-down process, the region's health value starts to decrement at intervals, and the throttle on MQGET calls takes effect. The amount of maximum MQGET calls an MQMONITOR can issue per second is gradually reduced at every interval.

Chapter 2. Configuring connections to IBM MQ

To bring workload into CICS using MQ messages, CICS provides two interfaces, CICS-MQ adapter and CICS-MQ bridge. This section provides setup instructions to make the adapter and bridge available to your CICS region.

Setting up the CICS-MQ adapter

Complete these tasks to make the CICS-MQ adapter available to your CICS region.

Before you begin

If you are not familiar with defining resources in CICS, refer to this information:

- Setting up shared data sets, CSD and SYSIN for information on setting up a CICS region and specifying CICS system initialization parameters.
- CICS resources for information on defining resources to CICS and coding PLT entries.

Procedure

1. Include the IBM MQ library *thlqual*.SCSQAUTH in the STEPLIB concatenation in your CICS procedure. Include the library after the CICS libraries to ensure that the correct code is used.

   thlqual is the high-level qualifier for the IBM MQ libraries.
2. Include the following IBM MQ libraries in the DFHRPL concatenation in your CICS procedure. Include the libraries after the CICS libraries to ensure that the correct code is used.

   ```
   thlqual.SCSQCICS
   thlqual.SCSQLOAD
   thlqual.SCSQAUTH
   ```

   thlqual is the high-level qualifier for the IBM MQ libraries.

   If you are using the CICS-MQ API-crossing exit (CSQCAPX), also add the name of the library that contains the load module for the program.

   The SCSQCICS library is required only if you want to run IBM MQ supplied samples. Otherwise it can be removed from the CICS procedure.
3. Define and install an MQCONN resource for the CICS region.

   The MQCONN resource specifies the attributes of the connection between CICS and IBM MQ, including the name of the default IBM MQ queue manager or queue-sharing group for the connection. For instructions, see Defining and installing an MQCONN resource.
4. Optional: Define and install MQMONITOR resources for the CICS region.

   The MQMONITOR resource defines the attributes of an MQ message consumer, such as the CICS-MQ trigger monitor, CKTI. Using MQMONITOR resources is the recommended method of controlling instances of CKTI. You can also install MQMONITOR resources to monitor other MQ queues. For more information, see “Defining and installing MQMONITOR resources” on page 20.
5. If you want the CICS-MQ adapter to connect automatically to IBM MQ during CICS initialization, choose and specify one of the following methods:

   - Specify the CICS system initialization parameter **MQCONN=YES**, which is the recommended way to connect to IBM MQ automatically.
   - Include the DFHMQCOD program in your CICS PLTPI list, following the instructions in Specifying DFHMQCOD or your own program in the PLTPI list in Configuring. The PLTPI list is the program list table (PLT) with the suffix that you specify on the **PLTPI** system initialization parameter.

- Write your own program, following the instructions in Writing a PLTPI program to start the connection to WebSphere MQ, and include it in your CICS PLTPI list, following the instructions in Specifying DFHMQCOD or your own program in the PLTPI list in Configuring.

An MQCONN resource definition must be installed before CICS can start the connection to IBM MQ. When you start the connection automatically at CICS initialization, for an initial or cold start, the MQCONN resource definition must be present in one of the groups named in the list or lists named by the **GRPLIST** system initialization parameter. For a warm or emergency start of CICS, the MQCONN resource definition must have been installed by the end of the previous CICS run.

6. If you are using the CICS-MQ adapter in a CICS system that has interregion communication (IRC) to remote CICS systems, ensure that the IRC facility is OPEN before you start the adapter, by specifying the CICS system initialization parameter IRCSTRT=YES.

   The IRC facility must be OPEN if the IRC access method is defined as cross-memory; that is, ACCESSMETHOD(XM).

7. Ensure that the coded character set identifiers (CCSIDs) used by your queue manager and by CICS, and the UTF-8 and UTF-16 code pages are configured to z/OS conversion services.

   The CICS code page is specified in the **LOCALCCSID** system initialization parameter.

8. Update your CICS CSD as follows:

   a) If you do not share your CSD with earlier releases of CICS, remove the groups CSQCAT1 and CSQCKB, and any copies of those groups or of items from those groups, from your CSD. You must also delete the CKQQ TDQUEUE from group CSQCAT1.

      The definition for CKQQ is now supplied in the CICS CSD group DFHDCTG.

   b) If you do share your CSD with earlier CICS releases, ensure that CSQCAT1 and CSQCKB, and any copies of those groups or of their content, are not installed for CICS TS 4.1 or CICS TS 3.2. You must also delete the CKQQ TDQUEUE from group CSQCAT1.

      The definition for CKQQ is now supplied in the CICS CSD group DFHDCTG.

      For CICS TS releases earlier than CICS TS 3.2, install the CSQCAT1 and CSQCKB groups as part of a group list, after installing DFHLIST, to override group DFHMQ and correctly install the required definitions.

9. If you want to install the CSQ4SAMP sample, which contains the definitions for sample application programs, into your CSD, add this fragment of JCL to your CSD upgrade (DFHCSDUP) job:

```
//SYSIN  DD DSN=thlqual.SCSQPROC(CSQ4S100),DISP=SHR
//       DD *
    ADD GROUP(CSQ4SAMP) LIST(yourlist)
/*
```

   where *yourlist* is a list of groups to be installed by CICS during a cold start of the system, specified in the **GRPLIST** system initialization parameter for CICS.

   Note: If you use the CEDA transaction to install redefined adapter resources in an active CICS system, you must first shut down the adapter and wait until the alert monitor has finished its work.

10. Update the IBM MQ definitions for the dead-letter queue, default transmission queue, and CICS-MQ adapter objects.

    You can use the sample CSQ4INYG, but you might need to change the initiation queue name to match the default initiation queue name in the MQINI resource definition for your CICS region. You can use this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. Using the CSQUTIL utility is preferable because you do not then have to redefine these objects each time that you restart IBM MQ.

Defining and installing an MQCONN resource

You must install an MQCONN resource before CICS can start the connection to IBM MQ.

About this task

The MQCONN resource specifies the attributes of the connection between CICS and IBM MQ.

Procedure

1. Create an MQCONN resource with a name and group of your choice, using one of the methods listed in Ways of defining CICS resources.

   If you plan to start the CICS-MQ connection automatically at CICS initialization, place the MQCONN resource in one of the groups named in the list or lists named by the CICS **GRPLIST** system initialization parameter.
2. Specify the **MQNAME** attribute as the 1 - 4 character name of either a single IBM MQ queue manager or a queue-sharing group of IBM MQ queue managers.

   If you specify a queue-sharing group, CICS uses any queue manager in the group, rather than waiting for a single queue manager. Queue-sharing groups increase flexibility when you reconnect to IBM MQ and help you standardize this aspect of CICS setup across CICS regions and z/OS images.
3. If you have specified a queue-sharing group of IBM MQ queue managers, use the **RESYNCMEMBER** attribute to select an appropriate resynchronization strategy for CICS if the connection is lost.
   - RESYNCMEMBER(YES), which is the default, specifies that if there are outstanding units of work for the connection, CICS is to reconnect to the same queue manager, waiting if necessary. This strategy increases the probability of resolving indoubt units of work, but it might increase the time taken to reconnect to IBM MQ.
   - RESYNCMEMBER(NO) specifies that CICS is to make one attempt to reconnect to the same queue manager. If that attempt fails, CICS connects to a different eligible queue manager in the group, and issues message DFHMQ2064 if there are any indoubt units of work with the original queue manager. A queue manager is eligible for connection to a CICS region if it is currently active on the same LPAR as the CICS region. This strategy might reduce the time taken to reconnect to IBM MQ. However, if CICS connects to a different queue manager, indoubt units of work cannot be resolved automatically, and you must resolve them manually.
   - RESYNCMEMBER(GROUPRESYNC) specifies that CICS can connect to any member of the queue-sharing group regardless of any outstanding indoubt units of work. IBM MQ chooses the queue manager that CICS connects to and it asks CICS to resolve indoubt units of work on behalf of all eligible queue managers in the queue-sharing group. This function is called *group unit of recovery*. This option can be used only when running a release of IBM MQ that supports group unit of recovery for CICS and when group unit of recovery has been enabled in the queue managers.
4. Specify the **INITQNAME** attribute as the 1 - 48 character name of a default initiation queue for the connection.
   CICS01.INITQ is the name of the initiation queue defined by the CSQ4INYG object sample.
5. Install the MQCONN resource.

Results

In addition to installing the MQCONN resource, CICS dynamically installs an MQMONITOR resource with the reserved name of DFHMQINI representing the default initiation queue. For details about DFHMQINI, see MQMONITOR resources.

What to do next

If you want to modify and reinstall your MQCONN resource, you must stop the connection between CICS and IBM MQ. You can install an MQCONN resource only when CICS is not connected to IBM MQ.

If you want to change the **QNAME** attribute of the dynamically installed MQMONITOR, you must first change the **INITQNAME** attribute of the MQCONN resource and then reinstall the MQCONN resource.

You can install more MQMONITOR resources for monitoring MQ queues, including initiation queues. You can have more than one MQMONITOR resource monitoring an MQ queue.

If you have concerns about the default settings of MQMONITOR DFHMQINI (for example, migrating to DFHMQINI proves more complicated than anticipated), it's possible to install a user-defined MQMONITOR resource with the name of DFHMQINI. This gives you the flexibility in setting the AUTOSTART, STATUS, MONUSERID and USERID attributes to user-defined values so as to be backward compatible, thus making migration easier. The TRANSACTION attribute must be CKTI.

Defining and installing MQMONITOR resources

You can choose to install MQMONITOR resources to process incoming messages on MQ queues including initiation queues. You can have more than one MQMONITOR resource processing incoming messages on an MQ queue. Using MQMONITOR resources is the recommended method of controlling instances of the CICS-MQ trigger monitor, CKTI.

Before you begin

You can install MQMONITOR resources only if you have previously installed an MQCONN resource.

About this task

You can define and install MQMONITOR resources to control the CICS-MQ trigger monitor, CKTI.

You can also define and install MQMONITOR resources for your user-written trigger monitors or MQ message consumers. When a user-written MQ monitor program, rather than the default CKTI, is to be used with an MQMONITOR resource, it is the responsibility of the user-written program to get messages directly from application queues and perform the required logic. When you code your program, ensure to follow the guidelines described in Developing and using user-written MQ trigger monitors and MQ message consumers.

Note: The CICS-supplied MQ trigger monitor program DFHMQTSK is reserved for use with the CICS-MQ trigger monitor and task initiator transaction CKTI. Any attempt to invoke DFHMQTSK as a user transaction will cause the user transaction to abend with abend code AMQO.

You can install a new MQMONITOR at any time, even when the CICS-MQ adapter is connected to MQ.

You can reinstall (by replacing) an existing MQMONITOR only when the MQMONITOR is disabled and no transaction is using it. Use the **SET MQMONITOR DISABLED STOPPED** command to stop the associated task and disable the resource.

You can discard an MQMONITOR only if it is disabled and there is no associated task. Use the **SET MQMONITOR DISABLED STOPPED** command to stop the associated task and disable the resource.

When you install an MQCONN resource that specifies a value in the **INITQNAME** attribute, CICS also dynamically creates and installs an MQMONITOR resource with the reserved name of DFHMQINI representing the default initiation queue.

For details about DFHMQINI, see MQMONITOR resources.

Procedure

1. Define MQMONITOR resources by using one of the methods listed in Ways of defining CICS resources. For example, you can use the **CREATE MQMONITOR** command to define an MQMONITOR resource.

   Table 2 on page 21 list shows important attributes for creating an MQMONITOR resource.

Table 2. Important MQMONITOR attributes

Attribute	Required / Optional	Default	Description
AUTOSTART	Optional	YES	This attribute controls auto-restart of the MQ monitor. **AUTOSTART(YES)** Enables the transaction as specified in the **TRANSACTION** attribute to restart automatically when the connection to the MQ queue manager is established. **AUTOSTART(NO)** The MQ monitor is not started automatically. After the connection to the MQ queue manager is established, you have to manually start the MQ monitor.
MONUSERID	Required	-	Specifies the user ID to be associated with the transaction monitoring the MQ queue. This attribute is only effective when security checking is active (that is, the **SEC** system initialization parameter is set to YES). If security checking is disabled (that is, **SEC** is set to NO), the user ID to be associated with the MQ monitor transaction is the user ID of the transaction that set the state of the MQMONITOR resource to started.
QNAME	Optional	If **QNAME** is omitted, the value is set to `&APPLID..INITIATION.QUEUE` by default when the resource is installed.	Specifies the name of the MQ queue to be monitored.
STATUS	Optional	ENABLED	This attribute makes the resource available for use in the region.
TRANSACTION	Optional	CKTI	Specifies the 4-character ID of the CICS transaction monitoring the MQ queue. If you are defining an MQMONITOR for processing incoming messages on an MQ initiation queue, specify CKTI or leave blank.

In order for CICS to attempt to automatically start the transaction that is associated with the MQMONITOR, ensure that the MQMONITOR resource has attributes AUTOSTART(YES) and STATUS(ENABLED).

2. Install the MQMONITOR resource.

Checks on definitions of MQMONITOR resources

When you define an MQMONITOR, CICS checks for consistency with other resource definitions in the same group or list.

For an MQMONITOR object, CICS performs the following checks:

- Two MQMONITORs of the same name do not appear in the same list. If they do, a warning message is issued.
- An MQCONN exists in the group or list. If one does not, a warning message is issued. This might not be an error because the MQCONN might exist in another group elsewhere, but a MQCONN must be installed before an MQMONITOR can be installed.

Writing a PLTPI program to start the connection to IBM MQ

You can write your own PLTPI program to start the CICS-MQ connection, based on the sample DFHMQPLT, which is supplied in the SDFHSAMP library.

About this task

Although this sample is written in assembler language, you can write your own program in any language supported by CICS. For more information about writing CICS PLTPI programs, see Writing initialization and shutdown programs .

Your PLTPI program must link to the CICS-MQ adapter program DFHMQQCN, and pass a parameter list that specifies the connection values to be used. The parameter list is described in Starting a connection by linking to DFHMQQCN from a CICS application program.

Procedure

- Your PLTPI program must issue a LINK command like this to link to DFHMQQCN and pass the parameter list:

```
EXEC CICS LINK PROGRAM('DFHMQQCN')
          COMMAREA(CONNPL) LENGTH(length of CONNPL)
```

  In this example, the parameter list is named CONNPL.

  Because no terminals are available at this stage of CICS starting, you must use the COMMAREA option to pass the parameter list.

Results

If you are using MQ monitors, when the MQ connection is started, the MQ monitors are started automatically provided that the MQMONITOR resources have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED).

Specifying DFHMQCOD or your own program in the PLTPI list

If you use DFHMQCOD or your own program to start the connection to IBM MQ automatically during CICS initialization, follow these instructions to add the program to the CICS PLTPI list.

About this task

The PLTPI list is the program list table (PLT) with the suffix that you specify on the PLTPI system initialization parameter. If you specify the CICS system initialization parameter **MQCONN=YES**, you do not need to add DFHMQCOD or your own program to your CICS PLTPI list. The **MQCONN** parameter always uses program DFHMQCOD to start the CICS-MQ connection, and it cannot be customized to use a user-supplied attach program of a different name. If you prefer not to use the **MQCONN** parameter, continue with these steps.

Procedure

1. If your current PLTPI list contains no entry for DFHDELIM, add one.

   DFHMQCOD or your own equivalent program must run during the third stage of CICS initialization and must therefore be added after the entry for DFHDELIM.
2. Use the CICS DFHPLT macro to add DFHMQCOD or your program to the PLTPI list.

   See Figure 3 on page 23.

3. Verify that you have specified the suffix of your PLTPI list correctly on the **PLTPI** system initialization parameter.
4. Verify that an MQCONN resource is present in one of the groups named in the list or lists named by the **GRPLIST** system initialization parameter.

   The MQCONN resource is required to start the connection to IBM MQ.

Results

If you are using MQ monitors, when the MQ connection is started, the MQ monitors are started automatically provided that the MQMONITOR resources have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED).

Example

This example shows you how to code the entry for DFHMQCOD after the entry for DFHDELIM in a CICS PLT called DFHPLT41:

```
DFHPLT41 DFHPLT TYPE=INITIAL,SUFFIX=41
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=ENTRY,PROGRAM=DFHMQCOD
         DFHPLT TYPE=FINAL
         END
```

Figure 3. Sample PLT for use with the CICS-MQ adapter

To use this sample PLT, you specify the suffix 41 on the **PLTPI** system initialization parameter. For more information about coding PLT entries, see Program list table (PLT).

Setting up the CICS-MQ bridge

Complete these tasks to make the CICS-MQ bridge available for a CICS region.

Before you begin

To use the CICS-MQ bridge, both IBM MQ and CICS must be running in the same z/OS image.

Procedure

Complete steps “1” on page 23 through “4” on page 24 in CICS:

1. In CICS, set up the CICS-MQ adapter, following the instructions in “Setting up the CICS-MQ adapter” on page 17.
2. In CICS, make the following modifications to the resource definitions for the CICS-MQ bridge components:
   a) Make sure that CICS temporary storage IDs with the prefix CKB are defined as not recoverable.

      The CICS-MQ bridge uses these temporary storage IDs.

   b) If you want to run 24-bit CICS DPL programs using the CICS-MQ bridge, change the TASKDATALOC attribute for the definition of transaction CKBP, or CKBC, or both, to BELOW.

      CICS DPL programs run under the transaction code CKBP by default. If you run 24-bit programs without changing the TASKDATALOC attribute, a CICS abend AEZC might be issued.

   c) If you want to run your CICS DPL programs under different transaction codes, install copies of the definition of CKBP, or CKBC, or both, and change the transaction names to the ones of your choice.

      Remember to change the TASKDATALOC attribute to BELOW if you are going to run 24-bit programs. You must specify the alternative transaction code in the MQCIH header in your IBM MQ request messages.

For a full list of the CICS-MQ bridge components, which are installed in group DFHMQ, see "Definitions for the CICS-MQ bridge transactions and programs in CICS" on page 25.

3. Optional: Set up a CICS MQ monitor to control the bridge.

   Using an MQMONITOR allows the bridge to automatically restart when the connection to the IBM MQ manager is established. This is one of the recommended methods of controlling the CICS-MQ bridge. For setup instructions, see "Setting up an MQMONITOR for the CICS-MQ bridge" on page 25.

   Note: An alternative of controlling the CICS-MQ bridge is to associate the Bridge Request queue with an initiation queue and to implement a process that will cause transaction CKBR to be started when a message arrives on the Bridge Request queue.

4. If you want to use the ROUTEMEM bridge start parameter, which determines whether mark expired messages are routed to the DLQ, install one of two optional TSMODEL resource definitions in CICS, depending on whether you are using shared or nonshared bridge queues.

   The definitions are as follows:

   a. Nonshared bridge queues

   ```
   DEFINE TSMODEL(DFHMQMDL) GROUP(DFHMQ)
   DESCRIPTION(CICS owned MQ-CICS bridge for RouteMEM non-shared)
        PREFIX(CKBR) LOCATION(AUXILIARY) RECOVERY(NO) SECURITY(NO)
   ```

   b. Shared bridge queues

   ```
   DEFINE TSMODEL(DFHMQMDL) GROUP(DFHMQ)
   DESCRIPTION(CICS owned MQ-CICS bridge - for RouteMEM shared)
        PREFIX(CKBR) LOCATION(AUXILIARY) RECOVERY(NO)
        SECURITY(NO) POOLNAME(MQPOOL)
   ```

   For more information about the ROUTEMEM bridge start parameter, see Administering the CICS-MQ bridge.

Complete steps "5" on page 24 through "8" on page 25 in IBM MQ:

5. In IBM MQ, define a local queue for the request messages.

   The IBM MQ request queue must be local to the CICS-MQ bridge, and it must not be used by any other application.

   You can use the sample thlqual.SCSQPROC(CSQ4CKBM) to define the default queue named SYSTEM.CICS.BRIDGE.QUEUE, or you can define your own.

   If you define your own local queue, you must specify its name whenever you start the bridge monitor transaction CKBR.

   Set the following attributes if you define your own local queue:

   a) Set the **SHARE** attribute so that both the bridge monitor and the bridge tasks can read the queue.

   b) If recovery is required, set the attribute **DEFPSIST(YES)** to define that messages are persistent on the queue by default.

   c) If you want to process messages in FIFO sequence, set the **MSGDLVSQ(FIFO)** attribute.

      Otherwise, messages are processed in priority sequence.

   d) If the request queue is defined with **QSGDISP(SHARED)**, also define it with **INDXTYPE(CORRELID)**.

      This setting is also recommended for nonshared queues.

   e) If you want to start the bridge monitor by triggering, set the attributes **TRIGGER TRIGTYPE(FIRST)PROCESS**(*procid*), where *procid* is a process specifying **APPLICID(CKBR)**.

   f) Consider specifying **BOQNAME** and **BOTHRESH** for the request queue, so that messages are put to the backout queue specified by the **BOQNAME** attribute when a message has been processed and backed out the number of times specified by the **BOTHRESH** attribute, rather than being placed on the dead-letter queue.

      If you specify a backout queue, put a process in place to process messages on this queue.

g) If you are using a container, consider using the **MaxMsgLen** attribute of the queue to limit the size of bridge input messages. The maximum message length that a queue can accommodate is defined by its **MaxMsgLen** attribute.

6. In IBM MQ, define one or more queues to hold the responses.

   The response queues can be local or remote.

   If a response queue is remote, you must define a transmission queue to hold the responses before they are forwarded to the response queue.
7. In IBM MQ, ensure that a dead-letter queue is defined and a procedure is defined for processing messages on this queue.
8. If the bridge is to be accessed remotely from IBM MQ, create channel and transmission queue definitions and a remote queue definition for the request queue.

   For more information about using remote queues, see IBM MQ product documentation.

Definitions for the CICS-MQ bridge transactions and programs in CICS

The definitions for the bridge transactions and programs are provided in group DFHMQ, which is part of grouplist DFHLIST.

CKBC	Bridge ProgramLink transaction for channel and container DPL bridge
DFHMQBP3	CICS-MQ Bridge DPL program to use with channel and container
CKBP	Bridge ProgramLink transaction for COMMAREA DPL bridge. By default, your CICS DPL programs run under this transaction code.
DFHMQBP0	CICS-MQ Bridge DPL program to use with COMMAREA
CKBR	Bridge monitor transaction
DFHMQDCI	Data conversion exit
DFHMQBR0	Bridge monitor program
DFHMQBP1	CICS-MQ Bridge DPL abend handler
DFHMQBR0	CICS-MQ Bridge monitor program
CSQCBP00	CICS-MQ Bridge alias of DPL program
CSQCBR00	CICS-MQ Bridge alias of monitor program
CSQCBP53	MQ V531 Bridge, alias of DPL program
CSQCBP10	MQ V531 Bridge, DPL abend handler
CSQCBR53	MQ V531 Bridge, alias of monitor program
DFHMQDCI	CICS-MQ Bridge data conversion exit
CSQCBDCI	CICS-MQ Bridge alias of data conversion exit

Setting up an MQMONITOR for the CICS-MQ bridge

Optionally, you can define and install an MQMONITOR resource for controlling the CICS-MQ bridge. Using an MQMONITOR resource allows the bridge to automatically restart when the connection to the IBM MQ manager is established. This is one of the recommended methods of controlling the CICS-MQ bridge.

Procedure

1. Define an MQMONITOR resource with the following attributes:

Table 3. MQMONITOR resource attributes for controlling the CICS-MQ bridge

Attribute	Required / Optional	Default	Description
AUTOSTART	Optional	YES	This attribute controls auto-restart of the MQ monitor: **AUTOSTART(YES)** Enables the bridge to restart automatically when the connection to the IBM MQ queue manager is established. **AUTOSTART(NO)** The MQ monitor is not started automatically. After the connection to the IBM MQ queue manager is established, you have to manually start the MQ monitor.
MONDATA	Required	-	Specify the parameters to be passed to the CKBR transaction.
MONUSERID	Required	-	Specify the user ID to be associated with CKBR. This attribute is only effective when security checking is active (that is, the **SEC** system initialization parameter is set to YES).
STATUS	Optional	ENABLED	This attribute makes the resource available for use in the region.
TRANSACTION	Required	CKBR	Specifies the 4-character transaction ID of the CICS-MQ bridge.

Note: AUTOSTART(YES) and STATUS(ENABLED) enable the bridge to start automatically when the connection to the MQ manager is established.

2. Install the MQMONITOR resource.

Setting up multiple CICS-MQ bridge monitors

If you expect a high volume of requests for CICS applications through the CICS-MQ bridge, consider setting up additional bridge monitors in further CICS regions. You might also want to use additional bridge monitors to provide more granular security.

About this task

The default transaction ID for the bridge monitor is CKBR, but you can change this or define additional transaction IDs.

When you set up multiple bridge monitors, you can create a separate request queue for each bridge monitor for its sole use. The advantage of this approach is that you may give each bridge monitor different service characteristics. However, the disadvantage is that applications must know the name of the appropriate request queue to use when making requests through the CICS-MQ bridge. For this reason, you might prefer to have several bridge monitors share the same request queue.

If you decide to use a shared request queue with several bridge monitors, ensure that you follow these requirements:

Procedure

- All bridge monitors must be associated with queue managers at WebSphere Version 6 or later. Do not attempt to run multiple bridge monitors on a shared queue with bridge monitors attached to a system earlier than WebSphere MQ Version 5.3.1, which leads to unpredictable results.
- Each CICS region that is running a bridge monitor must have a unique SYSID.
- All transactions in a 3270 pseudoconversation must specify the remote SYSID returned by the first transaction in all subsequent messages in the pseudoconversation. The remote SYSID is required even if you use CICS transaction routing facilities to direct the transactions to other CICS regions.
- You must define CICS ISC links between all the CICS regions where bridge monitors are running.
- If you use passtickets for user validation, all bridge monitors for a queue must specify the same APPLID using the PASSTKTA parameter when the bridge monitors are started.

What to do next

For problem determination with multiple CICS bridge monitors, you might have to look at all the logs of all the CICS regions to find any error messages produced by the bridge. You can use the command DISPLAY QSTATUS(queuename) TYPE(HANDLE) on each queue manager to show which CICS regions have the queue open.

Controlling CICS-MQ bridge throughput

You can control the throughput of the bridge by putting the bridge program transaction, CKBP, in a class of its own and setting the CLASSMAXTASK to suit your requirements.

Request messages browsed by the bridge monitor CKBR are marked and hidden for a period of time specified as the Mark Browse Interval (MarkBint) to allow time for CKBP to get the message once it has been started by CKBR. If CKBP does not get the message within MarkBint, it becomes available again for reprocessing. If CKBP is not getting messages from the queue for some reason, reprocessing continues indefinitely. This is the default action of the bridge in this circumstance.

You can change the default action by specifying the ROUTEMEM=Y parameter on the bridge start data. This parameter causes messages to be routed to the Dead Letter Queue (DLQ) when their mark expires and they become visible for reprocessing. For more information about the ROUTEMEM bridge start parameter, see Administering the CICS-MQ bridge.

Chapter 3. Administering the CICS-MQ adapter

You can use the control functions of the CICS-MQ adapter to initiate, manage, and view the connection between CICS and IBM MQ. You can use CICS-MQ monitors to control the CICS-MQ trigger monitor, CKTI.

Before you begin

Before you can operate the CICS-MQ adapter, you must configure your CICS region appropriately and install an MQCONN resource definition.

When you install an MQCONN resource that specifies a value in the **INITQNAME** attribute, CICS also dynamically creates and installs an MQMONITOR resource with the reserved name of DFHMQINI representing the default initiation queue.

DFHMQINI automatically starts when the MQ connection is established.

Optionally, you can install more MQMONITOR resources to define attributes for IBM MQ message consumers, such as the trigger monitor transaction CKTI.

For detailed instructions, see Setting up the CICS-MQ adapter.

About this task

You can manage the CICS-MQ adapter using any of the following methods:

Using the control functions of the CICS-MQ adapter

You can access the control functions using the CKQC transaction, **EXEC CICS** SPI and CEMT commands, CICSPlex SM, or the CICS Explorer.

- You can use the CEMT transaction to issue commands in the CICS environment, or use the **EXEC CICS** system programming interface (SPI) commands in an application program.
- You can use the CKQC transaction through the CICS-MQ adapter control panels, or from the command line, or from an application program.
- If your CICS region is managed by CICSPlex SM, you can control the connection in several ways such as CICSPlex SM web user interface, using the EXEC CPSM application programming interface in an application program, or using the CMCI programming interface. These methods effectively use the CICS system programming interface (SPI) commands to update the available CICSPlex SM resource tables.
- The CICS Explorer includes SM Operations and SM Administration views for the CICS-MQ connection and MQ monitors, which provide the same functionality as the **EXEC CICS** SPI commands. For information about working with CICS resources using the CICS Explorer, see the Help information that is provided with the CICS Explorer.

Using CICS-MQ monitors to control instances of the CICS-MQ trigger monitor, CKTI

The MQMONITOR resource is the recommended method of controlling CKTI. The benefits are as follows:

- You can set up multiple instances of CKTI to monitor an MQ initiation queue.
- CKTI can be started automatically when the connection to the MQ queue manager is established.
- MQMONITOR allows the CKTI or a message consuming transaction to run under a preset user ID.

CAUTION: You can still use CKQC to start or stop instances of CKTI. However, if the region has installed MQ monitors and has multiple instances of CKTI running, using CKQC to stop CKTI could cause unpredictable results.

Summary of operator actions

The following table summarizes the operator actions that you can perform for the CICS-MQ connection, and whether you can perform these actions using **EXEC CICS** and CEMT commands, the CKQC transaction, the CICS Explorer, or CICSPlex SM.

Table 4. Operator actions for CICS-MQ connection

Operator action	EXEC CICS, CEMT	CKQC	CICS Explorer or CICSPlex SM
Start CICS-MQ connection	Yes, using **SET MQCONN**, but you cannot specify the default initiation queue name	Yes	Yes
Stop CICS-MQ connection	Yes, using **SET MQCONN**	Yes	Yes
Display connection status and settings	Yes, using **INQUIRE MQCONN**	Yes	Yes
Display connect and disconnect time	Yes, using CICS statistics commands	No	Yes
Display and reset detailed connection statistics including call types	Yes, using CICS statistics commands (resets all statistics)	Yes (resets CICS-MQ connection statistics only)	No
Display tasks that are using the CICS-MQ connection	Yes, but only the number of tasks, using **INQUIRE MQCONN**	Yes, full listing of tasks	No
Purge individual tasks that are using the CICS-MQ connection	Yes, using **SET TASK FORCEPURGE**	No	Yes
Enable or disable CICS-MQ API-crossing exit	No	Yes	No
Start instances of CKTI (CICS-MQ trigger monitor or task initiator)	No	Yes, but it is recommended to use MQMONITOR to control CKTI	No
Stop instances of CKTI	No	Yes, but it is recommended to use MQMONITOR to control CKTI	No
Start MQ monitors (MQMONITOR resources)	Yes	No	Yes
Stop MQ monitors (MQMONITOR resources)	Yes	No	Yes
Display and reset detailed MQMONITOR statistics including call types	Yes	No	Yes CICSPlex SM returns CICS statistics in the MQMONITOR record. You can reset statistics using the CICSRGN resource table.

Accessing the CICS-MQ adapter control panels

To access the adapter control panels, use the CICS transaction CKQC.

About this task

You can access the adapter control panels without starting the queue manager. You can also start a connection but it will not be active until the queue manager is started.

Procedure

1. Type CKQC and press Enter.

   The CICS-MQ adapter control initial panel is displayed.
2. In the menu bar, use the TAB key to move between the three options **Connection**, **CKTI**, and **Task**. Press Enter to select your choice.
3. Select an action from one of the menus by typing the number of your choice and then pressing Enter to confirm or function key F12 to cancel.
4. In the displayed panel or secondary window, type new values in the fields, as required.
5. Press function key F1 to get help on any panel or window.

CKQC commands from a command line

You can bypass the CICS-MQ adapter control panels by using the CKQC transaction directly from a command line and specifying command parameters.

For example, you can start the CICS-MQ connection from the CICS command line, using the default connection values, by entering the following command:

```
CKQC START
```

You can also issue CKQC transaction commands from the console using z/OS commands. These commands take the following form:

```
MODIFY CICS-job-name CKQC command-line-command
```

The CKQC transaction commands are as follows:

CKQC START
Starts the connection to IBM MQ. For the syntax of this command and examples, see “Starting a CICS-MQ connection from the CICS command line” on page 37.

CKQC STOP
Stops the connection to IBM MQ. For the syntax of this command and examples, see “Stopping a CICS-MQ connection from the CICS command line” on page 43.

CKQC DISPLAY
Returns essential information about the connection to IBM MQ, in the form of a CICS message. This command has no options. For an example, see “Displaying CICS-MQ connection status and settings” on page 46.

CKQC STARTCKTI
Starts an instance of the CICS-MQ trigger monitor, CKTI. For the syntax of this command and examples, see “Starting CKTI from the CICS command line” on page 61.

CKQC STOPCKTI
Stops an instance of CKTI. For the syntax of this command and examples, see “Stopping an instance of CKTI from a terminal” on page 63.

CKQC MODIFY
Resets CICS-MQ connection statistics, and enables or disables the CICS-MQ API-crossing exit. For examples of this command, see Enabling the CICS-WebSphere MQ API-crossing exit, Disabling the

CICS-WebSphere MQ API-crossing exit, and “Resetting CICS-MQ connection statistics” on page 56. The full syntax of the command is as follows:

```
CKQC MODIFY Y|N E|D
```

where:

Y|N
Specify one of:

Y
Reset connection statistics.

N
Do not reset connection statistics.

This parameter is required.

E|D
Specify one of:

E
Enable the CICS-MQ API-crossing exit.

D
Disable the CICS-MQ API-crossing exit.

CKQC commands from CICS application programs

You can use the **EXEC CICS LINK** command to access most of the CICS-MQ adapter control functions from CICS application programs.

To carry out an action such as starting the connection, your application program must link to the appropriate CICS-MQ adapter program and issue a CKQC command. Table 5 on page 32 lists these programs and their functions.

Table 5. CICS-MQ adapter programs

CICS-MQ adapter program	Alternative name accepted for compatibility	Function	CKQC command
DFHMQQCN	CSQCQCON	Start the connection to IBM MQ	CKQC START
DFHMQDSC	CSQCDSC	Stop the connection	CKQC STOP
DFHMQRS	CSQCRST	Modify the connection (reset statistics or enable API-crossing exit)	CKQC MODIFY
DFHMQSSQ	CSQCSSQ	Start or stop an instance of CKTI	CKQC STARTCKTI or CKQC STOPCKTI
DFHMQDSL	CSQCDSPL	Display the status of the connection	CKQC DISPLAY

Command syntax in application programs

Some commands issued by the application program must be padded with trailing spaces to make the length of the command equal ten characters.

When an argument follows the command, an extra space character must be added as a separator. The commands affected by this restriction and the number of trailing spaces required for each command are as follows:

Command	Number of trailing spaces (not including the separator)
START	5
MODIFY	4
STARTCKTI	1
STOPCKTI	2

With all other commands the padding is optional.

This example illustrates how to pad adapter commands. The MODIFY command must be padded with 4 trailing spaces plus another space as a separator. Starting at the M in MODIFY, the argument Y is the twelfth character.

```
EXEC CICS LINK PROGRAM('DFHMQRS ')
          INPUTMSG('CKQC MODIFY     Y')
                         ↑          ↑
                         ............
                         1          12
```

Passing parameters from a CICS application program

Use the following rules to determine how to pass parameters between CICS and IBM MQ:

- The CICS transaction must be running on an attached terminal. If it is not, all IBM MQ commands are ignored.
- If a CICS application program on an attached terminal is connected to IBM MQ, you must use the INPUTMSG option with **EXEC CICS LINK** to pass parameters, except at PLTPI time.
- If you connect to IBM MQ at PLTPI time, you must use the COMMAREA option to pass parameters. If you use the INPUTMSG option, the command is ignored. However, the adapter STOP commands, CKQC STOP and CKQC STOP FORCE, cannot be run at PLTPI time, regardless of whether you use the INPUTMSG option or the COMMAREA option.

EXEC CICS LINK interface messages

When you call the adapter operation functions START and STOP from an application program using **EXEC CICS LINK**, the resulting messages are written to both the system console and a transient data queue (TDQ) named CKQQ.

When the application program returns from the LINK, it can read back the messages by repeating **EXEC CICS READQ TD** QUEUE(CKQQ) until the queue is empty. The following restrictions apply:

- The TDQ queue name is CKQQ and cannot be changed. The definition for CKQQ is supplied in the CICS CSD group DFHDCTG.
- The queue is not cleared before it is written to.
- If you have more than one application writing to the TDQ, the messages are not serialized. The calling programs must serialize themselves.
- The same set of messages is also displayed on the system console.

EXEC CICS and CEMT commands for the CICS-MQ connection and monitors

You can use **EXEC CICS** system programming interface (SPI) commands and CEMT commands to manage MQCONN and MQMONITOR resource definitions, start and stop the CICS-MQ connection and MQ

monitors, display information and statistics for the connection and MQ monitors, and purge tasks that are using the connection.

You can issue the **EXEC CICS** SPI commands in a CICS application program. You can issue CEMT commands directly through the CICS CEMT transaction, without using an application program.

The following table shows the **EXEC CICS** SPI and CEMT commands and their functions.

Table 6. SPI and CEMT commands for the CICS-MQ connection and monitors

EXEC CICS command	**CEMT command**	**Function**
CREATE MQCONN	Not available	Set up an MQCONN resource definition
DISCARD MQCONN	CEMT DISCARD MQCONN	Discard an MQCONN resource definition **Note:** When you discard an MQCONN resource that includes a setting for the INITQNAME attribute, the dynamically created MQMONITOR resource and any user-defined MQMONITOR resources are also discarded.
INQUIRE MQCONN	CEMT INQUIRE MQCONN	Inquire on MQCONN resource definition attributes, status of connection, and number of tasks using connection
SET MQCONN	CEMT SET MQCONN	Start and stop connection, and change default queue manager or queue-sharing group name and resynchronization strategy
CREATE MQMONITOR	Not available	Set up an MQMONITOR resource definition
DISCARD MQMONITOR	CEMT DISCARD MQMONITOR	Discard an MQMONITOR resource definition
INQUIRE MQMONITOR	CEMT INQUIRE MQMONITOR	Inquire on MQMONITOR resource definition attributes and the current status of the monitor (for example, MONSTATUS(STARTED) with a task ID that is not zero)
SET MQMONITOR	CEMT SET MQMONITOR	Start and stop MQMONITOR
INQUIRE SYSTEM	CEMT INQUIRE SYSTEM	Find name of installed MQCONN resource definition
SET TASK FORCEPURGE	CEMT SET TASK FORCEPURGE	Purge tasks that are using the CICS-MQ connection
EXTRACT STATISTICS MQCONN	not available	View statistics online for the CICS-MQ connection
SET STATISTICS	CEMT SET STATISTICS	Reset all statistics including those for the CICS-MQ connection

CICSPlex SM views for the CICS-MQ connection and monitors

You can use CICSPlex SM views to create and modify MQCONN and MQMONITOR resource definitions, start and stop the CICS-MQ connection, and display information and statistics for the connection and monitors.

The following viewsets are available in CICSPlex SM for the CICS-MQ connection and monitors:

CICS operations > DB2, DBCTL and WebSphere MQ operations views > WebSphere MQ Connection
The views in the MQCON viewset display information about the connection, including connection status and connect and disconnect times. The main views have action buttons that you can use to start and stop the connection, and you can change the settings for the connection.

CICS operations > DB2, DBCTL and WebSphere MQ operations views > WebSphere MQ initiation queue
The views in the MQINI viewset display the attributes of the installed MQINI resource definition for the CICS system, including the resource signature. You cannot carry out any actions on this resource definition.

CICS operations > DB2, DBCTL and WebSphere MQ operations views > WebSphere MQ Connection Statistics
The views in the MQCONN viewset display information and statistics about the connection. The MQCONN viewset is the viewset that existed before CICS resource definitions were introduced for the CICS-MQ connection, so these views do not have action buttons to start and stop the connection, and you cannot change the settings for the connection in these views.

CICS operations > DB2, DBCTL and WebSphere MQ operations views > WebSphere MQ monitors
The views in the MQMON viewset display status information and statistics for MQ monitors defined in the CICS region.

Administration > Basic CICS resource administration > Resource definitions > WebSphere MQ connection definitions
The views in the MQCONDEF viewset display the attributes of MQCONN resource definitions, which are represented in CICSPlex SM by MQCONDEF BAS objects. You can use these views to create, change, discard, or install an MQCONN resource definition.

Administration > Basic CICS resource administration > Resource definitions in resource group > MQCONN resources in a resource group
The MQCINGRP viewset shows information about the membership of an MQCONN resource definition (MQCONDEF) in a resource group (RESGROUP).

Administration > Basic CICS resource administration > Resource definitions in resource group > MQMON resources in a resource group
The MQMINGRP viewset shows information about the membership of an MQMONITOR resource definition (MQMONDEF) in a resource group (RESGROUP).

Starting a CICS-MQ connection

If the CICS-MQ connection is not started at CICS initialization, you can start the connection in several ways.

Before you begin

Before you can start the CICS-MQ connection, you must configure your CICS region appropriately and install an MQCONN resource definition. Setting up the CICS-MQ adapter tells you how to do this.

About this task

To set the CICS-MQ connection to start at CICS initialization in the future, specify the CICS system initialization parameter **MQCONN=YES**, or include the DFHMQCOD program or your own equivalent program in your CICS PLTPI list. For instructions to do this, see Setting up the CICS-MQ adapter.

Procedure

You can use any of the following methods to start a CICS-MQ connection:

- Using the CICS-MQ adapter control panels

  You can run the CICS-MQ adapter control transaction, CKQC, and then start the CICS-MQ connection from the CICS-MQ adapter control panel.
- Issuing **CKQC START** from the CICS command line

  You can start the CICS-MQ connection by issuing the **CKQC START** command from a command line.
- Using a CICS application program that issues **EXEC CICS SET MQCONN**

  You can start the connection to IBM MQ by issuing an **EXEC CICS SET MQCONN** command in a CICS application program.
- Using a CICS application program with linking to DFHMQQCN

  You can start a connection to IBM MQ by linking to the CICS-MQ adapter connect program, DFHMQQCN (or CSQCQCON, which is a previous name for the program), from a CICS application program. Although this method of starting the connection is maintained for compatibility, using the **EXEC CICS SET MQCONN** command is the preferred method of starting the connection from a CICS application program.
- Using the CICS CEMT transaction

  You can start the connection to IBM MQ by issuing the **SET MQCONN** command in the CICS CEMT transaction.
- Using the CICSPlex SM web user interface

  You can start the connection to IBM MQ from the **WebSphere MQ Connection** view in the CICSPlex SM web user interface (WUI).
- Using the CICS Explorer

  You can start the CICS-MQ connection from the CICS Explorer. The CICS Explorer provides a functional equivalent to the CICSPlex SM web user interface and the CEMT transaction.

Results

When the connection between CICS and MQ has been established, CICS will start MQ monitors that have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED) if the user ID associated with the task that set the MQCONN resource to CONNECTED has sufficient authority to start the associated transactions.

Starting a CICS-MQ connection from the CICS-MQ adapter control panels

You can run the CICS-MQ adapter control transaction, CKQC, and then start the CICS-MQ connection from the CICS-MQ adapter control panel.

Procedure

1. In the CICS-MQ adapter control initial panel, select **Connection** from the menu bar.
2. Select the **Start** action from the menu.
3. The connection values displayed in the **Start a Connection** secondary parameter window are taken from the installed MQCONN and MQINI resource definitions for the CICS region. The queue manager name can be either a single queue manager or a queue-sharing group. Overtype these settings if you want to change them.

   If you change the settings, the values for the corresponding attributes (MQNAME and INITQNAME) in the installed MQCONN and MQINI resource definitions are replaced with the settings that you specify. To revert to the original settings, reinstall the resource definitions.
4. Press Enter to confirm.

Results

Messages indicating the success or failure of the attempt to start the connection are displayed on the adapter messages panel, CKQCM1.

Example

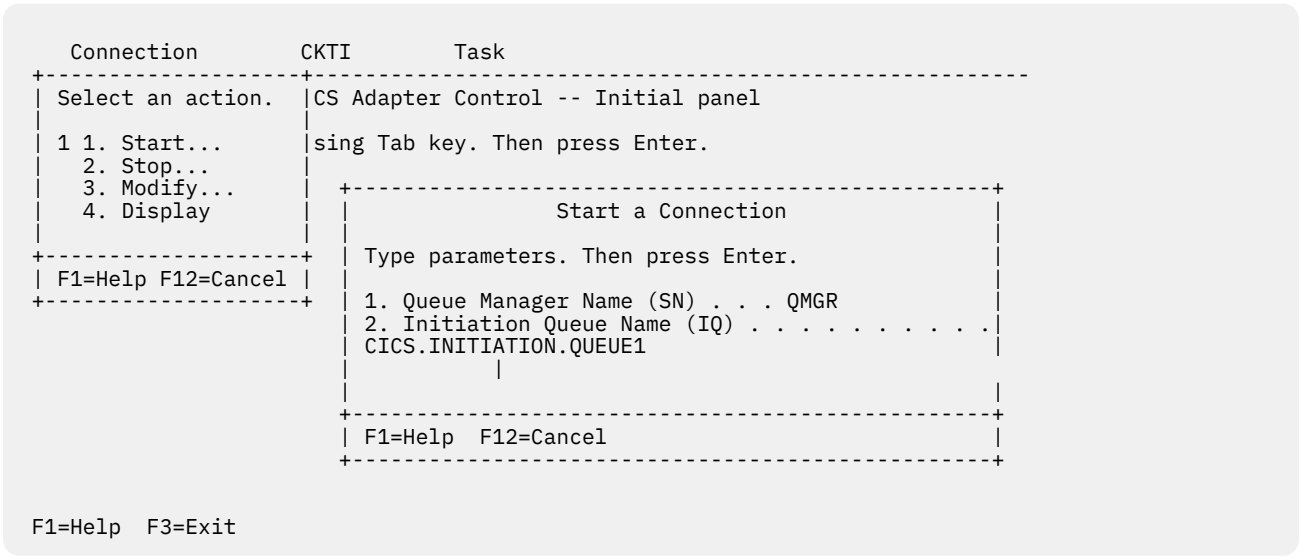


Figure 4. Starting a connection

Starting a CICS-MQ connection from the CICS command line

You can start the CICS-MQ connection by issuing the **CKQC START** command from a command line.

Before you begin

By default, CICS folds lowercase input, for both keywords and parameters, to uppercase input. Therefore, by default, these commands are equivalent:

```
CKQC START Y CSQ1     CICS01.INITQ
ckqc start y csq1     cics01.initq
```

If you want to use lowercase IBM MQ queue names:

1. Specify UCTRAN(TRANID) on the TYPETERM definition of terminals that start adapter control functions.
2. Specify UCTRAN(NO) on the transaction profile used by all “CKxx” transactions.

Thereafter, the adapter translates all lowercase arguments, *except queue names*, to uppercase arguments.

Procedure

- To start a connection using the default connection values for the queue manager name and the default initiation queue name, issue the command CKQC START with no parameters.

  The transaction uses the default connection values from the installed MQCONN and MQMONITOR resource definitions for the CICS region.
- To start a connection using connection values that you define explicitly, enter the following command:

  ```
  CKQC START Y|N <subsystem ID> <filler> <initiation queue name>
  ```

  where:

  Y|N
  Specify either:

Y
Use the default connection values; that is, if there are any blank arguments, substitute the default values taken from the installed MQCONN and MQMONITOR resource definitions.

N
Do not use the default connection values.

<subsystem ID>
Name of the IBM MQ queue manager to which CICS connects. The queue manager name can be either a single queue manager or a queue-sharing group.

<filler>
Three-character filler that takes the place of an obsolete 3-digit trace number for compatibility purposes. The filler is required because the parameters are positional.

<initiation queue name>
The name of the default initiation queue for the connection.

The parameters are positional; you must enter every field to its maximum length if you want to override the default.

When you specify connection values in this way, the INITQNAME value of the installed MQCONN resource definition and the QNAME value of the installed MQMONITOR resource definition are replaced with the settings that you specify. If you want to revert to the original settings, reinstall the resource definitions.

Starting a CICS-MQ connection by using EXEC CICS SET MQCONN from a CICS application program

You can start the connection to IBM MQ by issuing an **EXEC CICS SET MQCONN** command in a CICS application program.

About this task

The **SET MQCONN CONNECTED** command starts all installed MQMONITOR resources that have the attributes AUTOSTART(YES) and STATUS(ENABLED), including the MQMONITOR resource DFHMQINI that is dynamically created when the MQCONN attribute INITQNAME contains valid data.

Procedure

- To start the connection from a CICS application program using the default connection values, issue the following command:

```
EXEC CICS SET MQCONN CONNECTED
```

  This command starts a connection using the default connection value for the queue manager name. This value, which could be the name of a single queue manager or the name of a queue-sharing group, is taken from the installed MQCONN resource definition for the CICS region.

- To specify your own connection values when you start the connection from a CICS application program, issue a command like this example:

```
EXEC CICS SET MQCONN CONNECTED
                 MQNAME(qqqq)
                 RESYNC
```

  where *qqqq* is the 1 - 4 character name of an IBM MQ queue manager or queue-sharing group.

  Specify either RESYNC or NORESYNC to select the resynchronization behavior if the connection is lost. For more information about resynchronization, see Automatic reconnection and resynchronization.

Results

If the requested queue manager is active, control returns when CICS and IBM MQ are connected. If the requested queue manager is not active, CICS returns a NORMAL response with RESP2=8, indicating that

the CICS-MQ adapter is in connecting state and will connect to IBM MQ as soon as the requested queue manager becomes active.

What to do next
If you specify your own value for MQNAME, the queue manager name or queue-sharing group that you specified in the MQNAME attribute of the installed MQCONN resource definition is replaced with the name that you specified on this command. To revert to the original queue manager or queue-sharing group, set MQNAME again.

Starting a CICS-MQ connection by linking to DFHMQQCN from a CICS application program

You can start a connection to IBM MQ by linking to the CICS-MQ adapter connect program, DFHMQQCN (or CSQCQCON, which is a previous name for the program), from a CICS application program. Although this method of starting the connection is maintained for compatibility, using the **EXEC CICS SET MQCONN** command is the preferred method of starting the connection from a CICS application program.

About this task
Your CICS application program can be written in C, COBOL, PL/I, or assembler language.

Procedure

1. Your program must pass a parameter list that specifies the connection values to be used.

   The parameter list is as follows:

   CKQC
   Four-character transaction ID; must be `'CKQC'`.

   DISPMODE
   One-byte field; must contain a blank.

   CONNREQ
   Ten-character field; must contain `'START  '`.

   DELIM1
   One-byte delimiter field; must contain a blank.

   MQDEF
   One-character field that specifies whether this connection is to use the default connection values from the installed MQCONN and MQMONITOR resource definitions for the CICS region. Before CICS TS 4.1, this field was known as INITP. The possible values are:

   Y
   Use the values for the queue manager name and default initiation queue from the installed MQCONN and MQMONITOR resource definitions for the CICS region.

   N
   Do not use the default values. If you specify 'N', you must supply a queue manager name in the CONNSSN field, and you may optionally specify a default initiation queue name in the CONNIQ field. When you supply these values, the INITQNAME value of the installed MQCONN resource definition and the QNAME value of the installed MQMONITOR resource definition are replaced with the settings that you specify. To revert to the original settings, reinstall the resource definitions.

   ' '
   Equivalent to 'Y'.

   DELIM2
   One-byte delimiter field; must contain a blank.

   CONNSSN
   Four-character field used to specify the z/OS subsystem name of the target IBM MQ queue manager. You can specify either a single queue manager or a queue-sharing group.

DELIM3
Five-byte delimiter field; must contain blanks. Previous releases supported a trace point override. This is now ignored.

CONNIQ
48-character field that specifies the name of the default initiation queue.

2. Your program must issue a command like this to link to the adapter connect program, DFHMQQCN:

```
EXEC CICS LINK PROGRAM('DFHMQQCN')
          COMMAREA(CONNPL) LENGTH(length of CONNPL)
```

In this example, the name of the parameter list is CONNPL.

Results
Output messages from DFHMQQCN are displayed on the system console.

Starting a CICS-MQ connection through the CICS CEMT transaction

You can start the connection to IBM MQ by issuing the **SET MQCONN** command in the CICS CEMT transaction.

About this task

The **SET MQCONN CONNECTED** command starts all installed MQMONITOR resources that have the attributes AUTOSTART(YES) and STATUS(ENABLED), including the MQMONITOR resource DFHMQINI that is dynamically created when the MQCONN attribute INITQNAME contains valid data.

For details of how to start and use the CEMT transaction, see CEMT - master terminal.

Procedure

1. On the CICS command line, enter the command `CEMT SET MQCONN`.
   The status of the connection and the default connection values are displayed. The values are taken from the installed MQCONN resource definition for the CICS region.
2. Optional: If you want to change the name of the IBM MQ queue manager or queue-sharing group for the connection, overtype the value in the **Mqname** field with a different name.
   When you change this value, the queue manager name or queue-sharing group that you specified in the MQNAME attribute of the installed MQCONN resource definition is replaced with the name that you specified on this command. To revert to the original queue manager or queue-sharing group, change the value again.
3. Optional: If you want to change the resynchronization behavior, overtype the **Resyncmember** field with either `Resync`, `Noresync` or `Groupresync`.
   For more information about resynchronization, see Automatic reconnection and resynchronization.
4. To start the connection, overtype the value `Notconnected` with the value `Connected` and press Enter.
   CICS starts the connection, by using any new settings that you specified.

Starting a CICS-MQ connection through the CICSPlex SM web user interface

You can start the connection to IBM MQ from the **WebSphere MQ Connection** view in the CICSPlex SM web user interface (WUI).

About this task

In the CICS Explorer, the MQ Connections (MQCON) view provides a functional equivalent to this view in the WUI.

Procedure

1. From the CICSPlex SM WUI main menu, select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ Connection**.

   For a full listing and explanation of the information in this viewset, see WebSphere MQ connections - MQCON .
2. Click the name of the CICS system for which you want to start the connection.

   CICSPlex SM displays the **WebSphere MQ Connection - General Information** view.

   The view shows the current status of the connection and the default connection values, which are taken from the installed MQCONN resource definition for the CICS system.
3. Optional: If you want to change the name of the IBM MQ queue manager or queue-sharing group for the connection, enter a different name in the **MQ queue manager or QSG name** field.

   When you change this value, the queue manager name or queue-sharing group that you specified in the MQNAME attribute of the installed MQCONN resource definition is replaced with the name that you specified on this command. To revert to the original queue manager or queue-sharing group, change the value again.
4. Optional: If you want to change the resynchronization behavior, select the alternative option from the **Resynchronization member** field.

   For more information about resynchronization, see Automatic reconnection and resynchronization.
5. If you made any changes to the settings for the connection, click the **Apply changes** button.
6. To start the connection, click the **Start the CICS-MQ connection** button.

   The CICS system starts the connection using the settings that you specified.

Stopping a CICS-MQ connection

You can stop the CICS-MQ connection in several ways. When you stop the connection, you can choose whether to carry out a quiesced shutdown or a forced shutdown.

About this task

A quiesced shutdown lets each CICS transaction end before the interface is closed. A forced shutdown abnormally ends CICS transactions connected to the queue manager. For an explanation of the actions that the CICS-MQ adapter takes for a quiesced shutdown and a forced shutdown, see What happens when the CICS-WebSphere MQ connection shuts down.

You must stop the connection before you install a changed MQCONN resource definition for the CICS region.

When you restart the connection after a forced shutdown, there might be indoubt units of work. For information about indoubt units of work, see Automatic reconnection and resynchronization. For instructions about resolving indoubt units of work manually if necessary, see What happens when the CICS-WebSphere MQ adapter restarts.

Procedure

You can use any of the following methods to stop a CICS-MQ connection:

- Using the CICS-MQ adapter control panels

  You can run the CICS-MQ adapter control transaction, CKQC, and then use the CICS-MQ adapter control panel to stop the CICS-MQ connection.
- Issuing **CKQC STOP** from the CICS command line

  You can stop the CICS-MQ connection by issuing the **CKQC STOP** command from a command line.
- Using a CICS application program that issues **EXEC CICS SET MQCONN**

You can stop the connection to IBM MQ by issuing an **EXEC CICS SET MQCONN** command in a CICS application program.

- Using a CICS application program with linking to DFHMQDSC

  You can stop a connection to IBM MQ by linking to the CICS-MQ adapter program DFHMQDSC (or CSQCDSC, which is a previous name for the program), from a CICS application program. Although this method of stopping the connection is maintained for compatibility, using the **EXEC CICS SET MQCONN** command is the preferred method of stopping the connection from a CICS application program.

- Using the CICS CEMT transaction

  You can stop the connection to IBM MQ by issuing the **SET MQCONN** command in the CICS CEMT transaction.

- Using the CICSPlex SM web user interface

  You can stop the connection to IBM MQ from the **WebSphere MQ Connection** view in the CICSPlex SM web user interface (WUI).

- Using the CICS Explorer

  You can stop the CICS-MQ connection from the CICS Explorer. The CICS Explorer provides a functional equivalent to the CICSPlex SM web user interface and the CEMT transaction.

Results

If you are using MQ monitors, when the MQ connection is stopped, MQ monitors are stopped automatically.

Stopping a CICS-MQ connection from the CICS-MQ adapter control panels

You can run the CICS-MQ adapter control transaction, CKQC, and then use the CICS-MQ adapter control panel to stop the CICS-MQ connection.

Procedure

1. From the CICS-MQ adapter control initial panel, select **Connection** from the menu bar.
2. Select the **Stop** action from the menu.
3. Use the **Stop Connection** secondary parameter window to select the type of shutdown that you require, either quiesced or forced.

Results

The messages associated with stopping a connection are displayed on the system console.

Example

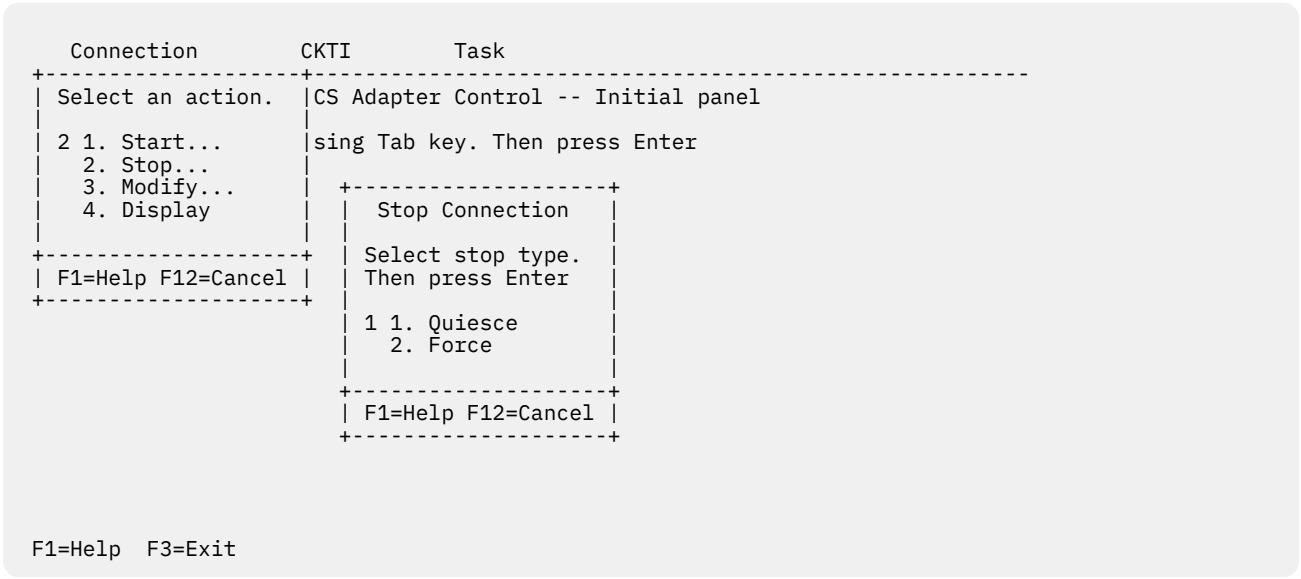


Figure 5. Stopping a connection from the CKQC initial panel

Stopping a CICS-MQ connection from the CICS command line

You can stop the CICS-MQ connection by issuing the **CKQC STOP** command from a command line.

Procedure

- To initiate a quiesced shutdown, issue the command CKQC STOP.
  The connection shuts down only after the last task has completed its work.
- To initiate a forced shutdown, issue the command CKQC STOP FORCE.
  The connection shuts down immediately, regardless of the state of any inflight tasks.

Stopping a CICS-MQ connection by using EXEC CICS SET MQCONN from a CICS application program

You can stop the connection to IBM MQ by issuing an **EXEC CICS SET MQCONN** command in a CICS application program.

Procedure

- To carry out a quiesced shutdown where control is not returned to the application until the connection is stopped, issue the following command:

  ```
  EXEC CICS SET MQCONN NOTCONNECTED
  ```

  The BUSY(WAIT) option, which makes the request synchronous, is the default.
- To carry out a quiesced shutdown where control is returned to the application before the connection is stopped (an asynchronous request), issue the following command:

  ```
  EXEC CICS SET MQCONN NOTCONNECTED NOWAIT
  ```

- To carry out a forced shutdown, issue the following command:

  ```
  EXEC CICS SET MQCONN NOTCONNECTED FORCE
  ```

  With a forced shutdown, control is not returned to the application until the connection is stopped.

Stopping a CICS-MQ connection by linking to DFHMQDSC from a CICS application program

You can stop a connection to IBM MQ by linking to the CICS-MQ adapter program DFHMQDSC (or CSQCDSC, which is a previous name for the program), from a CICS application program. Although this method of stopping the connection is maintained for compatibility, using the **EXEC CICS SET MQCONN** command is the preferred method of stopping the connection from a CICS application program.

About this task

When you issue an **EXEC CICS LINK** command to link to DFHMQDSC, the program requires a terminal associated task.

Procedure

- To stop a connection from a CICS application program with a quiesced shutdown, issue this command:

```
EXEC CICS LINK PROGRAM('DFHMQDSC ')
          INPUTMSG('CKQC STOP')
```

  The QUIESCE parameter is the default.
- To stop a connection from a CICS application program with a forced shutdown, issue this command:

```
EXEC CICS LINK PROGRAM('DFHMQDSC ')
          INPUTMSG('CKQC STOP FORCE')
```

Results

Output messages from DFHMQDSC are displayed on the system console.

Stopping a CICS-MQ connection through the CICS CEMT transaction

You can stop the connection to IBM MQ by issuing the **SET MQCONN** command in the CICS CEMT transaction.

About this task

For details of how to start and use the CEMT transaction, see CEMT - master terminal.

Procedure

1. On the CICS command line, enter the command `CEMT SET MQCONN`.

   You obtain a display that lists the current status of the connection and the default connection values, which are taken from the installed MQCONN resource definition for the CICS region.
2. To stop the connection, overtype the value `Connected` to select the type of shutdown as follows:
   a) If you want a quiesced shutdown, type `Notconnected`.
   b) If you want a forced shutdown, type `Forcenotcon`.
3. Press Enter.

   CICS stops the connection.

Stopping a CICS-MQ connection through the CICSPlex SM web user interface

You can stop the connection to IBM MQ from the **WebSphere MQ Connection** view in the CICSPlex SM web user interface (WUI).

About this task

In the CICS Explorer, the MQ Connections (MQCON) view provides a functional equivalent to this view in the WUI.

Procedure

1. From the CICSPlex SM WUI main menu, select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ Connection**.

   For a full listing and explanation of the information in this viewset, see WebSphere MQ connections - MQCON .
2. Click the name of the CICS system for which you want to stop the connection.

   CICSPlex SM displays the **WebSphere MQ Connection - General Information** view.
3. To stop the connection, click the **Stop the CICS-MQ connection** button.

   CICSPlex SM displays the **Stop the CICS-MQ connection** secondary panel.
4. Use the **Busy value** box to choose the type of disconnection you want, as follows:

   - To carry out a quiesced shutdown where CICSPlex SM pauses until the connection is stopped (a synchronous request), choose **WAIT**.
   - To carry out a quiesced shutdown where CICSPlex SM issues the shutdown request and then continues working whether or not the connection has been stopped (an asynchronous request), choose **NOWAIT**.
   - To carry out a forced shutdown, choose **FORCE**.
5. Under **Perform 'Disconnect'?**, click the **Yes** button.

   The selected CICS system stops the connection.

Displaying information about CICS-MQ connections

You can obtain basic information or detailed statistics about the CICS-MQ connection, depending on the information that you require and the environment in which you are currently working.

You can use the CEMT or **EXEC CICS INQUIRE** commands, the CICS-MQ adapter control panels, the CKQC DISPLAY command, or CICSPlex SM to display information about the connection. Each method produces a set of information selected for that environment. The following table shows the items of information that are available with each method:

Table 7. Displaying information about a CICS-MQ connection

Information	INQUIRE MQCONN using CEMT or in application	CICS-MQ adapter control panels	CKQC DISPLAY from command line or in application	CICSPlex SM views
Status of connection (for example, Connected)	Yes	Yes	Yes	Yes
Name of queue manager or queue-sharing group specified for connection	Yes	Yes	Yes	Yes
Name of queue manager actually connected	Yes	Yes	Yes	Yes
Name of default initiation queue	Use **INQUIRE MQMONITOR**	Yes	No	Yes
Number of inflight tasks using connection	Yes	Yes	Yes	Yes
Number of trigger monitor (CKTI) tasks using connection	Yes	Yes	No	Yes

Table 7. Displaying information about a CICS-MQ connection (continued)				
Information	**INQUIRE MQCONN using CEMT or in application**	**CICS-MQ adapter control panels**	**CKQC DISPLAY from command line or in application**	**CICSPlex SM views**
Details of tasks using connection, such as transaction ID and task number	No	Yes	No	No
Status of CICS-MQ API-crossing exit	No	Yes	No	No
Resynchronization behavior	Yes	No	No	Yes
IBM MQ release number	Yes	No	No	Yes
Resource signature for MQCONN resource definition	Yes	No	No	Yes
Name of installed MQCONN resource definition	Use INQUIRE SYSTEM	No	No	Yes
Connect and disconnect times	Use EXTRACT STATISTICS (**EXEC CICS** only)	No	No	Yes
Detailed statistics such as numbers of MQI calls	Use EXTRACT STATISTICS (**EXEC CICS** only)	Yes	No	Yes

Displaying CICS-MQ connection status and settings

To find basic information such as the status of the CICS-MQ connection and the name of the queue manager or queue-sharing group for the connection, you can use the CEMT or **EXEC CICS INQUIRE** commands, the CICS-MQ adapter control panels, the **CKQC DISPLAY** command, or CICSPlex SM.

About this task

Choose the most appropriate method to obtain basic information about the CICS-MQ connection, depending on the information that you require and your current environment. For a listing of the information that you can obtain by each method, see "Displaying information about CICS-MQ connections" on page 45.

Procedure

- To display connection status and settings from the CICS command line, enter one of the following commands:
  - **CEMT INQUIRE MQCONN**

    This command starts the CEMT transaction and displays the connection information on the screen. Use the **CEMT INQUIRE MQMONITOR** command in the same way to find the name of the default initiation queue, or the **CEMT INQUIRE SYSTEM** command to find the name of the installed MQCONN resource definition.
  - **CKQC DISPLAY**

This command produces only the most essential information about the connection. When you issue this command, CICS writes message DFHMQ0453I to the transient data queue CKQQ with the connection information, as follows:

```
DFHMQ0453I date time applid
Status of connection to qmgr-name is {Connecting |
Pending |Connected | Quiescing | Stopping-Force | Disconnected |
Inactive |Unknown}. number tasks are in flight.
```

- To obtain connection status and settings in a CICS application program, choose one of the following methods:
  - Issue an **EXEC CICS INQUIRE MQCONN** command. This command returns information to the application program. Use the **INQUIRE MQMONITOR** command in the same way to find the name of the default initiation queue, or the **INQUIRE SYSTEM** command to find the name of the installed MQCONN resource definition.
  - Issue an **EXEC CICS LINK** command to link to the CICS-MQ adapter program DFHMQDSL (or CSQCDSPL, which is accepted for compatibility), and issue the **CKQC DISPLAY** command, as in this example:

    ```
    EXEC CICS LINK PROGRAM('DFHMQDSL') INPUTMSG('CKQC DISPLAY')
    ```

    You can use the COMMAREA option instead of INPUTMSG, but only when the program is run at PLT time. CICS writes message DFHMQ0453I to the transient data queue CKQQ, as it does when you issue the **CKQC DISPLAY** command from the command line.
- To view connection status and settings in the CICS-MQ adapter control panels, select **Connection** from the menu bar on the initial panel, and then select the **Display** action from the menu.
  The display connection panel, shown here, includes the connection information. Press function key F1 to see an explanation of specific fields in this panel, and press Enter to refresh the panel.

```
CKQCM2                    Display Connection panel

Read connection information. Then press F12 to cancel.

  CICS Applid =  VICIC14   Connection Status = Connected       QMgr name = MQDD
  Mqname =       MQDD      Tracing           = On              API Exit = Off
  Initiation Queue Name = VICIC14.INITIATION.QUEUE
-------------------------------- STATISTICS ---------------------------------
Number of in-flight tasks =   1           Total API calls =             43912
Number of running CKTI    =   1
         APIs and flows analysis              Syncpoint          Recovery
---------------------------------------  -------------------  ---------------
Run OK        43874  MQINQ          6806  Tasks            26  Indoubt       0
Futile            0  MQSET             0  Backout           0  UnResol       0
MQOPEN         6833  ------ Flows ------  Commit           10  Commit        0
MQCLOSE        6823  Calls         43952   S-Phase         10  Backout       0
MQGET         10032  SyncComp      43922   2-Phase          0
 GETWAIT       3399  SuspReqd          0
MQPUT         13399  Msg Wait          7
MQPUT1            5  Switched      43940



F1=Help  F12=Cancel  Enter=Refresh
```

- To display connection information in CICSPlex SM, from the main menu select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ connections**.

  For a listing and explanation of the information in this viewset, see WebSphere MQ connections - MQCON .

  To see the name of the default initiation queue, select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ initiation queue**.

Displaying CICS-MQ connect and disconnect time

You can use CICSPlex SM or the CICS statistics to display connect and disconnect times for the CICS-MQ connection. The CKQC transaction does not provide connect and disconnect times.

Procedure

1. To display the connect and disconnect times in CICS, issue the command **EXEC CICS EXTRACT STATISTICS MQCONN** from a CICS application program.

   CICS returns the statistics to the application.
2. To display the connect and disconnect times in CICSPlex SM, from the main menu select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ connections**.

   For a listing and explanation of the information in this viewset, see WebSphere MQ connections - MQCON .

Displaying CICS-MQ connection statistics and call types

To display detailed statistics about the CICS-MQ connection, such as the number of each type of MQI call made using the connection and the number and state of any indoubt units of work, use the CICS statistics, the CICS-MQ adapter control panels, or CICSPlex SM.

Procedure

1. To display connection statistics in CICS, issue the command **EXEC CICS EXTRACT STATISTICS MQCONN** from a CICS application program.

   CICS returns the statistics to the application.

   For a listing and explanation of these statistics, see CICS-MQ connection statistics.
2. To display connection statistics in the CICS-MQ adapter control panels, select **Connection** from the menu bar on the initial panel, then select the **Display** action from the menu.
   The display connection panel, shown here, includes the connection statistics.

```
CKQCM2                    Display Connection panel

Read connection information. Then press F12 to cancel.

  CICS Applid =  VICIC14   Connection Status = Connected       QMgr name = MQDD
  Mqname =       MQDD      Tracing           = On              API Exit = Off
  Initiation Queue Name = VICIC14.INITIATION.QUEUE
--------------------------------- STATISTICS ---------------------------------
Number of in-flight tasks =   1           Total API calls =             43912
Number of running CKTI    =   1
         APIs and flows analysis              Syncpoint          Recovery
-----------------------------------------  ------------------  ---------------
Run OK        43874  MQINQ          6806  Tasks            26  Indoubt       0
Futile            0  MQSET             0  Backout           0  UnResol       0
MQOPEN         6833  ------ Flows ------  Commit           10  Commit        0
MQCLOSE        6823  Calls         43952   S-Phase         10  Backout       0
MQGET         10032  SyncComp      43922   2-Phase          0
 GETWAIT       3399  SuspReqd          0
MQPUT         13399  Msg Wait          7
MQPUT1            5  Switched      43940



F1=Help  F12=Cancel  Enter=Refresh
```

   The statistics displayed for inflight tasks, instances of CKTI, and API calls, are totals for the current connection, since statistics were last reset. The statistics displayed under the headings `APIs and flows analysis`, `Syncpoint`, and `Recovery`, are statistics produced by the adapter. Press function key F1 to see an explanation of specific fields in this panel, and press Enter to refresh the panel.
3. To display connection statistics in CICSPlex SM, from the main menu select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ connection statistics**.

For a listing and explanation of the information in this viewset, see WebSphere MQ connection statistics - MQCONN.

Displaying tasks that are using the CICS-MQ connection

You can use the CICS-MQ adapter control panels to display detailed information about CICS tasks that are using MQI calls, including their transaction IDs and task numbers.

About this task

If you want to see only the number of tasks using the connection, you can use any of the methods described in Displaying CICS-MQ connection status and settings. The **INQUIRE MQCONN** commands and the CICS-MQ adapter **Display Connection** panel also specify the number of trigger monitor (CKTI) tasks that are using the connection.

The CICS-MQ adapter **Display Task** panel provides details for each CICS task using the connection, as follows:

- Transaction ID (name)
- User ID
- CICS task number
- Task status
- Thread status
- Total number of API calls issued by this task
- Whether resource security checking is active for this task
- Whether this task is currently in the CICS-MQ API-crossing exit
- Most recent API call issued by this task
- Thread ID used by IBM MQ

Figure 6 on page 50 shows the layout of this information. For detailed information about each attribute, see the CKQC help panel, accessed by pressing PF1.

Procedure

To display the task information using the CICS-MQ adapter control panels:

a) Type CKQC and press Enter to access the CICS-MQ adapter control panels.

b) Select **Task** from the menu bar.

c) To obtain information about all tasks that are currently active, select the **List all tasks** action from the menu.

d) To specify the starting number of the first task to be displayed, select the **List from task** action from the menu.

Example

```
CKQCM3                        Display Task panel

Read task status information. Then press F12 to cancel.

Tasks   1 to   3 of   3

Tran    User   Task   Task     Thread     Total    Res  API    Last     Thread
 Id      Id    Num   Status   Status      APIs    Sec Exit   MQ call     ID
---- -------- ----- -------- --------  ---------- ---  --- ---------- --------
PUTQ CICSUSER 00065  Normal  In Queue         102 No   No    MQPUT1   00012420
GETQ CICSUSER 00067  Normal  Between           22 No   No    MQOPEN   00012620
CKTI CICSUSER 00123  Normal  Msg Wait           2 No   No    MQGET    00012C20









F1=Help  F7=Backward  F8=Forward  F12=Cancel  Enter=Refresh
```

Figure 6. The CICS-MQ adapter Display Task panel

Starting a CICS-MQ monitor

If set up properly, a CICS-MQ monitor can automatically start when the MQ connection is established. You can also manually start a CICS MQ monitor in several ways.

Before you begin

- The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.
- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).
- If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). If you want to control when a stopped MQ monitor starts, you must set AUTOSTART(NO) for this MQMONITOR resource. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

Procedure

- To automatically start a CICS MQ monitor when the MQ connection is established, you should define the MQMONITOR resource with attributes AUTOSTART(YES) and STATUS(ENABLED).

  When the connection between CICS and MQ has been established, CICS will start MQ monitors that have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED) if the user ID associated with the task that set the MQCONN resource to CONNECTED has sufficient authority to start the associated transactions.
- To manually start a CICS-MQ monitor, you can use any of the following methods:
  - Using a CICS application program that issues **EXEC CICS SET MQMONITOR**

You can start an MQ monitor by issuing an **EXEC CICS SET MQMONITOR** command in a CICS application program.

- Using a CICSPlex SM application program that issues **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(START)**

  You can start an MQ monitor by issuing an **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(START)** command in a CICSPlex SM application program.

- Using the CICS CEMT transaction

  You can start an MQ monitor by issuing the **SET MQMONITOR** command in the CICS CEMT transaction.

- Using the CICSPlex SM web user interface

  You can start an MQ monitor from the **WebSphere MQ monitor** view in the CICSPlex SM web user interface (WUI).

- Using the CICS Explorer

  You can start a CICS-MQ monitor from the CICS Explorer. The CICS Explorer provides a functional equivalent to the CICSPlex SM web user interface and the CEMT transaction.

Starting a CICS MQ monitor by using EXEC CICS SET MQMONITOR from a CICS application program

You can start an MQ monitor by issuing an **EXEC CICS SET MQMONITOR** command in a CICS application program.

Before you begin

- The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.
- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).
- If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). If you want to control when a stopped MQ monitor starts, you must set AUTOSTART(NO) for this MQMONITOR resource. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

Procedure

To start an MQ monitor, issue the following command in the CICS application program:

```
SET MQMONITOR (name)
    MONSTATUS(STARTED)
```

where *name* is the name of the MQMONITOR resource definition that you want to start.

Starting a CICS MQ monitor by using EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(START) from a CICSPlex SM application program

You can start an MQ monitor by issuing an **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(START)** command in a CICSPlex SM application program.

Before you begin

- The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.
- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).
- If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). If you want to control when a stopped MQ monitor starts, you must set AUTOSTART(NO) for this MQMONITOR resource. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

Procedure

To start an MQ monitor, issue the following command in the CICSPlex SM application program:

```
EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(START)
```

where *MQMON* is the name of the MQMONITOR resource definition that you want to start.

Starting a CICS MQ monitor through the CICS CEMT transaction

You can start an MQ monitor by issuing the **SET MQMONITOR** command in the CICS CEMT transaction.

Before you begin

- The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.
- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).
- If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). If you want to control when a stopped MQ monitor starts, you must set AUTOSTART(NO) for this MQMONITOR resource. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

About this task

For details of how to start and use the CEMT transaction, see CEMT - master terminal.

Procedure

1. On the CICS command line, enter the following command:

```
CEMT SET MQMONITOR (name)
```

where *name* is the name of the MQMONITOR resource that you want to start.

The AUTOSTART, ENABLESTATUS, and MONSTATUS values of the specified MQMONITOR resource definition are displayed.

2. If the MQMONITOR resource definition is disabled, overtype the value DISABLED in the ENABLESTATUS field with the value ENABLED to enable the resource for use in the CICS region.
3. Overtype the value STOPPED in the MONSTATUS field with the value STARTED to start the MQ monitor.
4. If AUTOSTART is not enabled, you can enable AUTOSTART for the MQMONITOR by overtyping the value NOAUTOSTART in the AUTOSTART field with the value AUTOSTART.

Starting a CICS MQ monitor through the CICSPlex SM web user interface

You can start an MQ monitor from the **WebSphere MQ monitor** view in the CICSPlex SM web user interface (WUI).

Before you begin

- The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.
- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).
- If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). If you want to control when a stopped MQ monitor starts, you must set AUTOSTART(NO) for this MQMONITOR resource. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

About this task

In the CICS Explorer, the MQ Monitors (MQMON) view provides a functional equivalent to this view in the WUI.

Procedure

1. From the CICSPlex SM WUI main menu, select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ monitors**.

   For a full listing and explanation of the information in this viewset, see WebSphere MQ monitors - MQMON.
2. Select the **Record** box for the WebSphere MQ monitor that you want to start.
3. Click the **Start** button.

Stopping a CICS MQ monitor

When the MQ connection is stopped, the MQ monitor is stopped automatically. You can also manually stop a CICS-MQ monitor in several ways.

If the z/OS Workload Manager (WLM) health service is active (see WLMHEALTH), with every increment in the z/OS WLM **HEALTH** value of the CICS region from zero to 100%, an attempt is made to start all stopped MQ monitors that have been defined with AUTOSTART(YES). Therefore, if you want certain MQ monitors to remain stopped during this period, you must set AUTOSTART(NO) for these MQMONITOR

resources. For more information, see Effect of z/OS Workload Manager Health service on MQMONITORs and Alert monitor (CKAM).

Procedure

You can use any of the following methods to manually stop an MQ monitor:

- Using a CICS application program that issues **EXEC CICS SET MQMONITOR**

  You can stop an MQ monitor by issuing an **EXEC CICS SET MQMONITOR** command in a CICS application program.
- Using a CICSPlex SM application program that issues **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(STOP)**

  You can stop an MQ monitor by issuing an **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(STOP)** command in a CICSPlex SM application program.
- Using the CICS CEMT transaction

  You can stop an MQ monitor by issuing the **SET MQMONITOR** command in the CICS CEMT transaction.
- Using the CICSPlex SM web user interface

  You can stop an MQ monitor from the **WebSphere MQ monitor** view in the CICSPlex SM web user interface (WUI).
- Using the CICS Explorer

  You can stop a CICS-MQ monitor from the CICS Explorer. The CICS Explorer provides a functional equivalent to the CICSPlex SM web user interface and the CEMT transaction.

Stopping a CICS MQ monitor by using EXEC CICS SET MQMONITOR from a CICS application program

You can stop an MQ monitor by issuing an **EXEC CICS SET MQMONITOR** command in a CICS application program.

Procedure

To stop an MQ monitor, issue the following command in the CICS application program:

```
SET MQMONITOR (name)
    MONSTATUS(STOPPED)
```

where *name* is the name of the MQMONITOR resource definition that you want to stop.

Stopping a CICS MQ monitor by using EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(STOP) from a CICSPlex SM application program

You can stop an MQ monitor by issuing an **EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(STOP)** command in a CICSPlex SM application program.

About this task

The MQMONITOR resource must be installed and enabled for use in the CICS region. For details, see Setting up the CICS-MQ adapter.

Procedure

To stop an MQ monitor, issue the following command in the CICSPlex SM application program:

```
EXEC CPSM PERFORM SET OBJECT(MQMON) ACTION(STOP)
```

where *MQMON* is the name of the MQMONITOR resource definition that you want to stop.

Stopping a CICS MQ monitor through the CICS CEMT transaction

You can stop an MQ monitor by issuing the **SET MQMONITOR** command in the CICS CEMT transaction.

About this task

For details of how to start and use the CEMT transaction, see CEMT - master terminal.

Procedure

1. On the CICS command line, enter the following command:

   ```
   CEMT SET MQMONITOR (name)
   ```

   where *name* is the name of the MQMONITOR resource that you want to stop.

   The AUTOSTART, ENABLESTATUS, and MONSTATUS values of the specified MQMONITOR resource definition are displayed.
2. Overtype the value STARTED in the MONSTATUS field with the value STOPPED to stop the MQ monitor.

Stopping a CICS MQ monitor through the CICSPlex SM web user interface

You can stop an MQ monitor from the **WebSphere MQ monitor** view in the CICSPlex SM web user interface (WUI).

About this task

In the CICS Explorer, the MQ Monitors (MQMON) view provides a functional equivalent to this view in the WUI.

Procedure

1. From the CICSPlex SM WUI main menu, select **CICS operations** > **DB2, DBCTL and WebSphere MQ operations views** > **WebSphere MQ monitors**.

   For a full listing and explanation of the information in this viewset, see WebSphere MQ monitors - MQMON.
2. Select the **Record** box for the WebSphere MQ monitor that you want to stop.
3. Click the **Stop** button.

Displaying information about a CICS MQ monitor

You can use the **CEMT INQUIRE MQMONITOR** or **EXEC CICS INQUIRE MQMONITOR** command, or CICSPlex SM to display information about an MQ monitor.

About this task

You can use the **INQUIRE MQMONITOR** command to inquire on all installed MQMONITOR resources in a CICS region. You can find out the following information about installed MQMONITOR resources:

- Whether an MQMONITOR resource is enabled or disabled.
- Whether an MQMONTIOR is enabled for automatic restart.
- Whether an MQMONTIOR is started or stopped, and if started, the ID of the task currently monitoring the MQ queue. Use **CEMT INQUIRE TASK** to see the details of this task.
- Data to be passed to the task monitoring the MQ queue (MONDATA attribute). For example, parameters that are passed to the MQ bridge monitor transaction CKBR.

  Note:

  When displayed and retrieved by the monitoring task, the MONDATA data is prepended with the following 18 bytes:

Byte 1: < (left chevron)
Bytes 2 - 9: *MQMONITOR resource name*
Bytes 10 - 17: *USERID*
Byte 18: > (right chevron)

Bytes 19 - 218 contains MONDATA as entered by the user.

- The name of the transaction used by the task monitoring the MQ queue.
- The name of the MQ queue that is being monitored by the MQ monitor.

Resetting CICS-MQ connection statistics

You can use the CKQC transaction to reset only the CICS-MQ statistics, or you can use standard CICS statistics reset commands to reset all the statistics, including the CICS-MQ statistics. CICS-MQ connection statistics are also reset at the end of a statistics interval.

Procedure

- To reset only the CICS-MQ connection statistics from the CICS command line, issue the command `CKQC MODIFY Y`. The option Y resets the statistics.
- To reset only the CICS-MQ connection statistics using a CICS application program, issue an **EXEC CICS LINK** command to link to the adapter reset program, DFHMQRS (or CSQCRST, which is retained for compatibility), and issue the **CKQC MODIFY** command with the option Y.
  This example shows you how to do this:

```
EXEC CICS LINK PROGRAM('DFHMQRS ')
          INPUTMSG('CKQC MODIFY     Y')
```

  The MODIFY command must be padded with 4 trailing spaces plus another space as a separator (see "CKQC commands from CICS application programs" on page 32).
- To reset only the CICS-MQ connection statistics using the CICS-MQ adapter control panels:
  a) Type CKQC and press Enter to access the CICS-MQ adapter control panels.
  b) Select **Connection** from the menu bar.
  c) Select the **Modify** action from the menu.
  d) In the **Modification Options** secondary parameter window, select **Reset statistics** and press Enter.
- To reset all the CICS statistics, including the CICS-MQ connection statistics, choose one of the standard CICS statistics reset methods as follows:
  - Use the CEMT transaction to issue the **CEMT SET STATISTICS** command with the RESETNOW option.
  - In a CICS application program, issue the **EXEC CICS SET STATISTICS** command with the RESETNOW option, or the **EXEC CICS PERFORM STATISTICS RECORD ALL RESETNOW** command.

Purging tasks that are using the CICS-MQ connection

You can use the CICS CEMT transaction to purge user tasks that are using the CICS-MQ adapter.

About this task

Tasks that are waiting on the adapter respond only to **CEMT SET TASK FORCEPURGE** commands. **CEMT SET TASK PURGE** commands are ignored. For the full syntax of the **CEMT SET TASK** command, see CEMT SET TASK.

Procedure

1. Use the CICS-MQ adapter control panels to display details of tasks that are using the CICS-MQ connection, including their task numbers. Note the numbers of any tasks that you want to purge.
   "Displaying tasks that are using the CICS-MQ connection" on page 49 explains how to do this.
2. For each task that you want to forcepurge, enter the command `CEMT SET TASK(number) FORCEPURGE` on the CICS command line, where *number* is the task number for the task.

Results

The way that the CICS-MQ adapter handles a FORCEPURGE command depends on the kind of wait state for the task:

- If a task is waiting for a message to arrive, for example, if the application has issued an `MQGET WAIT` call, the task is stopped with code AEXY immediately.
- Otherwise, the adapter waits for the request to complete, and then checks whether it is suitable to end the task.
  - If the task is in a critical state, the adapter lets the task continue and ignores the attempt to purge it, to preserve data and system integrity. Message DFHMQ0415I is displayed. A task is in a critical state when, for example, it is in the process of completing phase 2 of a 2-phase commit sequence.
  - If the task is not in a critical state, the adapter ends it with code AEXY. Message DFHMQ0414I is displayed.

The CICS-MQ trigger monitor

CKTI is the CICS-supplied trigger monitor (or task initiator) transaction, used in a CICS environment to start a transaction when the trigger conditions on any of its associated queues are met. The CICS-MQ trigger monitor CKTI provides the ability to track transactions initiated by the MQ trigger monitor. The benefit of this feature provides administrators with the ability to visualize and track work, therefore offering further value.

To a queue manager, a trigger monitor is like any other application that serves a queue. However, a trigger monitor serves initiation queues.

A trigger monitor is usually a continuously running program. When a trigger message arrives on an initiation queue, the trigger monitor retrieves that message. It uses information in the message to issue a command to start the application that is to process the messages on the application queue.

The trigger monitor must pass sufficient information to the program that it is starting so that the program can perform the right actions on the right application queue.

CKTI

You can start multiple instances of CKTI for each initiation queue. CKTI passes the MQTM structure of the trigger message to the program that it starts by **`EXEC CICS START TRANSID`**. The started program obtains this information by using the **`EXEC CICS RETRIEVE`** command. For more information about the MQTM structure, see MQTM trigger message.

A program can use the **`EXEC CICS RETRIEVE`** command with the `RTRANSID` option to determine how the program was started; if the value returned is CKTI, the program was started by the CICS-MQ trigger monitor.

For an example of how to use CKTI, see the source code supplied for module CSQ4CVB2 in the Credit Check sample application that is supplied with IBM MQ for z/OS. For more information about this sample, see Credit check sample.

Using MQMONITOR to manage CKTI

Using the MQMONITOR resource is the recommended method of controlling instances of CKTI. The benefits are as follows:

- You can set up multiple instances of CKTI to monitor an MQ initiation queue.
- CKTI can be started automatically when the connection to the MQ queue manager is established.
- MQMONITOR allows the CKTI or a message consuming transaction to run under a preset user ID.

CAUTION: Using the CKQC transaction and MQMONITORs at the same time to manage instances of the CKTI transaction can lead to confusing statistics and STAT of an MQMONITOR because CKQC is not aware of MQMONITORs and because MQMONITORs are not aware of CKTI transactions managed by using CKQC.

Important: If you want to use your own MQ trigger monitors to serve MQ queues in your CICS environment, instead of using the CICS-supplied trigger monitor, ensure that you follow the instructions in "Developing and using user-written MQ trigger monitors and MQ message consumers" on page 65 when designing and implementing your user-written MQ trigger monitor program.

Starting an instance of CKTI

You can start an instance of CKTI (the CICS-MQ trigger monitor or task initiator) in several ways. Using the MQMONITOR resource is the recommended method of controlling instances of CKTI.

About this task

User IDs for CKTI

If an instance of CKTI is started by an MQMONITOR, the user ID associated with the CKTI transaction is obtained as follows:

- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), the user ID is obtained from the **MONUSERID** attribute of the MQMONITOR resource.
- If security checking is disabled (that is, **SEC** is set to NO), the user ID is the user ID of the transaction that set the MQMONITOR state to started.

If CKTI is started from a terminal from the CKQC transaction or a user-written program that links to DFHMQSSQ (or CSQCSSQ, which is retained for compatibility), the user ID that CKTI uses is the same as the user ID of the terminal that started CKTI.

If CKTI starts other CICS transactions, such as user-written CICS applications, the user ID of CKTI is propagated to these applications. For example, if CKTI is running under user ID CIC1 and a trigger event occurs that requires the sender MCA transaction, CKSG, to be started, the CKSG transaction also runs under user ID CIC1. Therefore user ID CIC1 must have access to the required transmission queue.

Procedure

You can use any of the following methods to start CKTI:

- Using a CICS MQ monitor

  The MQMONITOR resource, if set up properly, enables the CKTI transaction to start automatically when the connection to the MQ queue manager is established. If the MQMONITOR that controls CKTI is not automatically started, you can manually start the MQ monitor in several ways.
- Using the CICS-MQ adapter control panel

  You can run the CICS-MQ adapter control transaction, CKQC, and then start instances of the CKTI transaction from the CICS-MQ adapter control panel.
- Issuing **CKQC STARTCKTI** from the CICS command line

  You can issue the **CKQC STARTCKTI** command to start an instance of CKTI that serves the default initiation queue or an initiation queue that you specify.
- Using a CICS application program

  To start an instance of CKTI from a CICS application program, the application program must link to the adapter task initiation program, DFHMQSSQ (or CSQCSSQ, which is retained for compatibility). If you

are using an MQMONITOR to control CKTI, you can start the MQMONITOR from an application program that issues `EXEC CICS SET MQMONITOR (name) STARTED`.

- Using an automation product

  To automate the starting of trigger monitors under a specific user ID, you can use an automation product, for example, NetView. You can use it to sign on to a CICS console and issue the **STARTCKTI** command.

Starting CKTI using a CICS MQ monitor

The MQMONITOR resource, if set up properly, enables the CKTI transaction to start automatically when the connection to the MQ queue manager is established. If the MQMONITOR that controls CKTI is not automatically started, you can manually start the MQ monitor in several ways.

Before you begin

- If you want to use MQMONITOR to control instances of CKTI in a CICS region, you must define and install such MQMONITOR resources. Table 8 on page 59 lists some important attributes that you should specify for an MQMONITOR that controls CKTI.

  Note that attributes AUTOSTART(YES) and STATUS(ENABLED) should be defined for an MQMONITOR that is to automatically restart when the MQ connection is established. If you want to disable automatic restart and prefer to manually start the MQ monitor, define the MQMONITOR resource with AUTOSTART(NO) or without the AUTOSTART attribute.

Table 8. MQMONITOR resource attributes for monitoring an initiation queue

Attribute	Required / Optional	Default	Description
AUTOSTART	Optional	YES	This attribute controls auto-restart of the MQ monitor: **AUTOSTART(YES)** Enables CKTI to restart automatically when the connection to the MQ queue manager is established. **AUTOSTART(NO)** The MQ monitor is not started automatically. After the connection to the MQ queue manager is established, you have to manually start the MQ monitor.
MONUSERID	Required	-	Specify the user ID to be associated with CKTI. This attribute is only effective when security checking is active (that is, the **SEC** system initialization parameter is set to YES).
QNAME	Optional	If omitted, the default is *&APPLID.*.INITIATION.QUEUE where *&APPLID.* is the applid of the CICS region	Specify the name of the MQ initiation queue to be monitored.
STATUS	Optional	ENABLED	This attribute makes the resource available for use in the region.

Table 8. MQMONITOR resource attributes for monitoring an initiation queue (continued)			
Attribute	**Required / Optional**	**Default**	**Description**
TRANSACTION	Optional	CKTI	Specifies the 4-character ID of the CICS-supplied trigger monitor (or task initiator) transaction. **Note:** If you are using your own MQ trigger monitor, specify the name of this transaction. For considerations about using your own MQ trigger monitors in your CICS environment, see "The CICS-MQ trigger monitor" on page 57.

- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).

Procedure

- To automatically start an MQ monitor that controls CKTI, check that the MQMONITOR resource has the attributes AUTOSTART(YES) and STATUS(ENABLED).

  When the connection between CICS and MQ has been established, CICS will start MQ monitors that have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED) if the user ID associated with the task that set the MQCONN resource to CONNECTED has sufficient authority to start the associated transactions. Then, an instance of CKTI will be running.
- To manually start an MQ monitor that controls CKTI, you can use any of the methods as described in "Starting a CICS-MQ monitor" on page 50.

Starting CKTI from the CICS-MQ adapter control panel

You can run the CICS-MQ adapter control transaction, CKQC, and then start instances of the CKTI transaction from the CICS-MQ adapter control panel.

Procedure

1. On the CICS-MQ adapter control initial panel, select **CKTI** from the menu bar.
2. Select the **Start** action from the menu.
3. In the **Start Task Initiator** secondary window, use the **Initiation Queue Name** field to specify the name of the initiation queue to be serviced by this CKTI instance.

   If you leave this field blank, the default initiation queue is used, if defined.

Example

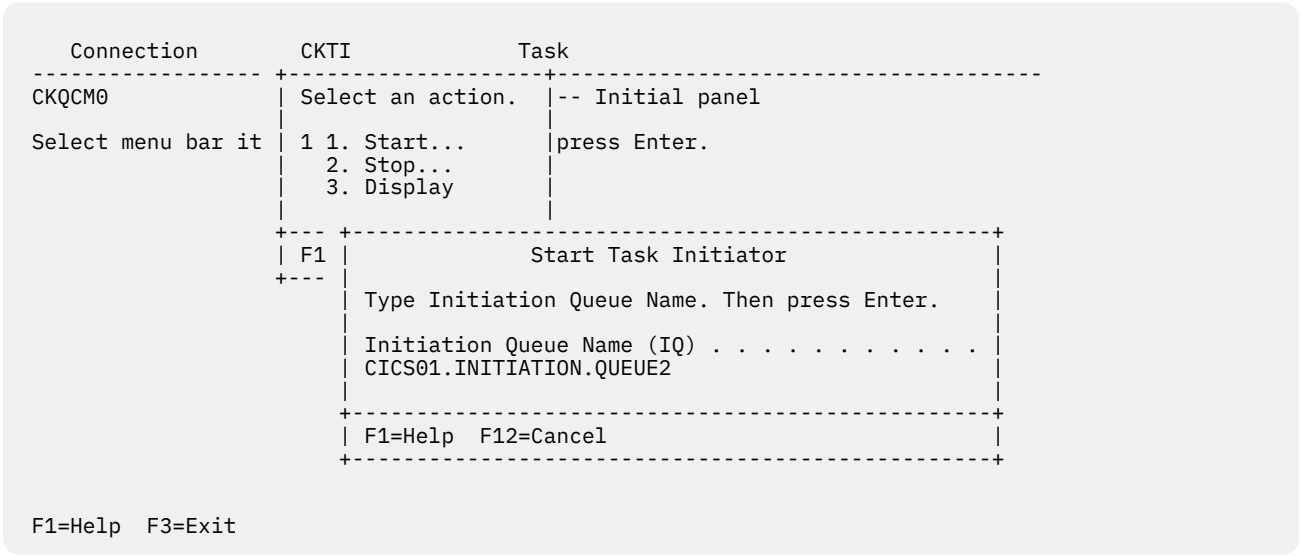


Figure 7. Starting an instance of CKTI

Starting CKTI from the CICS command line
You can issue the **CKQC STARTCKTI** command to start an instance of CKTI that serves the default initiation queue or an initiation queue that you specify.

About this task

If you issue a **CKQC STARTCKTI** command without specifying an initiation queue, this command is automatically interpreted as referring to the default initiation queue for the CICS region, `regionAPPLID.initiation.queue`.

Procedure

- Use this command to start an instance of CKTI to serve the default initiation queue, if defined:

  ```
  CKQC STARTCKTI
  ```

- Use this command to start an instance of CKTI to serve a specified initiation queue `CICS01.INITIATION.QUEUE2`:

  ```
  CKQC STARTCKTI CICS01.INITIATION.QUEUE2
  ```

Starting CKTI from a CICS application program
To start an instance of CKTI from a CICS application program, the application program must link to the adapter task initiation program, DFHMQSSQ (or CSQCSSQ, which is retained for compatibility). If you are using an MQMONITOR to control CKTI, you can start the MQMONITOR from an application program that issues `EXEC CICS SET MQMONITOR (`*name*`) STARTED`.

About this task

Using an application program that links to DFHMQSSQ
When you do an **EXEC CICS LINK** to DFHMQSSQ, the program requires a terminal associated task.

The STARTCKTI command must be padded to 10 characters; see "Command syntax in application programs" on page 32.

Using an application program that issues EXEC CICS SET MQMONITOR (*name*) STARTED
The application program will start the MQMONITOR that controls CKTI if the following conditions are met:

- The current tranid is not the same as the value in the TRANID attribute.
- The MQMONITOR is not already in STARTED state.
- The user ID associated with the current transaction is authorized to set the state of the transaction associated with the MQMONITOR to started.

Procedure

- To link to DFHMQSSQ and start a CKTI that uses the default initiation queue, issue a command like this:

```
EXEC CICS LINK PROGRAM('DFHMQSSQ ')
          INPUTMSG('CKQC STARTCKTI ')
```

- To link to DFHMQSSQ and start a CKTI that uses the initiation queue `CICS01.INITIATION.QUEUE2`, issue a command like this:

```
EXEC CICS LINK PROGRAM('DFHMQSSQ ')
          INPUTMSG('CKQC STARTCKTI  CICS01.INITIATION.QUEUE2')
```

- To start the MQMONITOR that controls CKTI, issue the following command in the application program:

```
SET MQMONITOR (name)
    MONSTATUS(STARTED)
```

  where *name* is the name of the MQMONITOR that controls CKTI.

Results

Output messages from DFHMQSSQ are displayed on the system console.

Starting CKTI automatically through an automation product
To automate the starting of trigger monitors under a specific user ID, you can use an automation product, for example, NetView. You can use it to sign on to a CICS console and issue the **STARTCKTI** command.

You can also use preset security sequential terminals, which have been defined to emulate a CRLP terminal, with the sequential terminal input containing the **CKQC STARTCKTI** command.

However, when the adapter alert monitor reconnects CICS to IBM MQ, for example, after a queue manager restart, only the CKTI specified at the initial IBM MQ connection is restarted. You must automate the starting of any extra CKTIs yourself.

Stopping an instance of CKTI

You can stop an instance of CKTI in several ways. Using the MQMONITOR resource is the recommended method of controlling instances of CKTI.

About this task

If you are using MQ monitors to control CKTI, when the MQ connection is stopped, the MQ monitors are automatically stopped.

If you are using MQ monitors in the region to control CKTI, when there are multiple instances of CKTI running, using CKQC to stop an instance of CKTI could cause unpredictable results.

Procedure

- You can stop an instance of CKTI using any of the following methods:
  - Stopping the CICS MQ monitor that controls CKTI

    If you are using an MQ monitor to control instances of CKTI, you can stop the MQ monitor in several ways.
  - Using the CICS-MQ adapter control panel

You can run the CICS-MQ adapter control transaction, CKQC, and then stop instances of the CKTI transaction from the CICS-MQ adapter control panel.

- Issuing a **CKQC STOPCKTI** command from a terminal

  You can issue the **CKQC STOPCKTI** command from a terminal to stop an instance of CKTI.

- Using an application program

  You can stop an instance of CKTI by linking to the adapter task-initiator program, DFHMQSSQ (or CSQCSSQ, which is retained for compatibility).

Stopping an instance of CKTI from the CICS-MQ adapter control panel
You can run the CICS-MQ adapter control transaction, CKQC, and then stop instances of the CKTI transaction from the CICS-MQ adapter control panel.

Procedure

1. On the CICS-MQ adapter control initial panel, select **CKTI** from the menu bar.
2. Select the **Stop** action from the menu.
3. Use the **Stop Task Initiator** secondary window to specify the name of the initiation queue serviced by this instance of CKTI.

   If you leave the name blank, the default initiation queue is used, if defined.

Example

```
   Connection        CKTI        Task
------------------ +-------------------+--------------------------------------
CKQCM0             | Select an action.  |-- Initial panel
                   |                    |
Select menu bar it | 2 1. Start...      |press Enter.
                   |   2. Stop...       |
                   |   3. Display       |
                   |                    |
                   +--- +-----------------------------------------------+
                   | F1 |              Stop Task Initiator               |
                   +--- |                                               |
                        | Type Initiation Queue Name. Then press Enter.  |
                        |                                               |
                        | Initiation Queue Name (IQ) . . . . . . . . . . |
                        | CICS01.INITIATION.QUEUE2                      |
                        |                                               |
                        +-----------------------------------------------+
                        | F1=Help  F12=Cancel                           |
                        +-----------------------------------------------+


F1=Help  F3=Exit
```

Figure 8. Stopping an instance of the task initiator CKTI

Stopping an instance of CKTI from a terminal
You can issue the **CKQC STOPCKTI** command from a terminal to stop an instance of CKTI.

About this task

If you issue a **CKQC STOPCKTI** command without specifying an initiation queue, this command is automatically interpreted as referring to the default initiation queue for the CICS region, `regionAPPLID.initiation.queue`.

Procedure

- Use this command to stop an instance of CKTI that is serving the default initiation queue:

  ```
  CKQC STOPCKTI
  ```

- Use this command to stop an instance of CKTI that serves a specified initiation queue:

  ```
  CKQC STOPCKTI queue_name
  ```

  where *queue_name* is the name of the initiation queue.

  Note: If you are using MQMONITOR resources, be aware that this command will also stop all MQMONITORs that are monitoring the specified MQ queue.

  For example, to stop an instance of CKTI that serves `CICS01.INITIATION.QUEUE2`, issue the following command:

  ```
  CKQC STOPCKTI CICS01.INITIATION.QUEUE2
  ```

Stopping an instance of CKTI from an application program
You can stop an instance of CKTI by linking to the adapter task-initiator program, DFHMQSSQ (or CSQCSSQ, which is retained for compatibility).

About this task

These examples show alternative LINK commands to stop an instance of CKTI from a CICS program. The STOPCKTI command must be padded to 10 characters; see "Command syntax in application programs" on page 32.

Procedure

- This command stops the CKTI that is serving the default initiation queue:

  ```
  EXEC CICS LINK PROGRAM('DFHMQSSQ ')
            INPUTMSG('CKQC STOPCKTI  ')
  ```

- This command stops the CKTI serving a specified initiation queue:

  ```
  EXEC CICS LINK PROGRAM('DFHMQSSQ ')
            INPUTMSG('CKQC STOPCKTI   CICS01.INITIATION.QUEUE2')
  ```

Displaying the current instances of CKTI

You can use the CICS-MQ adapter control panels to display details of the current instances of CKTI. The equivalent functionality is not available from the CICS command line or from a CICS application program.

Procedure

1. On the CICS-MQ adapter control initial panel, select **CKTI** from the menu bar.
2. Select the **Display** action from the pull-down menu.
   Figure 9 on page 65 shows the details provided for each instance of CKTI:
   - CICS task number
   - Task status
   - Thread status
   - Number of API calls it has issued
   - Most recent API call it has issued
   - Name of the initiation queue it is serving
3. Press function key F1 to display help information about each field in the panel.

Example

```
CKQCM4                        Display CKTI panel

Read CKTI status information. Then press F12 to cancel.

CKTI   1 to   1 of   1

 Task Num    Task Status    Thread Status    Num of APIs    Last API
----------  -------------  ---------------  -------------  ----------
 0000123       Normal         Msg Wait                2      MQGET
   Initiation Queue Name: CICS01.INITIATION.QUEUE1









F1=Help  F7=Backward  F8=Forward  F12=Cancel  Enter=Refresh
```

Figure 9. The CKQC Display CKTI panel

Developing and using user-written MQ trigger monitors and MQ message consumers

You can create your own MQ trigger monitors and use them to serve MQ queues in your CICS environment, instead of using the CICS-supplied trigger monitor CKTI. You can also create your own MQ message consumers that get messages directly from application input queues and perform the required logic. This topic provides important instructions for developing and using user-written MQ trigger monitors and MQ message consumers.

Responsibilities of a user-written MQ monitor or MQ message consumer program

If you are using an MQMONITOR to process the MQ queue, when the MQMONITOR is started, the **EXEC CICS START** command starts your transaction with **MONDATA** prepended with *<MQMONITOR_resource_nameUSERID>* as the FROM data, so the user-written program must retrieve this data into a structure that matches the description for the **MONDATA** attribute before issuing any calls to MQ.

Then, before opening the MQ queue, the program must set the state of the MQ monitor as specified in the retrieved data to started.

Before returning to CICS, the program must set the state of the MQ monitor to stopped.

Security considerations when your MQ message consumer is used to route work to remote regions

If you are using your MQ message consumer to route work to remote regions, you must take into consideration any effect the security configuration your intersystem communication definitions might have on determining the user ID that will be associated with the transaction in the remote region. For details, see the information about routed transactions started by **START** commands in START.

Procedure

Your user-written MQ monitor or MQ message consumer program must perform the following steps in the sequence as indicated below:

1. Check that the transaction has been started with data by using an **EXEC CICS ASSIGN STARTCODE** command.
2. Retrieve the FROM data that was passed when your transaction was started into a structure, by issuing an **EXEC CICS RETRIEVE** command.

   Use the MQMONITOR name that is passed in bytes 2 through 9 of this data for the following steps.
3. Issue EXEC CICS SET MQMONITOR(*MQMONITOR_name*) STARTED to set the state of the MQ monitor as specified in the retrieved data to started.
4. Open the MQ queue.
5. Get messages on the input queue and do the required application logic.
6. When ending the program for any reason, issue EXEC CICS SET MQMONITOR(*MQMONITOR_name*) STOPPED to set the state of the MQ monitor to stopped. Write out the CICS statistics collected when the MQMONITOR was started.
7. Return to CICS.

CAUTION:

- If the user-written MQ trigger monitor or MQ message consumer program fails to perform the steps in the sequence as described in the preceding procedure, unpredictable consequences might occur; for example, recorded statistics will be unpredictable, and the MQMONITOR's MONSTATUS value will be in an unpredictable state.
- The CICS-supplied MQ trigger monitor program DFHMQTSK is reserved for use with the CICS-MQ trigger monitor and task initiator transaction CKTI. Any attempt to invoke DFHMQTSK as a user transaction will cause the user transaction to abend with abend code AMQO.

Note: If the user-written MQ trigger monitor or MQ message consumer program issues **EXEC CICS START** requests, and if a policy that limits **EXEC CICS START** requests has been deployed and is active in the same region, the policy might abend the user-written program. In this situation, it is not suitable to activate policies that limit **EXEC CICS START** requests. For more information about policy rules, see Policy task rules.

Learn more

- CICS Developer Center: Using MQMONITORs to simplify the administration of CICS-MQ trigger monitors and MQ message consumers

  This blog shows you how to develop an MQ message consumer program that uses an MQMONITOR and provides a code example for your reference.

Chapter 4. Administering the CICS-MQ bridge

To start the bridge, run the CICS-MQ bridge monitor transaction. The default name of the transaction is CKBR, but you can define your own alternative transaction.

About this task

It is recommended that you use a CICS-MQ monitor to control the CICS-MQ bridge. An MQMONITOR resource allows the bridge to automatically restart when the connection to the IBM MQ manager is established. For setup instructions, see Setting up an MQMONITOR resource for the CICS-MQ bridge.

When you run CKBR, you can specify six optional parameters:

Q=*qqq*
qqq is the name of the IBM MQ request queue for the CICS-MQ bridge. If you have defined your own request queue, you must specify it here. Remember that names of objects in IBM MQ are case-sensitive. If you do not specify a queue name, CKBR uses the default request queue SYSTEM.CICS.BRIDGE.QUEUE.

AUTH=*LOCAL|IDENTIFY|VERIFY_UOW|VERIFY_ALL*
Specifies the level of authentication to use. The default is LOCAL. Note that with LOCAL, CICS programs run by the bridge task are started with the user ID under which the bridge monitor was started, which might affect your choice of method to start the bridge monitor. With the other levels of authentication, CICS programs run with the user ID extracted from the request message. See Security for the CICS-WebSphere MQ bridge for more information about the levels of authentication.

WAIT=*nnn*
nnn is the number of seconds that you want the bridge task to wait for subsequent requests before timing out when the bridge task is processing a unit of work that runs many user programs. This value must be in the range 0 through 999. If this parameter is not specified, the default is MQWI_UNLIMITED, which has a value of -1. You are recommended to specify a wait time, because if you do not specify a wait time, the bridge might inhibit the shutdown of CICS or the queue manager.

MSG=*CSMT|LOG|BOTH*
Determines whether messages generated by the CICS-MQ bridge are sent to the CICS job log, the CICS master terminal, or both. The default is BOTH.

PASSTKTA=*applid*
applid specifies the application ID to be used for validating the PassTicket. The default is the CICS region application ID. See Setting up security on z/OS in IBM MQ product documentation for information about PassTickets.

ROUTEMEM=*Y/N*
Determines whether mark expired messages are to be routed to the DLQ. If you do not specify the parameter the default is N, no routing.

Starting the CICS-MQ bridge

If you are using a CICS MQ monitor to control the bridge and if the MQMONITOR resource has been configured correctly, the bridge automatically starts when the connection to the MQ queue manager is established. Using an MQMONITOR resource is the recommended method of controlling the bridge. You can also manually start the bridge using the CKBR transaction.

Before you begin

- If you want to use an MQ monitor to control the bridge, you must define and install an MQMONITOR resource for controlling the MQ bridge. For detailed instructions, see Setting up an MQMONITOR resource for the CICS-MQ bridge.

- In addition, if security checking is active (that is, the **SEC** system initialization parameter is set to YES), ensure that the user ID associated with the transaction that attempts to set the MQ monitor state to started is a surrogate of the user ID defined in **MONUSERID** and is authorized to start transactions associated with the **MONUSERID**. In the case of setting the MQ monitor state through a CICSPlex SM API interface such as the CICS Explorer, the user ID to be associated with the MQ monitor transaction is either the region user ID or the PLTPIUSR user ID (if specified).

Procedure

- To automatically start the CICS-MQ bridge when the connection to the MQ queue manager is established, you must use an MQ monitor to control the bridge and ensure that this MQMONITOR resource has attributes AUTOSTART(YES) and STATUS(ENABLED).

  When the connection between CICS and MQ has been established, CICS will start MQ monitors that have been installed with attributes AUTOSTART(YES) and STATUS(ENABLED) if the user ID associated with the task that set the MQCONN resource to CONNECTED has sufficient authority to start the associated transactions.
- If you are using an MQ monitor to control the bridge, you can manually start the MQ monitor in several ways. Follow the instructions in "Starting a CICS-MQ monitor" on page 50.
- If you want to manually start the bridge using the CKBR transaction, start the CKBR task with one of the following methods:
  - Input a single line from a terminal (3270 or other):

```
CKBR Q=<queue name>,AUTH=<auth option>,WAIT=nnn,MSG=<msg
option>,PASSTKTA=<applid>,ROUTEMEM=<routemem option>
```

    For example:

```
CKBR Q=MyQueue,AUTH=IDENTIFY,WAIT=30,MSG=LOG,PASSTKTA=APP1,ROUTEMEM=Y
```

    Following your input, the terminal is unlocked so it can be used for other work.
  - Issue an **EXEC CICS START** command for the CKBR transaction with the parameters as data.

    You can have a program that runs as part of CICS PLTPI processing to issue this command, and specifies the user ID under which the bridge monitor transaction is to run.
  - Issue an **EXEC CICS LINK** to the program DFHMQBR0 (also known as CSQCBR00) with the parameters as data in the COMMAREA.

    DFHMQBR0 is a long-running task and this program returns only when the bridge stops.
  - Use TRIGGER TRIGTYPE(FIRST) on the bridge request queue to start a process specifying APPLICID(CKBR), with any parameters in USERDATA.

    Note that you cannot specify the **Q=qqq** parameter in USERDATA, so this method is not suitable if you have defined your own request queue.

What to do next

If you are running multiple bridge monitors sharing a request queue, you can start one of the monitors by putting a message onto the bridge request queue. However, for private local queues, only one trigger message is produced, so the bridge monitor starts on only one CICS region. To start further bridge monitors, you should therefore consider alternative methods, such as using a program in the CICS startup PLT processing to start the transaction with the required parameters, or using automation products to start the transaction. If you shut down the bridge by altering the request queue attributes, remember to reset GET(ENABLED) after the bridge has shut down.

Stopping the CICS-MQ bridge

You can shut down the CICS-MQ bridge, by altering the attributes of the request queue by setting GET(DISABLED), by shutting down CICS, or by shutting down the queue manager. If you are using a CICS MQ monitor to control the bridge, you can shut down the bridge by stopping the MQ monitor.

About this task

If you are using an MQ monitor to control the bridge, the bridge is automatically stopped when the connection to the MQ queue manager is stopped.

Procedure

- If you are using an MQ monitor to control the bridge, you can manually stop an MQ monitor in several ways. Follow the instructions in “Stopping a CICS MQ monitor” on page 53.

Chapter 5. Security for the CICS-MQ adapter

The CICS-MQ adapter provides information to IBM MQ specifically for use in IBM MQ security.

Information provided is as follows:

- Whether CICS resource-level security is active for this transaction. For more information, see Security of resource definitions.
- User IDs.
  - For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is one of the following:
    - The default CICS user ID as specified on the CICS **DFLTUSER** system initialization parameter.
    - A preset security user ID specified on the terminal definition.
  - For nonterminal tasks, the adapter acquires the user ID with a call to the user domain.

Implementing security for CICS-MQ adapter transactions

If you want a user to administer the CICS-MQ adapter, you must grant the user authorization to the appropriate CICS transactions.

If required, you can restrict access to specific functions of the adapter. For example, if you want to allow users to display the current status of the adapter, but nothing else, give them access to CKQC, CKBM, CKRT, and CKDP only.

Define these transactions to CICS with RESSEC(NO) and CMDSEC(NO). For more details, see Security of resource definitions and CICS command security.

Transaction	Function
CKAM	Alert monitor
CKBM	Controls the adapter functions
CKCN	Connect
CKDL	Line mode display
CKDP	Full screen display
CKQC	Controls the adapter functions
CKRS	Statistics
CKRT	Controls the adapter functions
CKSD	Disconnect
CKSQ	CKTI START/STOP
CKTI	Trigger monitor

As well as administrators, user IDs connecting to IBM MQ, the user ID set in the **PLTPIUSR** system initialization parameter, and the CICSPlex SM MAS agent user ID must also be authorized to run the CKTI and CKAM transactions.

Security for MQCONN and MQMONITOR commands

Use CICS command security to control users' ability to issue SPI commands against MQCONN and MQMONITOR resource definitions. For example, you can use it to control which users are allowed to issue CREATE and DISCARD commands against the MQCONN resource definition for the CICS region.

When command security is enabled for a transaction, the external security manager checks that the user ID associated with the transaction is authorized to use the command on the MQCONN or MQMONITOR resource as appropriate. Resource security is not available for MQCONN and MQMONITOR resources.

CICS command security covers the **EXEC CICS CREATE MQCONN**, **DISCARD MQCONN**, **SET MQCONN**, **INQUIRE MQCONN**, **CREATE MQMONITOR**, **DISCARD MQMONITOR**, **SET MQMONITOR**, and **INQUIRE MQMONITOR** commands. For an explanation of command security and instructions to set up command security for a CICS region, see CICS command security. For a listing of the level of authority required for each command, see Resource and command check cross-reference.

When command security is active, to use the **EXEC CICS SET MQCONN** command to start or stop the connection to IBM MQ, users must have authority to use the **EXEC CICS SET MQCONN** command, authority to issue **EXEC CICS START** commands for the transaction associated with the installed MQMONITOR resources, and also the authority to use the **EXEC CICS EXTRACT EXIT** command. If a user attempts to start or stop the connection when they do not have authority to use the **EXEC CICS EXTRACT EXIT** command, CICS issues messages DFHXS1111 and DFHMQ0302. If a user does not have authority to issue command **EXEC CICS START** specifying the transaction associated with the MQMONITOR, CICS issues message DFHMQ0390. User-written MQMONITOR programs must have authority to the **EXEC CICS SET MQMONITOR** command.

CICS-MQ adapter user IDs

The user ID associated with the CICS-MQ adapter is that of the IBM MQ-supplied task initiator transaction, CKTI.

User ID of CKTI started by MQMONITOR

If an instance of CKTI is started by an MQMONITOR, the user ID associated with the CKTI transaction is obtained as follows:

- If security checking is active (that is, the **SEC** system initialization parameter is set to YES), the user ID is obtained from the **MONUSERID** attribute of the MQMONITOR resource.
- If security checking is disabled (that is, **SEC** is set to NO), the user ID is the user ID of the transaction that set the MQMONITOR state to started.

User ID checking for IBM MQ resources during PLTPI and PLTSD

If an IBM MQ resource is accessed during the CICS PLTPI phase, the user ID passed to IBM MQ is that specified in the PLTPIUSR system initialization parameter. If an IBM MQ resource is accessed during the CICS PLTSD phase, the user ID passed to IBM MQ is the user ID associated with the shutdown transaction.

Connection security for the CICS-MQ adapter

IBM MQ carries out connection security checking either when an application program tries to connect to a queue manager by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS-MQ adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager, but, if you do that, any user can connect to that queue manager.

Only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID.

You can turn IBM MQ connection security checking on or off at either queue manager or queue-sharing group level.

Chapter 6. Security for the CICS-MQ bridge

When you start the CICS-MQ bridge, you can specify the level of authentication. If requested, the bridge monitor checks the user ID and password extracted from the IBM MQ request message before running the CICS program named in the request message.

When you run the CICS-MQ bridge monitor transaction (for example, CKBR or your transaction name), you can specify the **AUTH** parameter to select one of the following levels of authentication:

LOCAL
This level is the default. The bridge monitor starts the bridge task with the CICS default user ID. CICS user programs that the bridge task runs have the authority associated with this user ID. The IBM MQ request message cannot request higher authority because any user IDs or passwords in the message are ignored. If the bridge task runs a CICS program that tries to access protected resources, the CICS program might fail.

IDENTIFY
If the message descriptor (MQMD) in the request message specifies a user ID, the bridge monitor starts the bridge task with that user ID. CICS user programs that the bridge task runs have the authority associated with that user ID. The user ID is treated as trusted; that is, the bridge monitor does not authenticate the ID by using password or PassTicket information. If the MQMD does not specify a user ID, the bridge monitor starts the bridge task with the CICS default user ID, in the same way as the LOCAL option.

VERIFY_UOW
The bridge monitor uses the password or PassTicket to authenticate the user ID if all the following conditions apply:

- The message descriptor (MQMD) in the request message specifies a user ID.
- The request message includes an IBM MQ CICS information header (MQCIH).
- The Authenticator field in the MQCIH contains a password or PassTicket.

If authentication succeeds, the bridge monitor starts the bridge task with that user ID. If authentication fails, the bridge monitor fails the request with a MQCRC_SECURITY_ERROR return code.

If any one of the conditions listed earlier is not met, the bridge monitor starts the bridge task with the CICS default user ID, in the same way as the LOCAL option. Only the first request message in the unit of work is checked; the bridge ignores user ID and password or PassTicket information in subsequent messages that are part of the same unit of work.

VERIFY_ALL
This level is the same as VERIFY_UOW, except that the bridge task also checks that the user ID is the same in every request message in the same unit of work, and reauthenticates the user ID for each request message, using the password or PassTicket that the message contains.

If you require different levels of authentication for different applications, use multiple bridge monitors with different transaction IDs. You can use CICS surrogate security to restrict the combinations of transaction and user ID that a bridge monitor transaction and user ID can start.

Table 9 on page 75 shows the user ID under which the bridge monitor is started. The user ID depends on the method that you use to run the bridge monitor transaction, typically CKBR.

Table 9. CICS-MQ bridge monitor security

Bridge monitor start method	At a signed on terminal	User ID for bridge monitor
From a terminal or **EXEC CICS LINK** in a program	Yes	Signed-on user ID
From a terminal or **EXEC CICS LINK** in a program	No	CICS default user ID

Table 9. CICS-MQ bridge monitor security (continued)		
Bridge monitor start method	**At a signed on terminal**	**User ID for bridge monitor**
EXEC CICS START with user ID	–	User ID from START
EXEC CICS START without user ID	–	User ID from START
The CICS-MQ trigger monitor CKTI	–	User ID from START
The CICS MQ monitor (MQMONITOR)	–	• The **MONUSERID** attribute of the MQMONITOR resource, if security checking is active for the CICS region (that is, the **SEC** system initialization parameter is set to YES) • User ID that started the MQMONITOR resource, if security checking is disabled for the CICS region (that is, **SEC** is set to NO)

User IDs and passwords in request messages

When you use the IDENTIFY, VERIFY_UOW, or VERIFY_ALL authentication options, the bridge task and the CICS programs that it runs are started with the user ID that is specified in the message descriptor (MQMD) in the request message. With the VERIFY_UOW and VERIFY_ALL options, the bridge monitor also checks the password specified in the IBM MQ CICS information header (MQCIH) in the request message.

To use these levels of authentication, the IBM MQ applications must provide a user ID in the MQMD, and they must provide an MQCIH including the password. You must define these user IDs to RACF®. To control the user IDs used, the IBM MQ applications must open the request queue for the bridge monitor using open options that include MQOO_SET_IDENTITY_CONTEXT. The applications must also include a value of MQPMO_SET_IDENTITY_CONTEXT in their put message options.

If the bridge monitor finds a problem with the user ID or the password in a request message, it acts as follows:

- For the IDENTIFY level of authentication, if a message contains a user ID that was revoked, abend AICO might occur. The error reply has return code MQCRC_BRIDGE_ERROR with reason MQFB_CICS_BRIDGE_FAILURE.
- For the VERIFY_UOW or VERIFY_ALL level of authentication, if the user ID or password is invalid, the request fails with return code MQCRC_SECURITY_ERROR.
- If a request message omits either the user ID or the password, the bridge monitor runs the bridge task with the LOCAL level of authentication, even if you started the bridge monitor with one of the other authentication options. In that situation, the CICS programs started by the bridge task run with the user ID under which the bridge monitor was started.

You can use a PassTicket in place of a password to avoid the need to flow passwords in messages.

- The IBM MQ application must provide the PassTicket in the MQCIH in the request message.

- To generate a PassTicket, an application ID is required. The default application ID is the CICS APPLID. You can specify an alternative application ID by using the **PASSTKTA** parameter when you start the CICS-MQ bridge monitor transaction (for example, CKBR or your transaction name).
- If you use multiple bridge monitors for the same request queue, you must use the **PASSTKTA** parameter to specify the same application ID for each bridge monitor.

PassTicket validation is performed by using IBM MQ services, not **EXEC CICS VERIFY**, because the CICS service does not allow you to specify an APPLID. For more information about PassTickets, see Generating and using PassTickets for secure sign-on and Setting up security on z/OS in IBM MQ product documentation.

Authority

You must give the following permissions to the user IDs that you use with the CICS-MQ bridge. The user IDs include the user ID under which the bridge monitor transaction is started, as listed in Table 9 on page 75, and any user IDs that IBM MQ applications specify in request messages.

- The user ID under which the bridge monitor transaction is started must have authority to start the CKBP and CKBC transactions for CICS DPL programs, and any alternative transactions that IBM MQ applications specify in the TransactionId field in the MQCIH structure in request messages.
- If IBM MQ applications are specifying user IDs in request messages, the user ID under which the bridge monitor transaction is started must be defined to RACF as a surrogate of all the user IDs used in request messages. A surrogate user is one that has the authority to start work on behalf of another user, without knowing the password of the other user. For more information about surrogate user security, see Surrogate user security.
- The user IDs for the bridge monitor and for all bridge tasks need authority to get messages from the request queue.
- The user IDs for a bridge task need authority to put messages to its reply-to queue.
- To ensure that any error replies are received, the user ID under which the bridge monitor transaction is started must have authority to put messages to all reply-to queues.
- The user IDs for bridge tasks must have authority to put messages to the dead-letter queue.
- The user ID under which the bridge monitor transaction is started needs authority to put messages to the dead-letter queue, unless you want the bridge to stop if an error occurs.
- The user IDs for the bridge monitor and for all bridge tasks must have authority to put messages to the backout requeue queue, if one is defined.

Chapter 7. Developing applications to use the CICS-MQ bridge

A non-CICS application can communicate with a CICS program or transaction by sending and receiving IBM MQ messages over the CICS-MQ bridge. The data required by the CICS application is included in request messages, and the CICS-MQ bridge uses reply messages to return the data provided by the CICS application.

The following types of CICS application are suitable for use with the CICS-MQ bridge:

- CICS programs that are called using the **EXEC CICS LINK** command, known as DPL programs. The programs must conform to the DPL subset of the CICS API; that is, they must not use CICS terminal or sync point facilities. You can use the CICS-MQ bridge to run a single CICS program, or a set of CICS programs that form a unit of work.
- CICS transactions that were designed to be run from a 3270 terminal, known as 3270 transactions. The transactions can use Basic Mapping Support (BMS) or terminal control commands. They can be conversational or part of a pseudoconversation. They are permitted to issue sync points.

Typically, more complex application programming is required to run a 3270 transaction through the CICS-MQ bridge, because the non-CICS application must interact with the internal logic and flow of control in the CICS transaction. It is preferable to run a DPL program that contains the business logic of the CICS application. However, some CICS applications are not structured with the business logic of the application separated from the presentation logic, so the CICS-MQ bridge lets you communicate with either type of application.

A non-CICS application starts a CICS application by sending a structured IBM MQ message to the request queue for the CICS-MQ bridge. Any data required by the CICS application can be included in the request message:

- For DPL programs, the data required is the CICS communication area (COMMAREA) data used by the CICS application. If the CICS application does not require any COMMAREA data, the message data consists only of the name of the DPL program.
- For 3270 transactions, the data required comprises vectors describing the application data structures (ADSs) used by the CICS application.

The request message typically also includes the IBM MQ CICS information header (the MQCIH structure), which supplies control options for the CICS applications. If you are running a single DPL program with the default transaction code, and the program does not require any authorization, you do not need the MQCIH. In all other cases, and for all 3270 transactions, the MQCIH is required.

The CICS-MQ bridge takes the data that is produced as output CICS application send data back to the non-CICS application in an IBM MQ message that is sent to a reply queue:

- For DPL programs, the data sent back is the COMMAREA data produced as output by the CICS application.
- For 3270 transactions, the data sent back comprises vectors describing the application data structures (ADSs) produced as output by the CICS application. The non-CICS application must interpret and respond to these vectors.

For explanations of how the CICS-MQ bridge processes the IBM MQ messages to run single or multiple DPL programs or 3270 transactions, see How CICS DPL programs run under the CICS-WebSphere MQ bridge and How CICS 3270 transactions run under the CICS-WebSphere MQ bridge.

If you are planning to run a 3270 transaction, you might find it helpful to obtain a copy of CICS SupportPac CA1E CICS 3270 Bridge Passthrough to help you analyze the logic of the CICS transaction. The SupportPac enables you to display and capture the inbound and outbound data flows, to study how messages must be structured, and which values to insert into fields in the MQCIH and the vectors.

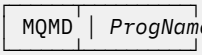
DPL message structure for the CICS-MQ bridge

These examples show the different structures that you can use for request messages from a non-CICS application to run CICS DPL programs through the CICS-MQ bridge.

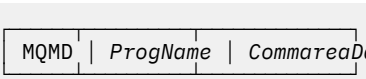
In the simplest case, the message data consists only of the name of a DPL program to be run. Follow this by COMMAREA data or with DFHREQUEST container data if you want to make data available to the DPL program (the user program) when it starts.

If you want to run more than one DPL program in a unit of work, or you prefer a specific transaction code (overwriting the default CKBP), or you require certain levels of authorization to run the DPL program, or utilise the channels and containers (by using the CKBC or equivalent transaction to run program DFHMQBP3), you must supply information in an MQCIH. The MQCIH must precede the DPL program name and any DPL program data that you send.

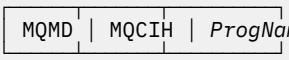
- Use this structure when your non-CICS application runs a single CICS DPL program using default processing options and does not send or receive any DPL program data:

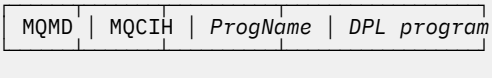
  The program specified by *ProgName* is called by CICS as a DPL program.
- Use this structure when your non-CICS application runs a single DPL program using default processing options and sends and receives COMMAREA data:

- Use this structure when your non-CICS application runs one or more DPL programs in a unit of work, or needs specific authorization to run the program, but does not send or receive any DPL program data:

- Use this structure when your non-CICS application runs one or more DPL programs in a unit of work, or needs specific authorization to run the program, and sends and receives COMMAREA data or sends DFHREQUEST container data, and receives DFHRESPONSE container data:

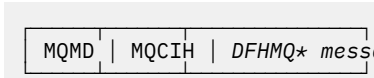
  

Note: When you send only a program name and no COMMAREA data to the CICS-MQ bridge, the program name must be 8 characters long. It must not be a name that is padded to the right with spaces; if it is the CICS-MQ bridge reports a COMMAREA negative length error. When you do send COMMAREA data, or DFHREQUEST container data you must pad the program name with spaces to the right, to give a total length of 8 characters.

Optionally, additional headers with format names beginning MQH, and containing standard link fields, can precede the MQCIH header. Such headers are returned unmodified in the output message because the CICS-MQ bridge makes no use of data in the headers.

Reply message structure

If a CICS-MQ bridge task running a DPL program ends abnormally, it returns a message to the reply queue with the following structure, whether or not the inbound message preceding the failure contains an MQCIH:



DFHMQ message* represents an error message that indicates the error type. The value of the MQCIH.Format field is set to MQFMT_STRING, so that the message can be properly converted if the final destination uses a different CCSID and encoding. The MQCIH also contains other fields that you can use to diagnose the problem.

Example: request message for a DPL program through the CICS-MQ bridge

This C-language code fragment shows how to call a CICS DPL program with COMMAREA data through the CICS-MQ bridge, by constructing a message buffer and including an IBM MQ CICS information header (MQCIH).

The DPL program that is called must conform to the DPL subset rules. See Application programming for CICS DPL for further details.

```
/* #defines                                                  */
#define   PGMNAME "DPLPGM"              /* DPL program name  */
#define   PGMNAMELEN 8
#define   CALEN 100                     /* Commarea length   */
.
/* Data declarations                                         */
MQMD     mqmd ;                /* Message descriptor        */
MQCIH    mqcih ;               /* CICS information header   */
MQCHAR * Commarea ;            /* Commarea pointer          */
MQCHAR * MsgBuffer ;           /* Message buffer pointer    */
.
/* allocate storage for the buffers                          */

Commarea  = malloc(CALEN * sizeof(MQCHAR)) ;
MsgBuffer = malloc(sizeof(MQCIH) + PGMNAMELEN + CALEN) ;
.
/* Initialize commarea with data                             */
.
/* Initialize fields in the MQMD as required, including:     */

memcpy(mqmd.MsgId, MQMI_NONE, sizeof(mqmd.MsgId)) ;
memcpy(mqmd.CorrelId, MQCI_NEW_SESSION, sizeof(mqmd.CorrelId)) ;

/* Initialize fields in the MQCIH as required                */
.
/* Copy the MQCIH to the start of the message buffer         */

memcpy(MsgBuffer, &mqcih, sizeof(MQCIH)) ;

/* Set 8 bytes after the MQCIH to spaces                     */

memset(MsgBuffer + sizeof(MQCIH), ' ', PGMNAMELEN) ;

/* Append the program name to the MQCIH. If it is less than  */
/* 8 characters, it is now padded to the right with spaces.  */

memcpy(MsgBuffer + sizeof(MQCIH), PGMNAME, PGMNAMELEN) ;

/* Append the commarea after the program name                */

memcpy(MsgBuffer + sizeof(MQCIH) + PGMNAMELEN, &Commarea, CALEN ) ;

/* The message buffer is now ready for the MQPUT             */
/* to the Bridge Request Queue.                              */
.
```

3270 transaction message structure for the CICS-MQ bridge

The examples in this section show the structures of request and reply messages when a non-CICS application runs a CICS 3270 transaction through the CICS-MQ bridge. CICS-MQ bridge vectors are used in the messages to represent the EXEC CICS commands in the transaction.

The CICS-MQ bridge can emulate a number of ways of starting a CICS transaction, including emulating a terminal user and issuing and receiving EXEC CICS commands. Without using IBM MQ, a CICS transaction can be started in several ways, including:

- A terminal user can enter the transaction name, followed (optionally) by data. The transaction can obtain any data that follows its identifier by issuing an EXEC CICS RECEIVE command when it starts.
- A preceding transaction at the terminal stops with EXEC CICS RETURN TRANSID(*transid*); the terminal sends a 3270 data stream and starts a new transaction. A transaction that is started in this way obtains the data in the 3270 data stream by issuing an EXEC CICS RECEIVE MAP or EXEC CICS RECEIVE command, depending on whether it uses BMS (Basic Mapping Support) mapping or terminal control.
- An application issues an EXEC CICS START command. The started transaction issues an EXEC CICS RETRIEVE command to retrieve any data that has been specified on the START command.

A transaction that has been started at a terminal can subsequently issue commands such as EXEC CICS CONVERSE, EXEC CICS SEND MAP, and EXEC CICS RECEIVE MAP in a conversation or pseudoconversation with a terminal user.

The CICS-MQ bridge can emulate any of these ways of starting CICS transactions. It can also emulate a terminal user sending and receiving screens of data from the transaction. These emulations are achieved by using CICS-MQ bridge vectors, which represent the EXEC CICS command being emulated and provide any data that is needed. The data needed by a CICS transaction accompanies inbound messages, and the data needed by a CICS-MQ bridge application accompanies outbound messages.

Vectors for 3270 transactions using the CICS-MQ bridge

The CICS-MQ bridge emulates all the functions of the CICS terminal API, including minimum function BMS. When you use the CICS-MQ bridge to run 3270 transactions, you use vectors to represent **EXEC CICS** commands in request and reply messages.

Vectors are identified in bridge messages by strings of numeric characters known as vector descriptors. Each vector descriptor is the CICS EIBFN value of the EXEC CICS command that it represents. For example, `0402` is the EIBFN value for **EXEC CICS RECEIVE** and also the vector descriptor of the RECEIVE vector. Vectors are further qualified by the letters `I` and `O` to show whether they are inbound (to the CICS-MQ bridge) or outbound (from the CICS-MQ bridge). Messages can contain several concatenated vectors.

An inbound message that is sent to the CICS-MQ bridge to run a 3270 transaction must include a vector structure after the MQCIH, unless you start a transaction with no data. Inbound messages can contain multiple RECEIVE MAP vectors in anticipation of future RECEIVE MAP requests from the CICS transaction. The application needs to know the flow of control in the transaction so that it can construct the message.

An outbound message can contain request vectors or reply vectors. These descriptions do not mean that they go to the request queue or the reply queue; all outbound messages go to the reply queue. A CICS transaction uses a request vector to request data from the non-CICS application that is acting as the virtual terminal. A CICS transaction uses a reply vector when it does not expect any data back. No such distinction is made for inbound vectors.

Outbound messages can contain several vectors; for example, as a result of successive **EXEC CICS SEND MAP** commands being issued by a transaction. The CICS transaction does not control whether the outbound message contains a single vector or multiple vectors. If the transaction issues a command that generates a request vector, the request vector is always the last one in the message.

The following vectors are available. To get the CICS command that each represents, prefix the vector name with EXEC CICS.

Inbound vectors:

- RECEIVE
- RECEIVE MAP
- CONVERSE
- RETRIEVE

Outbound reply vectors (no further data is required in the next inbound message):

- SEND
- SEND CONTROL
- SEND MAP
- SEND TEXT
- ISSUE ERASEAUP

Outbound request vectors (further data is required in the next inbound message):

- RECEIVE
- RECEIVE MAP
- CONVERSE

Each of these vectors is an architected structure followed by variable-length data. All the vectors have a common header, but their structures differ. For information about the structures, see Link3270 message formats .

The vector structure definitions are available in C-language header file `dfhbrmqh.h` and COBOL copybook DFHBRMQ0. Include them in any application that uses the bridge. These members are provided only with CICS Transaction Server on z/OS. If you want to create your application on another platform, copy them to that environment.

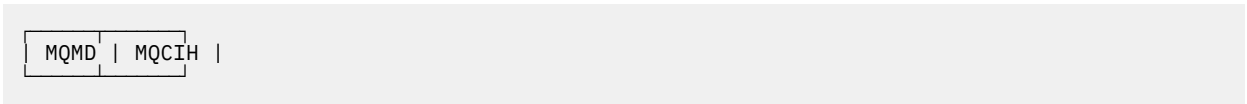
RETRIEVE vector

If the CICS 3270 transaction issues an **`EXEC CICS RETRIEVE … QUEUE(data-area)`** command to retrieve its start data, the non-CICS application must send a message to the CICS-MQ bridge with a RETRIEVE vector structure, which is defined in C as `brmq_retrieve`. The structure contains a character field of length 8 bytes in which the non-CICS application must specify the name of the temporary storage queue that contains the data to be retrieved. A message containing a RETRIEVE vector is always the first in an exchange representing a conversation or pseudoconversation.

Structures of 3270 request messages (inbound) using the CICS-MQ bridge

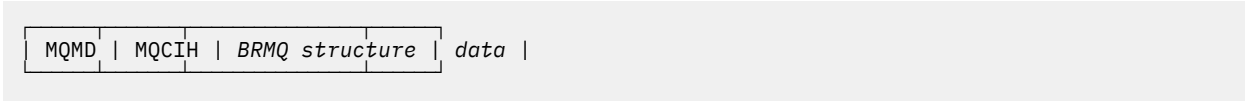
These examples show the possible structures of request messages sent by non-CICS applications to 3270 transactions using the CICS-MQ bridge, known as inbound messages.

- Use this structure when your non-CICS application calls a CICS transaction without any data:



  Set the field MQCIH.TransactionId to the name of the transaction that you want to start. Set the other fields in MQCIH to values that are appropriate for the application.

- Use this structure when your non-CICS application runs a transaction that will issue an EXEC CICS command that expects data to be available:



BRMQ structure represents any of the inbound vector structures RECEIVE, RECEIVE MAP, CONVERSE, or RETRIEVE. Note that the BRMQ structure itself consists of a header followed by vectors, and these vectors can contain data.

- Use this structure for request messages that have zero length data:

```
┌──────┬──────┬──────────────────┐
| MQMD | MQCIH | BRMQ structure |
└──────┴──────┴──────────────────┘
```

For example, an inbound RECEIVE MAP vector can represent an action where the user has only pressed a PF key. In this case, a field in the BRMQ structure specifies which AID key has been used, but no data follows the BRMQ structure.

- Optionally, additional headers with format names beginning MQH, and containing standard link fields, can precede the MQCIH header. Such headers are returned unmodified in the output message because the CICS-MQ bridge makes no use of data in the headers. Use this structure for request messages that have headers before the MQCIH:

```
┌──────┬─────────┬──────┐
| MQMD | MQRFH2 | MQCIH |
└──────┴─────────┴──────┘
```

Structures of 3270 reply messages (outbound) using the CICS-MQ bridge

These examples show the possible structures of reply messages sent by the CICS-MQ bridge on behalf of 3270 transactions to non-CICS applications, known as outbound messages.

Outbound messages from the CICS-MQ bridge have one of three structures, depending on whether an error occurred. Although only a single vector is shown in each of these examples, messages can contain several concatenated vectors, except when an error occurs.

- This structure is used when CICS-MQ bridge processing concludes normally, no errors were detected, and no data is to be returned to the non-CICS application:

```
-------------------------------
| MQMD | MQCIH |
BRMQ structure
|
-------------------------------
```

Even a CICS application abend is regarded as normal completion by the CICS-MQ bridge. The abend code issued by the CICS application is given in the MQCIH.

- This structure is used when CICS-MQ bridge processing concludes normally, no errors were detected, and data is to be returned to the non-CICS application:

```
---------------------------------------
| MQMD | MQCIH |
BRMQ structure
|
data
|
---------------------------------------
```

BRMQ structure represents any of the architected outbound reply or request vector structures.

- This structure is used when CICS-MQ bridge processing concludes abnormally, after an error was detected by the CICS-MQ bridge monitor:

```
-------------------------------
| MQMD | MQCIH |
DFHMQ* message
|
```

```
------------------------------
```

DFHMQ message* represents an error message that indicates the error type. The value of field MQCIH.Format is set to MQFMT_STRING, to ensure that the message can be properly converted if the final destination uses a different CCSID and encoding. The MQCIH also contains other fields that you can use to diagnose the problem.

For CICS transactions that issue explicit syncpoint or rollback requests, the CICS-MQ bridge also generates an additional message on the reply queue, showing the result of the syncpoint in the TaskEndStatus field in the MQCIH. This extra message is sent with an MQMD MsgType of MQMT_DATAGRAM. No input from the application is expected and the additional message is followed by the normal task end message.

Example: request message to issue CEMT INQUIRE TASK through the CICS-MQ bridge

This example shows how an application can use the CICS-MQ bridge to start a 3270 transaction, in this case CEMT, that would normally be started by entering its identifier and some command line arguments at a CICS terminal.

When the CEMT task starts, it issues an EXEC CICS RECEIVE command to receive any command-line arguments that follow its identifier. The non-CICS application that emulates the command line invocation must therefore start CEMT with a RECEIVE vector that contains appropriate values in the vector structure, and also include the command-line values.

Constructing the request message

The following C language code fragment shows how the request message can be constructed. The C language header file `dfhbrmqh.h` is in the CICS SAMPLIB library.

```
/* #includes                                                  */
#include cmqc.h                 /* IBM MQ header             */
#include dfhbrmqh.h             /* Vector structures         */
⋮
/* #defines                                                   */
#define CMDSTRING   "CEMT I TASK"      /* Command string     */
#define RCV_VECTOR  "0402"             /* Vector descriptor  */
#define INBOUND     "I   "             /* Inbound type       */
#define VERSION     "0000"             /* Vector version     */
#define YES         "Y   "             /* YES indicator      */
#define NO          "N   "             /* NO indicator       */
⋮
/* Data declarations                                          */
/* AID indicator value                                        */
const char          AID[ 4 ] = { 0x7d, ' ',' ',' ' } ;
MQMD                mqmd ;      /* Message descriptor         */
MQCIH mqcih = {MQCIH_DEFAULT} ; /* CICS information header    */
brmq_vector_header  brvh ;      /* Standard vector header     */
brmq_receive        brrcv ;     /* RECEIVE vector structure   */
MQCHAR            * MsgBuffer ; /* Message buffer pointer     */
⋮
```

The outbound message that is returned to the reply queue contains a SEND reply vector with data in terminal control format. Your application must interpret this format to analyze the data that it receives.

Allocating storage for the message buffer

```
/* allocate storage for the message buffer. Note that the RECEIVE     */
/* vector structure includes space for the standard vector header.    */

MsgBuffer = malloc(sizeof(MQCIH) + sizeof(brrcv) + strlen(CMDSTRING)) ;
⋮
```

Setting up the MQMD

```
memcpy(mqmd.Format, MQFMT_CICS, sizeof(MQFMT_CICS));
memcpy(mqmd.MsgId, MQMI_NONE, sizeof(MQMI_NONE));
memcpy(mqmd.CorrelId, MQCI_NEW_SESSION, sizeof(MQCI_NEW_SESSION));
```

```
mqmd.MsgType = MQMT_REQUEST;
strcpy(mqmd.ReplyToQueue, "MyReplyQueue");
```

Setting up the MQCIH

```
mqcih.LinkType = MQCLT_TRANSACTION ;
mqcih.GetWaitInterval = 1000 ;                  /* one second             */
mqcih.FacilityKeepTime = 10000 ;                /* |= 0 says return token */
memcpy(mqcih.Facility, MQCFAC_NONE, sizeof(MQCFAC_NONE)) ;
strncpy(mqcih.TransactionId, "CEMT", strlen("CEMT"));
strncpy(mqcih.FacilityLike, "    ", strlen("    "));
mqcih.UOWControl = MQCUOWC_FIRST;
memcpy(mqcih.AttentionId,AID,sizeof(mqcih.AttentionId);   /* enter pressed */
```

Setting up the BRMQ

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

Building the message

```
/* Copy the MQCIH to the start of the message buffer         */
memcpy(MsgBuffer, &mqcih, sizeof(MQCIH)) ;
/* Append the RECEIVE vector to the CIH                      */
memcpy(MsgBuffer + sizeof(MQCIH), brrcv, sizeof(brrcv)) ;
/* Overlay the standard vector header on the RECEIVE vector  */
memcpy(MsgBuffer + sizeof(MQCIH), brvh, sizeof(brvh)) ;
/* Append the command string to the vector structure         */
strncpy(MsgBuffer + sizeof(MQCIH) + sizeof(brrcv), CMDSTRING, strlen(CMDSTRING)) ;
/* the message is now ready for the MQPUT with length of     */
/* sizeof(MQCIH) + sizeof(brrcv) + strlen(CMDSTRING)         */
```

Fields that you must set in the MQMD and MQCIH structures for the CICS-MQ bridge

Your non-CICS application must set a number of fields in the MQMD and MQCIH structures in request messages for the CICS-MQ bridge.

MQMD fields for CICS-MQ bridge messages

Fields in the MQMD that can affect the operation of the CICS-MQ bridge must be initialized in your application program.

MQMD.CorrelId

For MQPUT commands to the request queue, set the value to MQCI_NEW_SESSION in the first or only message in a unit of work. On subsequent messages in the unit of work, set the value to the MQMD.MsgId that IBM MQ set in your message descriptor when you put your first message to the request queue.

For MQGET commands to the reply queue, use the value of MQMD.MsgId that IBM MQ set in your message descriptor when you put your most recent message to the request queue, or specify MQCI_NONE. See also “MsgId, CorrelId, and UOWControl fields for DPL programs” on page 90 and “MsgId, CorrelId, and UOWControl fields for 3270 transactions” on page 91.

MQMD.Expiry

Set a message expiry time based on how long you want your application to wait for a reply. Set the MQCIH flags to propagate the remaining expiry time to the reply message.

MQMD.Format

For DPL programs, set the value to MQCICS if you include an MQCIH in the message that you send to the bridge request queue; otherwise, set it to the format of the data following.

For 3270 transactions, the value must be MQFMT_CICS.

MQMD.MsgId

For MQPUT commands to the request queue, set MsgId to a unique value for the unit of work or to MQMI_NONE.

For MQGET commands to the reply queue, use the value of MQMD.MsgId that IBM MQ set in your message descriptor when you put your first message to the request queue. See also "MsgId, CorrelId, and UOWControl fields for DPL programs" on page 90 and "MsgId, CorrelId, and UOWControl fields for 3270 transactions" on page 91

MQMD.Persistence

Messages put to the request queue can be persistent or nonpersistent. Reply messages have the same persistence as the request messages.

MQMD.ReplyToQ

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

MQMD.UserIdentifier

This field is used only when the bridge monitor is running with authorization levels of IDENTIFY, VERIFY_UOW, or VERIFY_ALL. If you use any of these, set the value to the user ID that is checked for access to the CICS DPL program. If you use this field with one of the VERIFY_* options, you must also initialize the MQCIH.Authenticator field. Set it to the value of the password or PassTicket associated with the user ID.

Note: When you use an authorization level of IDENTIFY, VERIFY_UOW, or VERIFY_ALL, you must add the value MQOO_SET_IDENTITY_CONTEXT to the open options when you open the bridge request queue, and also add the value MQPMO_SET_IDENTITY_CONTEXT to the put message options when you send a message to the queue.

MQCIH fields for DPL program request messages

When you send a request message to run a CICS DPL program using the CICS-MQ bridge, you must initialize these key input fields in your application program.

MQCIH.Authenticator

This field applies only if you are using an authorization level of VERIFY_UOW or VERIFY_ALL. Set the value to the password or PassTicket that is to be associated with the user ID in the MQMD.UserIdentifier field. Together, the values are used by the external security manager to determine whether the user is authorized to link to the DPL program. If you are using PassTickets, the applid used for generating the PassTicket must be the same as the PASSTKTA keyword values used when starting the bridge monitor.

MQCIH.Flags

- Set to `MQCIH_PASS_EXPIRATION` to pass the remaining expiry time to the reply message.
- Set to `MQCIH_REPLY_WITHOUT_NULLS` to remove trailing null characters ('00'X) from the reply message.
- Set to `MQCIH_SYNC_ON_RETURN` to specify the SYNCONRETURN option on the EXEC CICS LINK command.

You can combine the values by adding them together.

MQCIH.Format

Specifies the format of the data following the MQCIH structure. If the data is character data, use `MQFMT_STRING` ; if no conversion is needed, use `MQFMT_NONE`.

MQCIH.GetWaitInterval
If you allow this field to default, the bridge task GET WAIT interval for messages in a unit of work is the value specified on the WAIT parameter when the bridge monitor was started. If you also allow the WAIT parameter to default, the GET WAIT interval is unlimited.

MQCIH.LinkType
Specify `MQCLT_PROGRAM` if you are using the CICS DPL bridge.

MQCIH.OutputDataLength
This field applies only to the CICS DPL bridge and sets the length of data returned by the program. MQCIH.OutputDataLength is ignored for DPL requests that use the channel and container interface; for these requests the output (that is, the response) length is the size of the DFHRESPONSE container.

MQCIH.RemoteSysId
Leave this field blank unless you need the request to be processed by a specific CICS system.

MQCIH.ReplyToFormat
Set this to `MQFMT_NONE` (the default value) if your application and the bridge are running in the same CCSID and encoding environment. Otherwise, set the value to the format of the COMMAREA data returned. MQCIH.replytoformat is ignored for DPL requests that use the channel and container interface. For these requests the reply-to format is set based on the content of the DFHRESPONSE container; if the content is character data, then the reply-to format is `MQFMT_STRING` ; if the content is binary data then the reply-to format is `MQFMT_NONE`.

MQCIH.TransactionId
Specify the transaction code that you want to run the CICS DPL bridge program under. You can specify one of the supplied transactions:

- CKBP, which runs program DFHMQBP0 using a COMMAREA
- CKBC, which runs program DFHMQBP3 using channels and containers
- Your own transaction code to run DFHMQBP0 or DFHMQBP3

If you leave the field blank (four spaces) this defaults to the supplied transaction code, CKBP.

MQCIH.UOWControl
This field controls the unit of work processing performed by the bridge. See "MsgId, CorrelId, and UOWControl fields for DPL programs" on page 90 for more information.

MQCIH fields for 3270 transaction request messages

When you send a request message to run a 3270 transaction using the CICS-MQ bridge, you must initialize these key input fields in your application program.

MQCIH.ADSDescriptor
This field applies to transactions that use BMS SEND MAP and RECEIVE MAP calls. In this case, and if the application that is sending CICS-MQ bridge request messages is on a workstation, set this value to MQCADSD_SEND + MQCADSD_RECV + MQCADSD_MSGFORMAT. This setting ensures that the vectors in the request and reply messages are correctly converted between the different CCSID and encoding schemes of the workstation and the mainframe.

MQCIH.AttentionId
Set this field to a value representing the AID key expected by the transaction, if any; otherwise, accept the default value of four spaces, which appear to the CICS transaction as the ENTER AID key.

The inbound RECEIVE, RECEIVE MAP, and CONVERSE vectors also have fields in which you can specify AID values. The value in the MQCIH is the value to which EIBAID is set when the application is started. It represents the function key used to start the transaction. The value in the inbound vector is the value used when the data is entered. For example, this would be the value of EIBAID after the EXEC CICS RECEIVE MAP instruction.

Note:

1. For conversational transactions the initial MQCIH value and the value on the vector have separate values.

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

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

MQCIH.Authenticator
This field applies only if you are using an authorization level of VERIFY_UOW or VERIFY_ALL. Set the value to the password or PassTicket that is to be associated with the user ID in the MQMD.UserIdentifier field. Together, the values are used by the external security manager to determine whether the user is authorized to start the 3270 transaction. If you are using a PassTicket, the applid used for generating the PassTicket must be the same as the PASSTKTA keyword values used when starting the bridge monitor.

MQCIH.ConversationalTask
Specify the value MQCCT_NO if all the input vectors needed for this CICS transaction are supplied in the input message. Specify MQCCT_YES if multiple messages can be used to supply input vectors for the transaction

MQCIH.Facility
Specify the value MQCFAC_NONE in the first message in a pseudoconversation, and also set the MQCIH.FacilityKeepTime field to a nonzero value. The CICS-MQ bridge returns a facility token in the first message. You must use this value in all subsequent inbound messages in the pseudoconversation.

MQCIH.FacilityKeepTime
If you are sending more than a single message in a pseudoconversation, set this field to a nonzero value (in seconds) in the first message for the CICS-MQ bridge to return a facility token. Successive transactions in a pseudoconversation must use the same facility token after it has been set in this way, ensuring that associated terminal areas, for example the TCTUA, are preserved for the period of the pseudoconversation.

MQCIH.FacilityLike
Either use the default value of four spaces, or specify the name of an installed terminal. You can find the names of installed terminals by entering the CICS command CEMT I TASK or CEMT I TERM at a CICS terminal.

MQCIH.Flags
Set the value to MQCIH_PASS_EXPIRATION to pass the remaining expiry time to the reply message.

MQCIH.Format
Set the value to DFHMQDCI (or CSCQBDCI). This value informs the CICS-MQ bridge that any data following the MQCIH is inbound to the bridge and might need to be converted. The bridge sets the value of MQCIH.Format in the outbound message, which is returned to the reply queue to DFHMQDCO (or CSCQBDCO).

MQCIH.GetWaitInterval
If you allow this field to default, the bridge task GET WAIT interval for messages within a unit of work is the value specified on the WAIT parameter when the CICS-MQ bridge monitor was started. If you also allow the WAIT parameter to default, the GET WAIT interval is unlimited.

MQCIH.LinkType
Specify MQCLT_TRANSACTION for 3270 transactions.

MQCIH.RemoteSysid
Set this field to blank on the first message of a pseudoconversation unless you require the request to be processed on a specific CICS system. In subsequent messages in the pseudoconversation, set this field to the value returned in the first reply message.

Note: In earlier versions of the CICS-MQ bridge the RemoteSysId field was not used; however, it is important that it is now passed thorough the conversation to enable the use of the facility for multiple CICS-MQ bridge monitors. The typical style of CICS programming is pseudoconversational; that is, a series of independent transactions that are linked together to form a complete application. When using the CICS-MQ bridge, the link between the transactions of a pseudoconversation is maintained

by passing the Facility token and RemoteSysId value returned by the first transaction of the sequence into subsequent messages of the conversation.

MQCIH.StartCode
When you start a 3270 transaction you must change the value of this field from the default value of MQCSC_NONE. The value you use depends on the nature of the transaction. Use a value of MQCSC_START if the transaction is started by an EXEC CICS START command without data, and it does not issue an EXEC CICS RETRIEVE command. Use a value of MQCSC_STARTDATA if the transaction is started by an EXEC CICS START command with data, and it issues an EXEC CICS RETRIEVE command. Otherwise, use a value of MQCSC_TERMINPUT.

MQCIH.TransactionId
This value is the transaction identifier of the 3270 transaction to be run by the bridge task. The first message must specify the first transaction to be started. Set this field in subsequent messages to the value of MQCIH.NextTransactionId that is returned in the preceding reply message.

MQCIH.UOWControl
This value controls the unit-of-work processing performed by the CICS-MQ bridge. See also "MsgId, CorrelId, and UOWControl fields for 3270 transactions" on page 91.

MsgId, CorrelId, and UOWControl fields for DPL programs

If your non-CICS application is using the CICS-MQ bridge to run a single DPL program, set the value of the MQCIH.UOWControl field to MQCUOWC_ONLY. However, if your application is sending and receiving multiple messages, you must set alternative options to handle units of work correctly.

To run multiple user programs within a unit of work, set the value of the MQCIH.UOWControl field as follows:

- MQCUOWC_FIRST in the first request
- MQCUOWC_MIDDLE in any intermediate requests
- MQCUOWC_LAST in the last request

Your application can send multiple request messages in a unit of work before receiving any reply messages. At any time after the first message, you can stop the unit of work by sending a message with the MQCIH.UOWControl field set to MQCUOWC_COMMIT or MQCUOWC_BACKOUT.

The first message must specify MQMD.CorrelId = MQCI_NEW_SESSION and subsequent messages must set the MQMD.CorrelId field to the message id of the first message. When you are running more than one user DPL program using the CICS-MQ bridge, the MsgId field in the request message is set by the queue manager (to M1 in the example shown here), and subsequently copied to the CorrelId field.

The following diagram summarizes the values to use and expect in key fields in the MQMD and MQCIH for typical CICS DPL programs that you run using the CICS-MQ bridge.

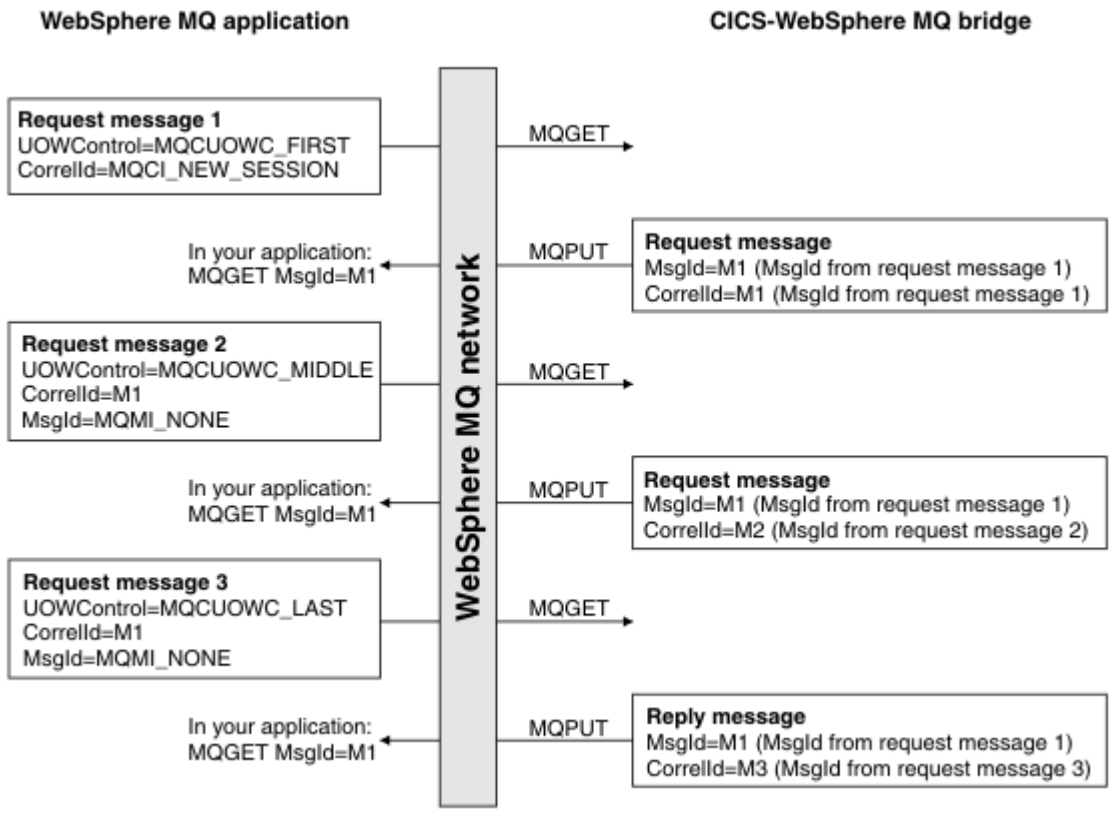


Figure 10. Setting of key fields for many CICS user programs in a unit of work viewed from the perspective of the bridge

MsgId, CorrelId, and UOWControl fields for 3270 transactions

Typically, a series of CICS 3270 transactions are linked together in a pseudoconversation to form a complete application. When your non-CICS application is using the CICS-MQ bridge to run 3270 transactions, you can use the MQCIH.UOWControl field to control how the transactions are grouped into units of work.

Note: As well as setting the MQCIH.UOWControl and MQMD.Correlid fields, it is important that you use the MQCIH.Facility field and the MQCIH.RemoteSysId field to pass the Facility token and RemoteSysId value returned by the first transaction of the sequence into subsequent messages of the conversation. For more information, see "MQCIH fields for 3270 transaction request messages" on page 88.

To group multiple 3270 transactions into a single CICS-MQ bridge session, set the value of the MQCIH.UOWControl field as follows:

- Use MQCUOWC_FIRST for the first message of the bridge session.
- Use MQCUOWC_MIDDLE in subsequent messages of the bridge session, whether they supply additional data to a transaction or start a new transaction.
- Use MQCUOW_LAST to indicate a proposed end of session. See "Ending a session" on page 92.

Set the value of the MQMD.Correlid field as follows:

- Use MQCI_NEW_SESSION for the first message of the bridge session.
- In all subsequent messages for the bridge session, use the message identifier generated for the first message of the transaction.

Set the value of the MQCIH.Facility field as follows:

- Specify MQCFAC_NONE in the first message of the bridge session, and also set the MQCIH.FacilityKeepTime field to a nonzero value.

- In all subsequent messages for the bridge session, use the facility token that is returned in the first reply message.

Set the value of the MQCIH.RemoteSysid field as follows:

- Use blanks for the first message of the bridge session, unless you require the request to be processed on a specific CICS system, in which case specify the CICS system.
- In all subsequent messages for the bridge session, use the value that is returned in the first reply message.

Ending a session

When you set the value of the MQCIH.UOWControl field to MQCUOW_LAST to indicate a proposed end of session, your application must check the message type of the reply message to determine what further actions to take.

- If the reply message type, shown in the Msgtype field, is MQMT_REPLY, meaning that the CICS transaction ends with no more requests for data, the session is ended.
- If the reply message type is MQMT_REQUEST, meaning that the CICS transaction requests more data, you can supply the data and send the next message with the MQCIH.UOWControl field set to either MQCUOWC_LAST or MQCUOWC_CONTINUE.
- If the reply message type is MQMT_REQUEST but you do not want to continue the session, send a message with the MQCIH.UOWControl field set to MQCUOWC_COMMIT to end the session.

If you need to end a running 3270 transaction, set the MQCIHCancelCode field to a four-character abend code.

If you want to end a session between transactions, set the MQCIH.UOWControl field to MQCUOWC_COMMIT.

The following diagram summarizes the values to use and expect in key fields in the MQMD and MQCIH when interacting with a typical, conversational, 3270 transaction.

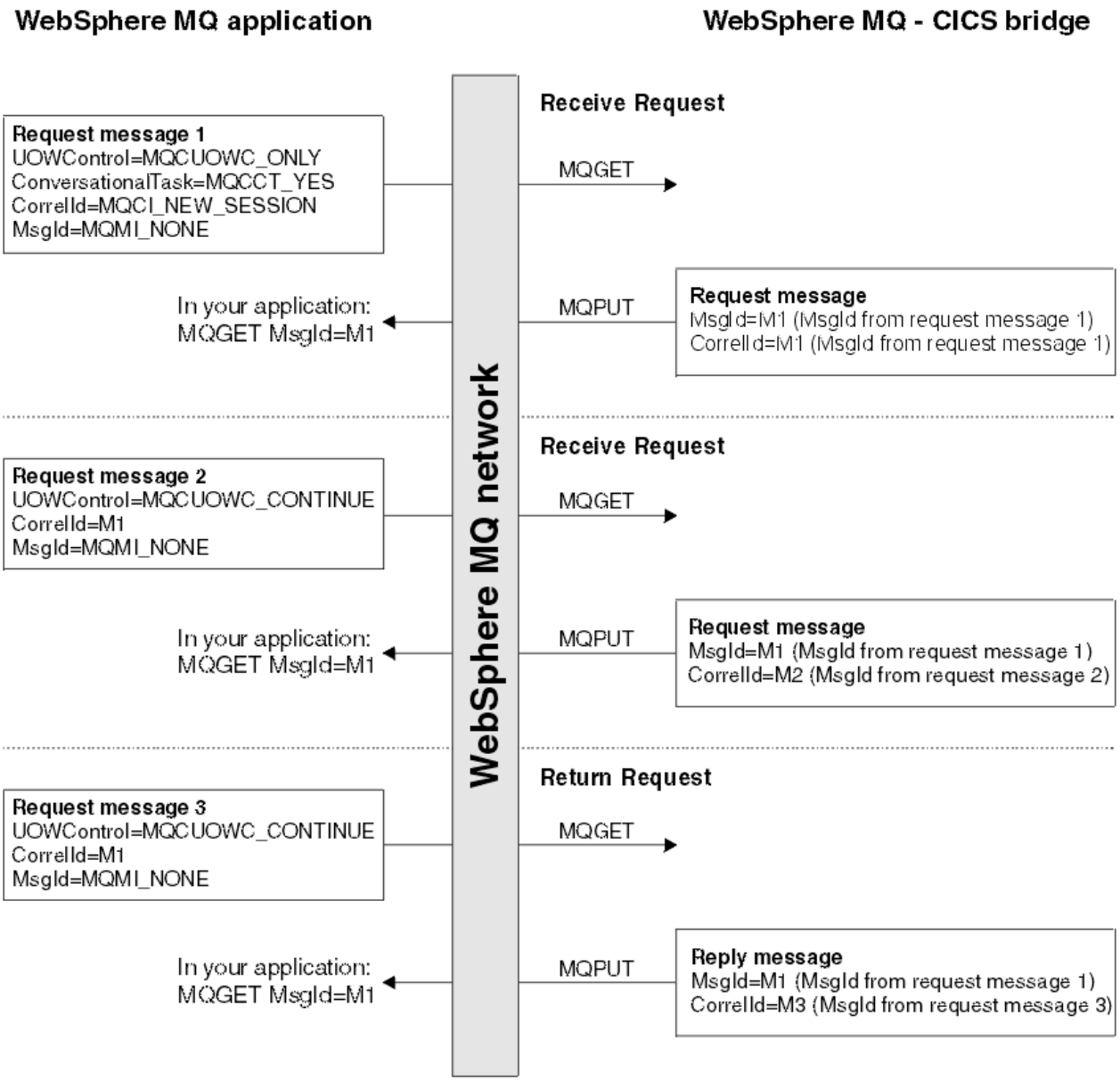


Figure 11. Setting of key fields: IBM MQ - conversational 3270 transaction viewed from the perspective of the bridge

Data conversion for the CICS-MQ bridge in the distributed environment

CICS DPL programs and 3270 transactions can be driven through the CICS-MQ bridge when the client application resides on a workstation. The main consideration when programming for the distributed environment is data conversion between the different encoding schemes and CCSID values of the workstation and z/OS.

Conversion is carried out by two different routines, one for the MQCIH structure and another for the data or vectors supplied in the message. If you have an MQCIH in the message then you must specify `MQFMT_CICS` in the format field of the header immediately before the MQCIH. If there are no headers before the MQCIH then you must specify `MQFMT_CICS` in the MQMD.Format field. Data and vector conversion requires a little more consideration.

If you are driving a DPL program that does not receive or return COMMAREA or container data, or if the COMMAREA or container data is purely character data, you can achieve data conversion by specifying `MQFMT_STRING` as the format value. If the COMMAREA or container data is purely binary, data conversion

is unnecessary, so you then specify MQFMT_NONE as the format value. If your COMMAREA or container data is a mix of character and binary data, you must write your own conversion routine and specify an appropriate format value. See IBM MQ product documentation for information about writing conversion routines.

If the message includes an MQCIH, specify the format value in MQCIH.Format. If the message does not include an MQCIH but does include other headers, then specify the format value in the last header. If the message does not include any headers then specify the format value in the MQMD.

If you are driving a DPL program that generates response data in the COMMAREA and you provided a reply-to queue, then you must specify the format value for this response data in your request message in MQCIH.ReplyToFormat. If you are driving a DPL program that generates response data in a container then you do not need to specify MQCIH.ReplyToFormat because the container identifies itself as character or binary data.

If you are driving a 3270 transaction, to convert the SEND MAP and RECEIVE MAP vectors:

- Make sure that you assemble your maps by specifying DSECT=ADSDL in your DFHMSD macro. If you do not have the original mapset definition, re-create the map using the CICS DFHBMSUP utility.
- Specify a value of MQCADSD_SEND+MQCADSD_MSGFORMAT in field MQCIH.ADSDescriptor. If you are using an ADSD or ADSDL to build your RECEIVE MAP ADS, add in the value MQCADSD_RECV for this field.
- Specify a value of DFHMQDCI or CSCQBDCI in field MQCIH.Format on every inbound message. If the input format is CSQCBDCI the output format is CSQCBDCO. If the input format is DFHMQDCI the output format is DFHMQCDO.
- Ensure that CONVERT=YES is specified on the channel between z/OS and the workstation, and that the data conversion program DFHMQDCI is installed in the channel initiator.

No support is provided for conversion between workstation and mainframe formats of vectors other than SEND MAP (outbound) and RECEIVE MAP (both inbound and outbound).

When reply messages from 3270 transactions are sent to a non-z/OS system, the sender channel must specify the CONVERT(YES) option.

Running Basic Mapping Support (BMS) applications using the CICS-MQ bridge

If a CICS 3270 transaction uses BMS maps, the non-CICS application can use the copybooks created during map assembly to help interpret data in the vectors. If you do not have access to the copybooks, the application can analyze the data indirectly through the use of an application data structure (ADS) descriptor.

CICS Basic Mapping Support (BMS) provides a way for CICS applications to support a number of different terminal types. When the application issues an EXEC CICS SEND MAP command, BMS merges terminal-specific control data with the application data to produce a 3270 data stream that can be displayed at the terminal. When the application issues an EXEC CICS RECEIVE MAP command, application data is extracted from an inbound 3270 data stream and returned to the application.

A BMS map for a CICS application is created by assembling a set of BMS macros that define the characteristics of fields that are required for the display. One of the outputs from map assembly is a copybook that maps the display fields to an ADS. The CICS application must include the copybook in its data definitions so that it can address the fields in the map symbolically. The application data in a SEND MAP, and expected by a RECEIVE MAP, is mapped directly to the ADS in the copybook.

When the transaction runs under the CICS bridge, EXEC CICS SEND MAP and EXEC CICS RECEIVE MAP commands generate SEND MAP and RECEIVE MAP vectors in outbound messages. Instead of a 3270 data stream, these vectors contain ADSs equivalent to those used by the CICS application to address fields in the map.

The format of the ADS is unique for each map. It is described by a copybook created as part of map generation. Without this copybook it is difficult to interpret the data. Typically, non-CICS applications that

use the IBM MQ bridge include the BMS copybooks so that they can create RECEIVE MAP data and interpret SEND MAP data. However, you can write an application without the specific BMS copybooks. The format of the data is described by a structure known as the ADS descriptor (ADSD). The ADSD is added to the end of a SEND MAP vector, and it describes the format of the ADS in the vector. The ADSD contents include the names, positions, and lengths of the fields in the ADS. An ADSD can also be sent on a RECEIVE MAP request. You can use this information in conversational applications to tell the non-CICS application the structure of the ADS requested by the CICS application. The non-CICS application can then build a RECEIVE MAP vector with this ADS, and send it as a new request.

As an application programmer, you can choose whether you want to interpret vector data in messages using the ADS, the ADSD, or the ADSDL (long form of the application data structure descriptor). To interpret the ADS directly, include the copybook from map assembly in your CICS-MQ bridge application. If you want to include it, but you do not have access to the copybook or map, re-create the map with the CICS utility DFHBMSUP.

To interpret the ADS indirectly through the ADSD or ADSDL, for example if you are creating a generic application that will handle all maps, you do not have to include the copybook in your bridge application. Instead, you send control information to the bridge that tells it to include the ADSD or ADSDL in outbound SEND MAP and RECEIVE MAP request vectors as required.

If your application must run in the distributed environment, include an ADSDL in outbound SEND MAP vectors. IBM MQ can then convert data in the outbound message.

You can specify any of the following options by setting appropriate values in field MQCIH.ADSDescriptor in inbound messages:

- To include an ADSD (short form of the application data structure descriptor) with the SEND MAP vector, set:

  ```
  MQCIH.ADSDescriptor = MQCADSD_SEND
  ```

  With this setting, you also get the short form of the ADS (application data structure) included in the SEND MAP vector.
- To include an ADSD with the RECEIVE MAP vector, set:

  ```
  MQCIH.ADSDescriptor = MQCADSD_RECV
  ```

  The ADS is never present in an outbound RECEIVE MAP request vector.
- To include an ADSDL in the SEND MAP or RECEIVE MAP vector, set:

  ```
  MQCIH.ADSDescriptor = MQCADSD_MSGFORMAT
  ```

  With this setting, you also get the long form of the ADS included in the SEND MAP vector.
- To not include an ADS descriptor in the SEND MAP or RECEIVE MAP vector set:

  ```
  MQCIH.ADSDescriptor = MQCADSD_NONE
  ```

  This setting is the default. With this setting, you get the short form of the ADS included in the SEND MAP vector.

You can add together MQCADSD_* values to include the long form of the application data structure descriptor in both SEND MAP and RECEIVE MAP vectors:

```
MQCIH.ADSDescriptor = MQCADSD_SEND + MQCADSD_RECV + MQCADSD_MSGFORMAT
```

In this case, the SEND MAP vector also includes an ADS in long form.

Interpreting SEND MAP vectors for the CICS-MQ bridge

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

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

1. Get the message containing the SEND MAP vector from the bridge reply queue into a message buffer.
2. Locate the start of the outbound SEND MAP vector in the message buffer. This vector is appended to the MQCIH, and so is at an offset equal to the length of the MQCIH from the start of the message buffer. You can use the following code fragment as a model:

```
/* #includes                                                  */
#include cmqc.h                 /* IBM MQ header             */
#include dfhbrmqh.h             /* Vector structures         */
⋮
/* #defines                                                   */
⋮
MQCHAR        * MsgBuffer ;     /* Message buffer pointer    */
brmq_send_map * pVector   ;     /* Vector pointer            */
⋮
/* Get message from reply queue                               */
⋮
/* Set the vector pointer to the start of the vector         */
pVector = MsgBuffer + ((MQCIH *) MsgBuffer)->StrucLength ;
```

3. Identify the starting addresses of the application data structure (ADS) and the application data structure descriptor (ADSD or ADSDL) from the SEND MAP vector.

   The following diagram shows the structure of an outbound SEND MAP vector, assuming that you have set a pointer called `pVector` to address the start of the `brmq_send_map` vector, as in the previous code fragment:

```
|--------------------------x'vvvvvvvv'------------------------->|
|---------------------x'zzzzzzzz'------------------->|
|----------------x'xxxxxxxx'------------>|
|---------sizeof(brmq_send_map)---------->|
                                    --> x'wwwwwwww' <--
      1804 0                                    --> x'yyyyyyyy' <--
 --------------  ...  -------------------------...----------...-----
|vvvv|FFFF|D444|     |wwww|xxxx|yyyy|zzzz| ADS       | ADSD or   |
|vvvv|1804|6000|     |wwww|xxxx|yyyy|zzzz|           | ADSDL     |
 --------------  ...  -----------------------...----------...-----
↑                    ↑    ↑    ↑    ↑
pVector              |    |    |    pVector->brmq_sm_adsd_offset
                     |    |    pVector->brmq_sm_adsd_len
                     |    pVector->brmq_sm_data_offset
                    pVector->brmq_sm_data_len
```

   Values in the diagram shown like this:

```
ABCD
1234
```

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

   Fields pVector->brmq_sm_data_offset and pVector->brmq_sm_data_len give the offset and length of the ADS, and fields pVector->brmq_sm_adsd_offset and pVector->brmq_sm_adsd_len give the offset and length of the ADSD or ADSDL.

   Fields brmq_sm_adsd_offset and brmq_sm_adsd_len are both set to zero if no ADSD or ADSDL is included in the message.

4. Identify the fields in the ADSD or ADSDL.

   The ADSD and ADSDL are both mapped to structures that are defined in header file `dfhbrarh.h` , which is distributed in CICS library `<hlq>.SDFHC370` . You can examine the structure definitions there to see how the fields are laid out. The fields of the ADSD are also described in Link3270 ADS descriptor .

To compile your bridge application on a workstation, copy file `dfhbrarh.h` to that environment.

Both the ADSD and the ADSDL are represented by two types of structure. The first structure is the descriptor, which occurs only once at the start of the ADSD or ADSDL. These types are defined as follows:

ads_descriptor
Descriptor for the ADSD (short form)

ads_long_descriptor
Descriptor for the ADSDL (long form)

The second structure is the field descriptor, which is repeated once for each field in the map. These types are defined as follows:

ads_field_descriptor
Field descriptor for the ADSD (short form)

ads_long_field_descriptor
Field descriptor for the ADSDL (long form)

This structure can be shown diagrammatically for the ADSDL and the ADSD:

The ADSDL:

```
 ------------------------------------------------------ ...
| ADS Descriptor | field descriptor | field descriptor |
 ------------------------------------------------------ ...
↑                ↑                  ↑
|                |                  ads_long_field_descriptor
|                ads_long_field_descriptor
ads_long_descriptor
```

The ADSD:

```
 ------------------------------------------------------ ...
| ADS Descriptor | field descriptor | field descriptor |
 ------------------------------------------------------ ...
↑                ↑                  ↑
|                |                  ads_field_descriptor
|                ads_field_descriptor
ads_descriptor
```

Fields adsd_field_count and adsdl_field_count in the descriptors identify the number of field descriptors in the ADSD and ADSDL.

You can use the following code fragment to set pointers to the start of the ADSD or ADSDL structures and process the field descriptors sequentially. It is assumed that `pVector` already addresses the start of the `brmq_send_map` vector, and that you have an MQCIH structure named `mqcih` that contains the CIH from the inbound message.

```
/* #includes                                                  */
#include cmqc.h                 /* IBM MQ header             */
#include dfhbrmqh.h             /* Vector structures         */
#include dfhbrarh.h             /* ADSD structures           */
⋮
/* Ptr to ADSD descriptor                                     */
ads_descriptor            * pADSD_D ;
/* Ptr to ADSDL descriptor                                    */
ads_long_descriptor       * pADSDL_D ;
/* Ptr to ADSD field descriptor                               */
ads_field_descriptor      * pADSD_FD ;
/* Ptr to ADSDL field descriptor                              */
ads_long_field_descriptor * pADSDL_FD ;
⋮
/* Initialize the pointer to the ADSDL descriptor or the      */
/* ADSD descriptor depending on mqcih.ADSDescriptor           */

if (mqcih.ADSDescriptor && MQCADSD_MSGFORMAT)
{
    pADSDL_D = pVector->brmq_sm_adsd_offset;   /* Long form  */
    pADSDL_FD = pADSDL_D + sizeof(ads_long_descriptor) ;
    ⋮
    /* Enter a loop where we process all field descriptors   */
```

```
    /* in the ADSDL sequentially                                */

    do
    {
        /* Perform some processing                              */
        .
        pADSDL_FD += sizeof(ads_long_field_descriptor) ;
    }
    while (pADSDL_FD < pADSDL_D->adsdl_length ) ;
}

else              /* Short form                                 */
{
    pADSD_D = pVector->brmq_sm_adsd_offset;    /* Short form */
    pADSD_FD = pADSD_D + sizeof(ads_descriptor) ;
    /* Enter a loop where we process all field descriptors   */
    /* in the ADSD sequentially                              */

    do
    {
        /* Perform some processing                          */
        .
        pADSD_FD += sizeof(ads_field_descriptor) ;
    }
    while (pADSD_FD < pADSD_D->adsd_length ) ;
}
.
```

5. Identify the fields in the ADS.

   The ADS is mapped to a structure that is generated when you assemble your map. If you include a `keyword=parameter` value of `DSECT=ADSDL` in your mapset definition macro, you get the long form of the ADS. The output from map assembly is a union of two structures: an input structure and an output structure. This example shows part of such a union (only the first field definition is shown for each structure, and the comments have been added following map assembly).

```
union
{
    struct {
        char   dfhms1[12];    /* 12 reserved bytes           */
        int    dfhms2;        /* Offset to next field        */
        int    tranidl;       /* Data length of this field   */
        int    tranidf;       /* Flag or attribute value     */
        int    dfhms3[7];     /* Extended attributes array   */
        char   tranidi[4];    /* Data value of field         */
        ...
    } bmstmp1i;               /* Input structure             */

    struct {
        char   dfhms56[12];   /* 12 reserved bytes           */
        int    dfhms57;       /* Offset to next field        */
        int    dfhms58;       /* Data length of this field   */
        int    tranida;       /* Flag or attribute value     */
        int    tranidc;       /* Extended attribute          */
        int    tranidp;       /* Extended attribute          */
        int    tranidh;       /* Extended attribute          */
        int    tranidv;       /* Extended attribute          */
        int    tranidu;       /* Extended attribute          */
        int    tranidm;       /* Extended attribute          */
        int    tranidt;       /* Extended attribute          */
        char   tranido[4];    /* Data value of field         */
        ...
    } bmstmp1o;               /* Output structure            */

} bmstmp1;                    /* Union                       */
```

   The two structures are functionally identical, except that the input structure includes the extended attribute values in a 7–element array, and the output structure provides individually named fields.

   You can use the following code fragment to set pointers to the start of the ADS. The structure names shown in the example DSECT, are used for illustration. Two pointers are set, the first to address inbound data and the second to address outbound data. It is assumed that `pVector` already addresses the start of the `brmq_send_map` vector.

```
/* #includes                                                */
#include cmqc.h               /* IBM MQ header               */
```

```
#include dfhbrmqh.h           /* Vector structures           */
#include dfhbrarh.h ..        /* ADSD structures             */
#include mydsect.h            /* DSECT from map assembly     */
.
bmstmp1i * pADSI ;            /* Pointer to the inbound ADS  */
bmstmp1o * pADSO ;            /* Pointer to the outbound ADS */
bmstmp1i * pADSI_An ;         /* Inbound ADS Anchor          */
bmstmp1o * pADSO_An ;         /* Outbound ADS Anchor         */
.
/* We are dealing with an outbound vector, so we will       */
/* initialize the outbound pointer to address the ADS       */

pADSO = pVector->brmq_sm_adsd_offset ;

/* Save initial value as anchor                             */

pADSO_An = pADSO ;

/* Move to the start of the first field                     */

pADSO += pADSDL_FD->adsdl_field_offset ;

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

do
{
    /* Perform some processing */
    .
    /* Add offset of next field to ADS Anchor value         */
    /* to address the next field                            */

    pADSO = pADSO_An + pADSDL_FD->adsdl_field_offset ;
}
while (pADSDL_FD < pADSDL_D->adsd_length ) ;
.
```

Interpreting RECEIVE MAP vectors for the CICS-MQ bridge

A RECEIVE MAP request is a request for the client to provide a RECEIVE MAP on the next input message.

Unlike a SEND MAP vector, an outbound RECEIVE MAP request vector never contains an ADS. It contains an ADSD or ADSDL that describes the ADS data that it requires in the next inbound RECEIVE MAP vector, if MQCADSD_RECV has been specified in MQCIH.ADSDescriptor. The RECEIVE MAP vector structure differs from that of the SEND MAP vector. The main difference is that no fields give the offset and length of the ADS.

To interpret a RECEIVE MAP vector (assuming that the message contains an ADSD or ADSDL):

1. Get the message containing the RECEIVE MAP request vector from the bridge reply queue into a message buffer.
2. Locate the start of the outbound RECEIVE MAP vector in the message buffer. This vector is appended to the MQCIH and so is at an offset equal to the length of the CIH from the start of the message buffer. You can use this code fragment as a model.

```
/* #includes                                                */
#include cmqc.h                /* IBM MQ header            */
#include dfhbrmqh.h            /* Vector structures        */
.
/* #defines                                                 */
.
MQCHAR * MsgBuffer ;           /* Message buffer pointer   */
brmq_receive_map_request * pVector ;   /* Vector pointer   */
.
/* Get message from reply queue                             */
.
/* Set the vector pointer to the start of the vector        */
pVector = MsgBuffer + ((MQCIH *) MsgBuffer)->StrucLength ;
.
```

3. Identify the starting address ADSD or ADSDL from the RECEIVE MAP vector.

   This following diagram shows the structure of an outbound RECEIVE MAP request vector. The diagram assumes that you have set a pointer called `pVector` to address the start of the `brmq_receive_map_request` vector, as in the previous code fragment.

```
|--------x'vvvvvvvv'----------------->|

 sizeof(brmq_receive_map_request)
|------------------------>|

      1802 0           --> x'wwwwwwww' <--
 -------------  ...  ----------------
|vvvv|FFFF|D444|     |wwww| ADSD or   |
|vvvv|1802|6000|     |wwww| ADSDL     |
 -------------  ...  ----------------
↑                    ↑
pVector              pVector->brmq_rmr_adsd_len
```

   Values in the diagram shown like this:

```
ABCD
1234
```

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

   Field pVector->brmq_rmr_adsd_len gives the length of the ADSD or ADSDL. No offset is given because the ADSDL is appended directly to the `brmq_receive_map_request` vector.

4. Identify the fields in the ADSD or ADSDL.

   To do so, proceed in general as for the SEND MAP vector described in “Interpreting SEND MAP vectors for the CICS-MQ bridge” on page 96 . Use the following code fragment, however, to set pointers to the start of the ADSD or ADSDL.

```
⋮
if (mqcih.ADSDescriptor && MQCADSD_MSGFORMAT)
{
    pADSDL_D = pVector + sizeof(brmq_receive_map_request) ;
    ⋮
}

else                                  /* Short form          */
{
    pADSD_D = pVector + sizeof(brmq_receive_map_request) ;
    ⋮
}
⋮
```

   The ADSD or ADSDL has exactly the same structure in the RECEIVE MAP vector as in the SEND MAP vector; so, after you have identified its start address, you can proceed as described for the SEND MAP vector.

Example of an ADSDL and an ADS

An example showing parts of an ADSDL and an ADS is explained.

Values in the diagrams shown like this:

```
ABCD
1234
```

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

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

```
...½ADSL..........±....CHO         L .........┌.......&$...TRANID
000BCCED0000000100040000CCD444444444D400000F000F000100054000EDCDCC44...
00581423000100OB00190003386000000000300000F000E00080000A00039159400...
↑          ↑                                  ↑   ↑       ↑
```

```
|           adsdl_field_count                    |   |       adsdl_first_field
adsdl_length                                     |   adsdl_map_columns
                                                adsdl_map_lines
```

The fields named in this example show the following values:

adsdl_length
This ADSDL is 0x05B8 bytes long.

adsdl_field_count
The ADS contains 0x1B (27) named fields.

adsdl_map_lines
The map has 0x18 (24) lines.

adsdl_map_columns
The map has 0x50 (80) columns.

adsdl_first_field
The start of the first field description in the ADSDL.

The next diagram shows the ADSDL first field descriptor and part of the next field descriptor:

```
TRANID                           ..............L ..TERMID
EDCDCC4444444444444444444444444400000000000000D400ECDDCC4444444444...
3915940000000000000000000000000000060000000C000430003594940000000000...
↑                                ↑       ↑   ↑       ↑
adsdl_field_name                 |       |   |       adsdl_next_field
                                 |       |   adsdl_field_data_len
                                 |       adsdl_field_offset
                                 adsdl_field_name_len
```

The fields named in this example show the following values:

adsdl_field_name
The name of the field in the ADS, in this case TRANID. Only the value of the field appears in the ADS, not its name.

adsdl_field_name_len
The length of the name of the field, in this case 6 bytes.

adsdl_field_offset
The absolute offset of the field from the start of the ADS. The offset is given as 0x0C (12) bytes, even though this field is the first one. The reason is that the first 12 bytes of the ADS are reserved and do not contain information for the application programmer.

adsdl_field_data_len
The data length of the named field, in this case 4 bytes.

adsdl_next_field
The start of the next field description.

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

```
.....................................................BAAA............
000000000000000200000000000000000000000000000000000CCCC000200000000...
000000000000000C000000000000000000000000000000000000002111000C00000000...
↑           ↑                                       ↑   ↑
|           Offset to next field                    |   Start of next field
12 bytes reserved                                   Value of field
```

The meanings of the values shown here are as follows:

12 bytes reserved
Reserved space at the start of every ADS, in both short and long form

Offset to next field
The information given for the current field is 0x2C bytes long, from the start of this fullword length value.

Value of field
The value of the field, with a name identified as TRANID in the ADSDL, is BAAA. The offset of the data is always 0x28 bytes from the start of the field for an ADS in long form.

Start of next field
The start of the information for the next field in the ADS.

In this case, the field information is an exact multiple of fullwords. If the information is not an exact multiple, padding bytes are added after the data value and before the next field to ensure that it starts on a fullword boundary. The padding bytes are included in the *offset to next field* value.

A number of attribute and extended attribute values for the field, not identified here, are between the fullword giving the offset to the next field and the field value itself.

Example of a 3270 transaction running under the CICS-MQ bridge

This example highlights the differences in the data flows that take place when a CICS 3270 transaction interacts with a 3270 terminal, and when it interacts with a CICS-MQ bridge application.

In this example, the transaction has an identifier of BAAA. It uses BMS maps, which allow the transaction to be adapted to run under the CICS-MQ bridge.

In the CICS environment, the transaction is started by entering its name at the CICS 3270 terminal and pressing Enter. Logic in the transaction causes it to issue an EXEC CICS SEND MAP command the first time that it is called in a pseudoconversation, and then to terminate by issuing the command EXEC CICS RETURN TRANSID(BAAA).

The user enters values into fields in the map that is displayed at the terminal, and then presses an AID key. Logic in the transaction the second time that it is called causes it to issue an EXEC CICS RECEIVE MAP command to receive the map. It updates certain fields in the map by changing values in its own application data structure, and then issues an EXEC CICS SEND MAP command to redisplay the map at the user's terminal.

The user can then update fields in the redisplayed map, and start the RECEIVE MAP - SEND MAP cycle again. The logic can be illustrated like this (where EC represent EXEC CICS):

```
Terminal user                                   3270 Transaction
BAAA <ENTER>           ------------------->     <Initial start>
                                                <business logic>
                       <-------------------     EC SEND MAP FROM(ads)
                                                EC RETURN TRANSID(BAAA)

Update fields
<ENTER>                ------------------->     EC RECEIVE MAP INTO(ads)
                                                <business logic>
                       <-------------------     EC SEND MAP
                                                EC RETURN TRANSID(BAAA)
Update fields
<ENTER>                ------------------->     EC RECEIVE MAP
```

When you run this transaction using the CICS-MQ bridge, the physical terminal is replaced by your non-CICS application. The logic of the 3270 transaction is unchanged, and the application data that it receives is the same, but the data that flows and the means by which it is transmitted are different. Instead of a 3270 data stream, an IBM MQ message is used that contains an MQCIH structure, a vector structure, and, optionally, a representation of the application data structure.

Including these objects in the message depends on the direction in which the message flows (inbound to the bridge or outbound from the bridge), the sequence of the message in the exchange, and whether an application data structure descriptor has been requested by setting the appropriate value in a field in the MQCIH.

"Exact emulation without optimization" on page 103 shows the flows that take place when the previous scheme is emulated exactly. You can optimize the flow by including more than one vector in inbound messages, as shown in "Improved emulation with optimization" on page 104.

In these examples, assume that MQCIH.ADSDescriptor is set to:

```
MQCADSD_SEND + MQCADSD_RECV + MQCADSD_MSGFORMAT
```

As a result, application data structure descriptors in long form are appended to both outbound and inbound application data structures during the exchange of messages.

For clarity, the details of messaging are omitted here. For a description of the queuing model used by the CICS-MQ bridge, see IBM MQ product documentation.

Exact emulation without optimization

This example shows the data flows when you use the CICS-MQ bridge to emulate the original logic of the example 3270 transaction exactly.

```
CICS-MQ bridge application              3270 Transaction

MQPUT to     --MQCIH--------------------------------> <Initial start>
Bridge                    (Note 1)                   <business logic>
RequestQ

MQGET from  <--MQCIH+brmq_send_map+ADS+ADSDL--------  EC SEND MAP FROM(ads)
Bridge                    (Note 2)                   EC RETURN
ReplyQ                                                  TRANSID(BAAA)

MQPUT to     --MQCIH--------------------------------> <Start>
Bridge                    (Note 3)
RequestQ

MQGET from  <--MQCIH+brmq_receive_map_request+ADSDL-  EC RECEIVE MAP INTO(ads)
Bridge                    (Note 4)                         |
ReplyQ                                                     |
                                                        waits
Updates fields                                             |
                                                           |
MQPUT to     ----MQCIH+brmq_receive_map+ADS--------->      ↓
Bridge                    (Note 5)
RequestQ
                                                        <business logic>

MQGET from  <--MQCIH+brmq_send_map+ADS+ADSDL--------  EC SEND MAP FROM(ads)
Bridge                    (Note 6)                   EC RETURN
ReplyQ                                                  TRANSID(BAAA)
                                                        ⋮
```

Note:

1. The initial flow from the application contains just an MQCIH. The MQCIH includes control information specifying which transaction is to be started.
2. The return flow from the 3270 transaction contains an MQCIH, which includes a facility token, to be used for all subsequent flows, and diagnostic information if an error has occurred. It also contains a SEND MAP vector structure containing control information relating to the map itself and data that represents the map. If the initiating application has requested it, an application data structure descriptor is also included.
3. The bridge application sends a message back containing only an MQCIH, which contains control information to start the transaction once more.
4. The 3270 transaction issues an EXEC CICS RECEIVE MAP command, as it does in the legacy environment. However, in the bridge environment the map data is not immediately available. The call is converted to a message containing an outbound RECEIVE MAP request vector. The application data structure descriptor is also included in the message. In this example, the transaction waits while the message is turned around by the bridge application. The model here is a little different from that in the legacy environment. However, the bridge architecture allows messages to contain more than one vector, so a number of requests can be satisfied by a single inbound message.
5. After updating fields in the application data structure, the bridge application sends an inbound RECEIVE MAP reply vector to satisfy the outbound request.
6. The 3270 transaction issues EXEC CICS SEND MAP, which converts to a SEND MAP vector, and the cycle repeats.

If you examine the flows that are captured when such a transaction is run using the Passthrough tool (CICS SupportPac CA1E CICS 3270 Bridge Passthrough), you can identify the structures with the help of the available online documentation.

Improved emulation with optimization

This example shows optimized data flows for the example 3270 transaction using the CICS-MQ bridge.

```
CICS-MQ Bridge Application                              3270 Transaction

MQPUT to     --MQCIH---------------------------------→ <Initial start>
Bridge                                                 <business logic>
RequestQ

MQGET from  ←--MQCIH+brmq_send_map+ADS+ADSDL--------- EC SEND MAP FROM(ads)
Bridge                                                 EC RETURN
ReplyQ                                                    TRANSID(BAAA)

MQPUT to     ----MQCIH+brmq_receive_map+ADS---------→ <Start>
Bridge
RequestQ                                               EC RECEIVE MAP INTO(ads)
                                                       <business logic>

MQGET from  ←--MQCIH+brmq_send_map+ADS+ADSDL--------- EC SEND MAP FROM(ads)
Bridge                                                 EC RETURN
ReplyQ                                                    TRANSID(BAAA)
                                                       ⋮
```

If you compare this sequence with the unoptimized flows shown in “Exact emulation without optimization” on page 103 , you can see that the CICS transaction does not have to send a RECEIVE MAP request vector, because the inbound RECEIVE MAP vector has already anticipated the requirement and the inbound map is already available to the transaction.

Chapter 8. Developing applications to use the CICS-MQ adapter

The CICS-MQ adapter implements the IBM MQ Message Queue Interface (MQI) for use by CICS application programs. The adapter also supports a CICS-MQ API-crossing exit. You can use this exit to intercept MQI calls as they are being run, for monitoring, testing, maintenance, or security purposes.

For transaction integrity, the adapter supports sync pointing under the control of the CICS Recovery Manager, so that units of work can be committed or backed out as required. The adapter also supports security checking of IBM MQ resources when used with an appropriate security management product, such as Security Server (previously known as RACF). The adapter provides high availability with automatic reconnection after a queue manager termination, and automatic resource resynchronization after a restart. It also features an alert monitor that responds to unscheduled events such as a shutdown of the queue manager.

API stub program to access IBM MQ MQI calls

To access IBM MQ, application programs that run under CICS must be link-edited with the supplied API stub program, CSQCSTUB, unless they are using dynamic calls. CSQCSTUB provides the application with access to all MQI calls.

For information about calling the CICS stub dynamically, see Developing applications in IBM MQ product documentation.

CSQCSTUB is shipped as module DFHMQSTB in the SDFHLOAD load library, and as DFHMQSTX in the SDFHAUTH load library. Both modules are defined with the following aliases:

CSQCBFMH
CSQCCB
CSQCCLOS
CSQCCONN
CSQCCONX
CSQCCTL
CSQCCTMH
CSQCDISC
CSQCDTMH
CSQCDTMP
CSQCINQ
CSQCIQMP
CSQCMHBF
CSQCOPEN
CSQCGET
CSQCPUT
CSQCPUT1
CSQCSET
CSQCSTAT
CSQCSTMP
CSQCSTUB
CSQCSUB
CSQCSUBR

The aliases CSQBFMH, CSQCCB, CSQCCTL, CSQCCTMH, CSQCDTMH, CSQCDTMP, CSQCIQMP, CSQCMHBF, CSQCSTAT, CSQCSTMP, CSQCSUB, and CSQCSUBR are only used when CICS is connected to Version 7 or later of IBM MQ.

Existing CICS-MQ applications can run unchanged with the CICS shipped CICS-MQ Adapter. You do not need to compile or link-edit them again. For new or changed applications, you can use existing link-edit procedures, and you can use stubs shipped with CICS or IBM MQ, unless the application uses the new API calls that were added in Version 7 of IBM MQ. The Version 7 API calls are only supported in CICS when you use the stubs shipped with CICS, not the stubs shipped with IBM MQ. The new API calls are MQBUFMH, MQCB, MQCTL, MQCRTMH, MQDLTMH, MQDLTMP, MQINQMP, MQMHBUF, MQSETMP, MQSTAT, MQSUB, and MQSUBRQ.

Asynchronous message consumption and callback routines

Asynchronous message consumption, which is available in Version 7 or later of IBM MQ, uses the MQCB and MQCTL MQI calls. If you use these functions in a CICS application program, CICS carries out some aspects of the processing, so when you code your application you must use the information given here in addition to using the IBM MQ programming documentation.

To use IBM MQ asynchronous message consumption, you can register programs known as callback routines for multiple message destinations, including queues and topics. When a suitable message is sent to the destination, it is passed to the callback routine. You can also set up event handlers to be notified of conditions such as a queue manager quiescing.

When you register a CICS command level application program as a callback routine, CICS obtains the messages passed by IBM MQ and links to the callback routine using EXEC CICS LINK. CICS passes the message data to the callback routine in a set of containers on a channel.

A program that you use as a callback routine must meet the following requirements:

- Your callback routine can be in any programming language supported by CICS, but note that using Java for a callback routine causes excessive TCB switching between the Java and non-Java environments, which affects performance.
- Compile and link-edit your callback routine with the RENT option and the options AMODE(31), RMODE(ANY).
- Code your callback routine to threadsafe standards, and define it to CICS with the attributes CONCURRENCY(THREADSAFE) and API(CICSAPI), to optimize the amount of TCB switching. Using API(OPENAPI) would affect performance because excessive TCB switching occurs if storage protection is active for a callback routine.

  A program defined with CONCURRENCY(REQUIRED) and API(CICSAPI) enables a program coded to threadsafe standards to run from the start of the program on an open TCB. An L8 open TCB is used regardless of the execution key.
- Define your callback routine as a local program, not as remote, or dynamic. Remote and dynamic programs are not supported as callback routines because the data structures that CICS passes in the containers include pointers to local storage in a region. An attempt to register a callback routine that is defined as remote fails with the reason code MQRC_MODULE_NOT_FOUND.

CICS containers for callback routines

The containers that CICS passes to callback routines use a channel named MQ_ASYNC_CONSUME. The name of the channel, and the names of the containers, are all 16 characters long, so the names of the containers given here are padded with blanks. The containers are as follows:

Table 10. Message data containers passed to callback routines

Container name	Content
MsgDesc	MQMD2 - Message descriptor version 2

Table 10. Message data containers passed to callback routines (continued)	
Container name	**Content**
GetMsgOpts	MQGMO - Get Message Options
Buffer	The message. Options specified in MQGMO, such as truncation and conversion, have the same effects on the message as they do with an MQGET call.
Context	MQCBC - Callback Context version 2

The IBM MQ product documentation describes the MQMD, MQGMO, and MQCBC data structures. See IBM MQ product documentation.

When your callback routine retrieves data from the message data containers, use GET CONTAINER SET commands rather than GET CONTAINER INTO commands, to allow for future changes in the size of the data structures. Code the FLENGTH keyword and test for its value being zero, to check for empty containers. For example, for event invocations of callback routines, the MsgDesc, GetMsgOpts, and Buffer containers are empty.

Handling sync points, abends, and quiesce in callback routines

To ensure that your callback routine interacts correctly with the CICS-MQ adapter, ensure that the following requirements are met:

1. Use the EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK commands only in message consumer callback routines when they are started for a message. Do not use these commands in event handlers, or when callback routines are called to register, deregister, start, or stop.
2. If you use the EXEC CICS ABEND command in a callback routine, the MQCTL command fails with reason code MQRC_CALLBACK_ERROR(2452), because the abend is handled and control is returned to the caller. An abend in one callback routine stops invocation of all callback routines being managed by the CICS task that issued the MQCTL request.
3. In an event handler callback routine, issue an MQCTL STOP command when the queue manager or connection is quiescing, to enable the CICS-MQ interface to quiesce successfully. If you do not issue this command, the CICS-MQ interface cannot quiesce because of the active event handler, and the CICS-MQ interface must be force closed.
4. When you use MQCB to register a callback routine:
   a. In the CallBackDesc (MQCBD) parameter, *do not* set MQCBDO_FAIL_IF_QUIESCING in the Options field. CICS needs to issue MQCB commands as part of its processing to stop new work arriving.
   b. In the GetMsgOpts (MQGMO) parameter, *do* set MQGMO_FAIL_IF_QUIESCING in the Options field, to ensure proper quiesce of the callback routine. If you do not set these quiesce options correctly, the CICS-MQ interface cannot quiesce because of the active event handler, and the CICS-MQ interface must be force closed.
   c. In the CallBackDesc (MQCBD) parameter, if you want to use the CallbackArea field to pass an area of storage to the callback routine, set it to the address of storage obtained using EXEC CICS GETMAIN rather than the program's dynamic storage. The callback routine accesses this storage using the CallbackArea field in the Callback context structure MQCBC. Also, when the program ends, do not free the storage or any further storage areas anchored off it, but let CICS freemain the storage at the end of the task. This allows the storage to be available if CICS invokes the callback routine to deregister during end of task processing.
5. When you use MQCTL to control invocations of callback routines, in the ControlOpts (MQCTLO) parameter, if you want to use the ConnectionArea field to pass an area of storage to be shared by all callback routines, set it to the address of storage obtained using EXEC CICS GETMAIN rather than the program's dynamic storage. Callback routines access this storage using the ConnectionArea field in the Callback context structure MQCBC. Also, when the program ends, do not free the storage or any further storage areas anchored off it, but let CICS freemain the storage at the end of the task. This

allows the storage to be available if CICS invokes the callback routines to deregister during end of task processing.

Sample programs for asynchronous message consumption

This set of sample programs in COBOL shows you how IBM MQ asynchronous message consumption and callback routines work in the CICS environment. To use IBM MQ asynchronous message consumption, you must have Version 7 or later of IBM MQ. The sample programs are supplied with IBM MQ.

Run the sample registration client program to register three callback routines, which are an event handler and two message consumers, and start asynchronous message consumption. You can then use the sample messaging client program to send messages from a CICS terminal to a queue and a topic in IBM MQ, for asynchronous consumption by the two message consumers. You can use these messages to instruct one of the message consumers to deregister, resume, or suspend the callback routines, or to stop asynchronous message consumption.

Table 11. Sample COBOL programs for asynchronous message consumption

Sample program	Function	Actions
CSQ4CVRG	Registration client	Registers event handler and message consumers. Runs under CICS transaction MVRG.
CSQ4CVPT	Messaging client	Publishes your messages to a topic or sends your control messages to a queue. Runs under CICS transaction MVMP.
CSQ4CVCN	Message consumer for basic messages	Consumes your IBM MQ messages that are published under the topic `News/Media/Movies`.
CSQ4CVCT	Message consumer for control messages	Consumes your control messages from the SAMPLE.CONTROL.QUEUE queue and issues appropriate MQCB or MQCTL commands to stop asynchronous message consumption or deregister, resume, or suspend a callback routine.
CSQ4CVEV	Event handler	Receives notifications from IBM MQ when a condition occurs that affects the whole callback environment, such as a queue manager stopping or quiescing.

Design of the samples for asynchronous message consumption

The sample COBOL programs include a registration client and messaging client that run under CICS transaction s, and three callback routines that consume messages or events from CICS-MQ. Each of the sample programs uses COBOL DISPLAY statements to display messages at appropriate points so that you can observe its behavior.

The messages from the sample programs are sent to the transient data queue CESE, which is the CEEMSG data set. The messages issued by the event handler and message consumers indicate the CICS-MQ call type for which the program was driven, such as START or REGISTER.

Registration client program CSQ4CVRG

The registration client is started from a CICS terminal under the CICS transaction MVRG, but it does not take any input. The registration client registers the event handler CSQ4CVEV, the message consumer CSQ4CVCN, and the message consumer CSQ4CVCT with IBM MQ as callback routines. It also passes to CSQ4CVCT a data structure containing the names of all three registered callback routines, with the object handles associated with the two message consumers.

When the registration client has registered the callback routines, it issues an MQCTL START_WAIT command to start asynchronous message consumption. It then suspends until control is returned to it.

Control is returned to the registration client if one of the callback routines issues an MQCTL STOP command to stop asynchronous message consumption.

Messaging client program CSQ4CVPT

The messaging client is started from a CICS terminal under the CICS transaction MVMP, and it takes command line input. The messaging client has two functions:

- Publish a basic text message to the topic `News/Media/Movies` to be consumed by CSQ4CVCN.
- Put a control message on the queue SAMPLE.CONTROL.QUEUE to be consumed by CSQ4CVCT. The control messages make CSQ4CVCT issue commands to change the behavior of the set of sample programs, and you can observe the results through the messages displayed by the sample programs.

"CSQ4CVPT, sample messaging client" on page 110 explains how to use CSQ4CVPT to create messages.

Message consumer program CSQ4CVCN, for basic messages

CSQ4CVCN is a message consumer that consumes basic IBM MQ messages that you publish under the topic `News/Media/Movies` , using the messaging client.

When CSQ4CVCN is called with the IBM MQ call type MSG_REMOVED, it retrieves the inbound message and echoes it to the CICS job log.

Message consumer program CSQ4CVCT, for control messages

CSQ4CVCT is a message consumer that consumes IBM MQ messages that you put on the queue SAMPLE.CONTROL.QUEUE. When CSQ4CVCT is called with the IBM MQ call type MSG_REMOVED, it retrieves the inbound message and also the data structure that the registration client passed to it.

You can use the messaging client to create control messages to instruct CSQ4CVCT to take the following actions. CSQ4CVCT issues appropriate MQCB or MQCTL commands to carry out the action that you request:

- Deregister a specified callback routine
- Resume a specified callback routine
- Suspend a specified callback routine
- Stop asynchronous message consumption by issuing an MQCTL STOP command, returning control to the registration client

You may instruct CSQ4CVCT to suspend or deregister itself, but, if you do that, you can no longer control the behavior of the sample programs. If you suspend or deregister all the callback routines, control returns to the registration client, and the task ends.

Event handler program CSQ4CVEV

As an event handler, CSQ4CVEV receives IBM MQ notifications when a condition occurs that affects the whole callback environment, such as a queue manager stopping or quiescing.

If reason code CONNECTION_QUIESCING is received, CSQ4CVEV issues an MQCTL STOP command to stop asynchronous message consumption and returns control to the registration client.

Setting up the samples for asynchronous message consumption

Set up the sample programs to demonstrate IBM MQ asynchronous message consumption and callback routines.

About this task

The sample programs CSQ4CVRG, CSQ4CVPT, CSQ4CVCN, CSQ4CVCT, and CSQ4CVEV are supplied with IBM MQ. For IBM MQ Version 7.01, apply the PTF for APAR PM06722 to obtain the samples. The source for the samples is in the IBM MQ library SCSQCOBS, and the load modules are in the IBM MQ library SCSQCICS.

The CICS resource definitions for the sample programs are provided in the IBM MQ group CSQ4SAMP. CSQ4SAMP also includes resource definitions for the CICS transactions MVRG and MVMP, which you use for the programs CSQ4CVRG (registration client) and CSQ4CVPT (messaging client) respectively.

Procedure

1. Include the IBM MQ library *thlqual*.SCSQCICS in the DFHRPL concatenation in your CICS procedure, and ensure that the high-level qualifier *thlqual* is for IBM MQ Version 7.0.1 or above.

   Include the library after the CICS libraries to ensure that the correct code is used.
2. In CICS, install the resource definitions supplied by IBM MQ for the sample programs and the transactions MVRG and MVMP, as follows:
   a) Delete any existing CSQ4SAMP group in the CICS region.
   b) If you are applying the PTF for IBM MQ Version 7.0.1, use the version of member CSQ4S100 in the library SCSQPROC that is shipped with the PTF.

      This member contains the new resource definitions.
   c) Use DFHCSDUP to update the CSD with the data set *thlqual*.SCSQPROC(CSQ4S100), which populates the CSQ4SAMP group.
   d) Install the CSQ4SAMP group.

   If you use the CEDA transaction to install CICS-MQ adapter resources in an active CICS system, you must first shut down the adapter and wait until the alert monitor has finished its work.
3. Define the queue SAMPLE.CONTROL.QUEUE, which is used by the message consumer CSQ4CVCT, to the IBM MQ queue manager or queue-sharing group that is associated with the CICS region.

   The MQCONN resource definition for the CICS region names the IBM MQ queue manager or queue-sharing group. The queue name is coded in the sample programs.
4. Optional: Define the topic News/Media/Movies to the IBM MQ queue manager or queue-sharing group.

   If you do not define the topic, IBM MQ creates it at run time under the default Administrative Object.

   The topic, like the queue name, is coded in the sample programs.
5. Start the sample registration client program CSQ4CVRG under the CICS transaction MVRG.

   The registration client registers the event handler CSQ4CVEV, the message consumer CSQ4CVCN, and the message consumer CSQ4CVCT with IBM MQ as callback routines.

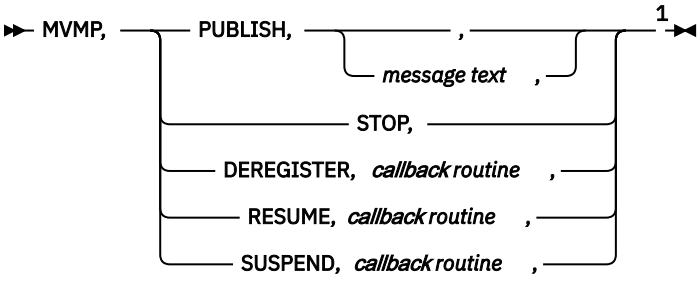
What to do next

You can now use the sample messaging client program CSQ4CVPT to publish basic messages and create control messages and observe the effects. "CSQ4CVPT, sample messaging client" on page 110 explains how to use CSQ4CVPT.

CSQ4CVPT, sample messaging client

Use the sample messaging client program CSQ4CVPT from a CICS terminal under the CICS transaction MVMP.

Messaging client commands



Notes:

[1] The keywords and the names of callback routines are not case-sensitive. Input fields are positional and are separated and terminated by commas.

Parameters

MVMP
The CICS transaction for CSQ4CVPT.

PUBLISH , { , | *message text* , }
Publish the specified message text as an CICS-MQ retained message under the topic `News/Media/Movies` . The message is consumed by the message consumer program CSQ4CVCN. If you omit the message text and specify the closing comma, default message text is published.

DEREGISTER , *callback routine* ,
Put an CICS-MQ message to the queue SAMPLE.CONTROL.QUEUE to instruct the message consumer program CSQ4CVCT to deregister the specified callback routine.

- If you deregister CSQ4CVCT, you can no longer control the behavior of the sample programs.
- If you deregister all the callback routines, control returns to the registration client.

RESUME , *callback routine* ,
Put an CICS-MQ message to the queue SAMPLE.CONTROL.QUEUE to instruct the message consumer program CSQ4CVCT to resume the specified callback routine.

SUSPEND , *callback routine* ,
Put an CICS-MQ message to the queue SAMPLE.CONTROL.QUEUE to instruct the message consumer program CSQ4CVCT to suspend the specified callback routine.

- If you suspend CSQ4CVCT, you can no longer control the behavior of the sample programs.
- If you suspend all the callback routines, control returns to the registration client.

STOP ,
Put an CICS-MQ message to the queue SAMPLE.CONTROL.QUEUE to instruct the message consumer program CSQ4CVCT to stop asynchronous message consumption by issuing an MQCTL STOP command. This action returns control to the registration client CSQ4CVRG.

To publish the default message text for consumption by CSQ4CVCN, use this command:

```
MVMP,PUBLISH,,
```

To publish the message text `A short message` for consumption by CSQ4CVCN, use this command:

```
MVMP,publish,A short message,
```

To stop asynchronous message consumption, use this command:

```
MVMP,STOP,
```

To deregister the event handler CSQ4CVEV, use this command:

```
MVMP,DEREGISTER,CSQ4CVEV,
```

To resume the message consumer CSQ4CVCN, use this command:

```
MVMP,resume,csq4cvcn,
```

To suspend the event handler CSQ4CVEV, use this command:

```
MVMP,SUSPEND,CSQ4CVEV,
```

The CICS-MQ API-crossing exit

CICS provides an API-crossing exit for use with the CICS-MQ adapter. You can use this exit to intercept IBM MQ calls as they are being run, for monitoring, testing, maintenance, or security purposes. The exit runs in the CICS address space.

This section contains product-sensitive programming interface information.

You can use the CICS-MQ API-crossing exit for these purposes:

- Operate additional security checks by examining the contents of each message before and after each MQI call
- Replace the queue name supplied in the message with another queue name
- Cancel the call and either issue a return code of 0 to simulate a successful call or another value to indicate that the call was not performed
- Monitor the use of MQI calls in an application
- Gather statistics
- Modify input parameters on specific calls
- Modify the results of specific calls

Using the CICS-MQ API-crossing exit degrades IBM MQ performance. Plan your use of it carefully.

The CICS-MQ API exit program is called CSQCAPX. CICS supplies a sample exit program, and a program definition for CSQCAPX in group DFHMQ. To use the exit, this program must be in the DFHRPL concatenation, it must be defined in the CICS system definition file (CSD), and it must be enabled. You can write your own exit program in place of the supplied sample, and you must call your program CSQCAPX as well.

When CSQCAPX is loaded, a confirmation message is written to the CICS-MQ adapter control panel, CKQC, or the console. If it cannot be loaded, a diagnostic message is displayed, but otherwise the application program runs normally.

Important: The supplied definition of CSQCAPX specifies the parameter CONCURRENCY(THREADSAFE). If you write your own exit program, specify CONCURRENCY(THREADSAFE) when you define it, and use only threadsafe CICS commands in the exit. Also specify this setting, and use only threadsafe CICS commands, for any programs that your exit program calls. Examine any API-crossing exits written in earlier CICS releases to ensure that their logic is threadsafe.

How the CICS-MQ API-crossing exit is called

Using the CICS-MQ API-crossing exit degrades the performance of IBM MQ for z/OS, so plan your use of it carefully. When enabled, the CICS-MQ API-crossing exit is called in the following circumstances.

- CSQCAPX is called by all applications that use the CICS-MQ adapter.
- CSQCAPX is called every time one of the following MQI calls is made:
  - MQCB
  - MQCTL
  - MQCLOSE
  - MQGET
  - MQINQ
  - MQOPEN
  - MQPUT
  - MQPUT1
  - MQSET
  - MQSTAT

- MQSUB
- MQSUBRQ

CSQCAPX is called both before and after a call.

- During processing for asynchronous message consumption, CSQCAPX is called in three circumstances:
  - Before calling the IBM MQ routine CSQAVICD, if data conversion is required for a message before passing it to the callback routine.
  - Before and after calls to the callback routine.

  For these calls, the IBM MQ MQI parameter list that is referenced by the MQXP_PCOPYPARM fields in the parameter list passed to CSQCAPX contains the same data that is passed to a callback routine, that is, the MQMD, MQGMO, Buffer, and Context.

CSQCAPX is *not* called for the function calls for message properties and message handles, which are the MQCRTMH, MQDLTMH, MQSETMP, MQINQMP, MQDLTMP, MQMHBUF, and MQBUFMH calls.

To reduce excessive use of TCB switching, you should ensure that the exit program is defined as THREADSAFE.

The exit program is called once before a call is executed, and once after the call is executed. On the "before" type of exit call, the exit program can modify any of the parameters on the MQI call, suppress the call completely, or allow the call to be processed. If the call is processed, the exit is called again after the call has completed.

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

Communication with the CICS-MQ crossing exit program

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

The CICS Exec Interface Block field EIBCALEN shows the length of this area. The structure of this communication area is defined in the crossing exit program as follows:

```
*
MQXP_COPYPLIST       DSECT
                     DS  0D         Force doubleword alignment
MQXP_PXPB            DS   AL4       Pointer to exit parameter block
MQXP_PCOPYPARM       DS 11AL4       Copy of original plist
*
                       ORG  MQXP_PCOPYPARM
MQXP_PCOPYPARM1      DS   AL4       Copy of 1st parameter
MQXP_PCOPYPARM2      DS   AL4       Copy of 2nd parameter
MQXP_PCOPYPARM3      DS   AL4       Copy of 3rd parameter
MQXP_PCOPYPARM4      DS   AL4       Copy of 4th parameter
MQXP_PCOPYPARM5      DS   AL4       Copy of 5th parameter
MQXP_PCOPYPARM6      DS   AL4       Copy of 6th parameter
MQXP_PCOPYPARM7      DS   AL4       Copy of 7th parameter
MQXP_PCOPYPARM8      DS   AL4       Copy of 8th parameter
MQXP_PCOPYPARM9      DS   AL4       Copy of 9th parameter
MQXP_PCOPYPARM10     DS   AL4       Copy of 10th parameter
MQXP_PCOPYPARM11     DS   AL4       Copy of 11th parameter
*
MQXP_COPYPLIST_LENGTH  EQU  *-MQXP_PXPB
                       ORG  MQXP_PXPB
MQXP_COPYPLIST_AREA  DS   CL(MQXP_COPYPLIST_LENGTH)
*
```

Field *MQXP_PXPB* points to the exit parameter block, MQXP.

Field *MQXP_PCOPYPARM* is an array of addresses of the call parameters. For example, if the application issues an MQI call with parameters P1, P2, or P3, the communication area contains:

```
PXPB,PP1,PP2,PP3
```

where *P* denotes a pointer (address) and XPB is the exit parameter block.

Writing your own CICS-MQ API-crossing exit program

You can use the sample API-crossing exit program (CSQCAPX) that is supplied with CICS as a framework for your own program. You must write your exit program to be threadsafe and declare your exit program as threadsafe.

About this task

For performance reasons, write your program in assembler language. If you write it in any of the other languages supported by CICS , you must provide your own data definition file. Link edit your program as AMODE(31) and RMODE(ANY). The layout of the MQXP parameter list that is passed to the crossing exit program is defined in the program.

Procedure

- Exits are written as extensions to the CICS-MQ code. Ensure that your exit does not disrupt any IBM MQ for z/OS programs or transactions that use the MQI.

  They are indicated with a prefix of CSQ, DFH, or CK.
- Your program can use all the APIs (for example, IMS, Db2®, and CICS) that a CICS task-related user exit program can use.

  You must use only threadsafe CICS commands in your exit program and in any programs that your exit program calls.
- Your program can use any of the MQI calls except MQCONN, MQCONNX, and MQDISC.

  However, any MQI calls in the exit program do not call the exit program a second time.
- When it is called after an MQI call, your program can inspect and modify the completion and reason codes set by the call.
- To find the name of an MQI call issued by an application, examine the **ExitCommand** field of the MQXP structure. To find the number of parameters on the call, examine the **ExitParmCount** field.

  You can use the 16 byte **ExitUserArea** field to store the address of any dynamic storage that the application obtains. This field is retained across uses of the exit and has the same lifetime as a CICS task.
- Your program can suppress the execution of an MQI call by returning MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION in the **ExitResponse** field.

  For the call to run (and the exit program to be called again after the call has completed), your exit program must return MQXCC_OK.
- Your program can issue **EXEC CICS SYNCPOINT** or **EXEC CICS SYNCPOINT ROLLBACK** commands. These commands commit or rollback all the updates done by the task up to the point that the exit was used.
- Parameter MQXP_EXITCALLPROG is included in the parmlist which is passed to the module through the commarea, and contains the name of the program that called the module. Because CSQCAPX is called as a User Replaceable Module (URM), an EXEC CICS ASSIGN INVOKINGPROG command returns blanks. The parameter MQXP_EXITCALLPROG can be used instead, as it contains the name of the calling program, identical in format to the output received from an EXEC CICS ASSIGN INVOKINGPROG command. This parameter is available if the parameter MXQP_VERISON has a value of 2.
- Your program must end by issuing an **EXEC CICS RETURN** command. It must not transfer control with an XCTL command.
- Create PROGRAM resource definitions for your exit program and any programs that your exit program calls.

  You must call your exit program CSQCAPX.

  Specify the following settings in your PROGRAM resource definitions:

  a) Specify CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED) for your exit program and any programs that your exit program calls.

You must use only threadsafe CICS commands in your exit program and in any programs that your exit program calls.

b) If you are using the CICS Transaction Server storage protection feature, specify EXECKEY(CICS) for your exit program and any programs to which it passes control, so that they run in CICS execution key.

For more information, see Exit programs and the CICS storage protection facility.

c) The parameters in the communication area are addresses so you must define the exit program as local to the CICS system, not as a remote program.

CICS supplies a sample program definition for CSQCAPX in group DFHMQ. This program definition specifies CONCURRENCY(THREADSAFE) and EXECKEY(CICS). You can use this sample as a basis for the program definition for your exit program.

Results

The CICS system tries to load the exit program when CICS connects to IBM MQ for z/OS. If this attempt is successful, message DFHMQ0301I is sent to the CKQC panel or to the system console. If the load is unsuccessful, for example if the load module does not exist in any of the libraries in the DFHRPL concatenation, message DFHMQ0315E is sent to the CKQC panel or to the system console.

The sample API-crossing exit program, CSQCAPX

The sample exit program is supplied as an assembler-language program. The source file (CSQCAPX) is supplied in *thlqual* .SDFHSAMP, where *thlqual* is the high-level qualifier used by your installation. This source file includes pseudocode that describes the program logic.

The sample program contains initialization code and a layout that you can use when writing your own exit programs.

The sample shows how to perform these tasks:

- Set up the exit parameter block
- Address the call and exit parameter blocks
- Determine for which MQI call the exit is being called
- Determine whether the exit is being called before or after processing of the MQI call
- Put a message on a CICS temporary storage queue
- Use the macro DFHEIENT for dynamic storage acquisition to maintain reentrancy
- Use DFHEIBLK for the CICS exec interface control block
- Trap error conditions
- Return control to the caller

The sample exit program writes messages to a CICS temporary storage queue (CSQ1EXIT) to show the operation of the exit. The messages show whether the exit is being called before or after the MQI call. If the exit is called after the MQI call, the message contains the completion code and reason code returned by the call. The sample uses named constants from the CMQXA macro to check on the type of entry; that is, before or after the call.

The sample does not perform any monitoring, but places time-stamped messages into a CICS queue indicating the type of call it is processing. The messages indicate the performance of the MQI, as well as the proper functioning of the exit program.

The sample exit program issues six EXEC CICS calls for each MQI call that is made while the program is running. When you use this exit program, the performance of CICS-MQ applications is degraded.

Enabling the CICS-MQ API-crossing exit

The supplied program definition for CSQCAPX specifies STATUS(DISABLED), The program is installed in a disabled state because using the exit program can significantly reduce IBM MQ performance. You must enable the API-crossing exit before you can use it.

About this task

You can activate the API-crossing exit temporarily using the CICS-MQ adapter control panels, the command line, or a CICS application program. You can enable the API-crossing exit permanently by modifying your CICS resource definitions.

Procedure

- To activate the API-crossing exit temporarily using the CICS-MQ adapter control panels, follow these steps:
  a) Type CKQC and press Enter to access the CICS-MQ adapter control panels.
  b) Select **Connection** from the menu bar.
  c) Select the **Modify** action from the menu.
  d) In the **Modification Options** secondary parameter window, select **Enable API Exit** and press Enter to alter the status of the API-crossing exit to Enabled.

  The screen shows this process:

```
   Connection        CKTI        Task
 +--------------------+------------------------------------------------------
 | Select an action.  |CS Adapter Control -- Initial panel
 |                    |
 | 3 1. Start...      |sing Tab key. Then press Enter.
 |   2. Stop...       |
 |   3. Modify...     |  +--------------------------------+
 |   4. Display       |  |      Modification Options      |
 |                    |  |                                |
 +--------------------+  | Select modify option. Then     |
 | F1=Help F12=Cancel |  | press Enter.                   |
 +--------------------+  |                                |
                         | 4 1. Reset statistics          |
                         |   2. Enable API Exit           |
                         |   3. Disable API Exit          |
                         |--------------------------------+
                         | F1=Help  F12=Cancel            |
                         +--------------------------------+


 F1=Help  F3=Exit
```

- To activate the API-crossing exit temporarily from the CICS command line, choose one of these methods:
  - Issue the command `CEMT S PROGRAM(CSQCAPX) ENABLED` from the CICS master terminal.
  - Issue the command `CKQC MODIFY N E` from the command line. The option E enables the connection. The option N relates to the connection statistics and is required, but has no effect.
- To activate the API-crossing exit temporarily using a CICS application program, issue an EXEC CICS LINK command to link to the adapter reset program, DFHMQRS (or CSQCRST, which is retained for compatibility), and issue the CKQC MODIFY command with the option E.
  This example shows how to do this:

```
EXEC CICS LINK PROGRAM('DFHMQRS ')
          INPUTMSG('CKQC MODIFY N E ')
```

  The MODIFY command must be padded with 4 trailing spaces plus another space as a separator (see CKQC commands from CICS(r) application programs). The option N relates to the connection statistics and is required, but has no effect.
- To run with the API-crossing exit permanently enabled, follow these steps:

a) Copy the definition of CSQCAPX from the CICS-supplied group DFHMQ to your own group.

b) Alter the CSQCAPX definition by changing the status from DISABLED to ENABLED, and install your new group.

c) Ensure that your group is installed after DFHMQ in any grouplist.

Disabling the CICS-MQ API-crossing exit

To improve IBM MQ performance, disable the CICS-MQ API-crossing exit when you no longer need it.

About this task

You can disable the API-crossing exit using the CICS-MQ adapter control panels, the command line, or a CICS application program.

Procedure

1. To disable the API-crossing exit using the CICS-MQ adapter control panels, follow these steps:

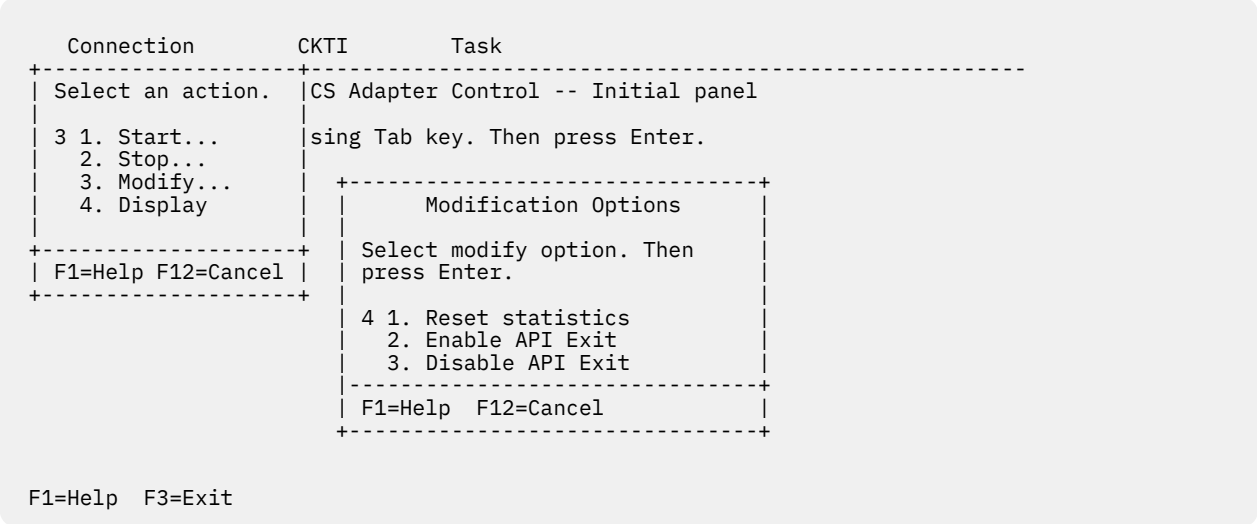
   a) Type CKQC and press Enter to access the CICS-MQ adapter control panels.

   b) Select **Connection** from the menu bar.

   c) Select the **Modify** action from the menu.

   d) In the **Modification Options** secondary parameter window, select **Disable API Exit** and press Enter to alter the status of the API-crossing exit to Disabled.

   The screen shows this process:



2. To disable the API-crossing exit from the CICS command line, choose one of these methods:

   - Issue the command `CEMT S PROGRAM(CSQCAPX) DISABLED` from the CICS master terminal.
   - Issue the command `CKQC MODIFY N D` from the command line. The option `D` disables the connection. The option `N` relates to the connection statistics and is required, but has no effect.

3. To disable the API-crossing exit using a CICS application program, issue an EXEC CICS LINK command to link to the adapter reset program, DFHMQRS (or CSQCRST, which is retained for compatibility), and issue the CKQC MODIFY command with the option `D`.
   This example shows how to do this:

```
EXEC CICS LINK PROGRAM('DFHMQRS ')
          INPUTMSG('CKQC MODIFY     N D ')
```

   The MODIFY command must be padded with 4 trailing spaces plus another space as a separator (see CKQC commands from CICS(r) application programs). The option `N` relates to the connection statistics and is required, but has no effect.

Chapter 9. Troubleshooting the CICS-MQ adapter

If you are having problems with the CICS-MQ adapter, CICS provides a range of information to help you diagnose the cause.

Depending on the information that you require and the environment in which you are currently working, you can use CEMT or **EXEC CICS INQUIRE** commands, the CICS-MQ adapter control panels, the CKQC DISPLAY command, or CICSPlex SM to display information about the CICS-MQ connection. For information about these different methods, see Displaying information about CICS-MQ connections.

The CKQC transaction (from the CICS-MQ adapter control panels) can display details of individual tasks using the connection to IBM MQ, and the state that they are in, such as a GET WAIT. Displaying tasks that are using the CICS-MQ connection explains how to display this information.

The CICS execution diagnostic facility (CEDF) traps entry to and exit from the CICS-MQ adapter for each MQI call, as well as trapping calls to all CICS API services.

The CICS-MQ adapter uses AP domain trace points in the range A000 to A1FF, and the trace entries are written to standard CICS trace destinations. The contents of trace points issued by the CICS-MQ adapter are documented in CICS IBM MQ trace points. Exception trace entries are unconditionally produced for error conditions. Non-exception trace entries are controlled by RI (Resource Manager Interface) and RA (Resource Manager Adapter) Level 1 and Level 2 tracing. For more information about CICS tracing, see CETR - trace control.

IBM MQ waits

If a task is waiting on the resource type MQseries, WMQ_INIT, or WMQCDISC, the CICS-MQ adapter has suspended it.

Resource type MQseries
The CICS-IBM MQ MQ adapter (DFHMQTRU module) put the task into a CICS wait because the WAIT option was used with the MQGET call and there was no message available. The resource name used for the wait is GETWAIT. The WAIT_MVS function of the dispatcher is used for this wait, and the wait type for workload management is OTHER_PRODUCT. The task can be purged.

Resource type WMQ_INIT
DFHMQIN1, the CICS-IBM MQ initialization program, issues this wait for DFHMQIN2 to complete. The WAIT_OLDC function of the dispatcher is used for this wait, and the wait type for workload management is MISC. The task can be purged.

Resource type WMQCDISC
A SET MQCONN NOTCONNECTED command has been issued with the WAIT or FORCE option, and the DFHMQTM module waits for the count of user tasks using IBM MQ to reach zero. The resource name is given as the name of the installed MQCONN resource definition for the CICS system. The WAIT_OLDC function of the dispatcher is used for this wait, and the wait type for workload management is MISC. The task can be purged.

What happens when the CICS-MQ connection shuts down

The connection between CICS and IBM MQ has two types of shutdown, quiesced (or orderly) shutdown and forced shutdown. Connection shutdown can result from an operator action, from a shutdown of CICS, or from the IBM MQ queue manager.

Table 12 on page 120 summarizes how the adapter handles different forms of connection shutdown when the connection is active. If CICS or IBM MQ shuts down when the connection is not active (for example, after it has been quiesced), no action is taken and no messages are issued.

Table 12. Shutting down a CICS adapter connection

Method of shutdown	**How this is handled by the adapter**
Quiesced shutdown of connection (EXEC CICS SET MQCONN NOTCONNECTED BUSY(WAIT|NOWAIT), CEMT SET MQCONN NOTCONNECTED, or CKQC STOP)	Mark the status of the adapter as *Quiescing*. Allow both active and waiting tasks to complete. Allow sync point. Do not allow calls from a new task. The last task initiates disconnection from IBM MQ.
Forced shutdown of connection (EXEC CICS SET MQCONN NOTCONNECTED BUSY(FORCE), CEMT SET MQCONN FORCENOTCON, or CKQC STOP FORCE)	Mark the status of the adapter as *StoppingForce*. Disconnect from IBM MQ. First resume tasks waiting in IBM MQ, including instances of CKTI, then forcepurge any inflight tasks that accessed WebSphere MQ.
CICS warm shutdown	Issue message DFHMQ0411I. Initiate a quiesced shutdown of the connection.
CICS immediate shutdown	Issue message DFHMQ0410 I. Any inflight tasks using IBM MQ are backed out.
CICS abend	Issue message DFHMQ0412 I.
IBM MQ quiesced	Initiate a quiesced shutdown of the connection.
IBM MQ abend or forced shutdown	Initiate a forced shutdown of connection.

Quiesced (or orderly) shutdown

A quiesced shutdown of the connection allows each CICS transaction to end before the interface is closed. When you use this method, you can expect no indoubt units of work when you reconnect CICS.

A quiesced shutdown occurs in each of the following situations:

- The CICS terminal operator issues an **EXEC CICS** or **CEMT SET MQCONN NOTCONNECTED** command, or a **CKQC STOP** command. CICS and the queue manager remain active.
- The CICS terminal operator issues the **CEMT PERFORM SHUTDOWN** command.
- The queue manager is quiesced by the command:

```
+CSQ1 STOP QMGR MODE(QUIESCE)
```

  This command stops the queue manager, allows the currently identified tasks to continue normal processing, and does not allow new tasks to identify themselves to the queue manager. CICS remains active.

Forced shutdown

A forced shutdown of the connection can abnormally end CICS transactions connected to the queue manager. Therefore, you might have indoubt units of work when the system is reconnected.

A forced shutdown occurs in each of these situations:

- The CICS terminal operator issues an **EXEC CICS SET MQCONN NOTCONNECTED** command with the FORCE option, or a **CEMT SET MQCONN FORCENOTCON** command, or a **CKQC STOP FORCE** command.
- The CICS terminal operator issues the CICS immediate shutdown command:

  ```
  CEMT PERFORM SHUTDOWN IMMEDIATE
  ```

  The queue manager remains active. For information about this command, see CEMT PERFORM SHUTDOWN.
- The IBM MQ forced shutdown command is issued:

  ```
  +CSQ1 STOP QMGR MODE(FORCE) or +CSQ1 STOP QMGR MODE(RESTART)
  ```

  CICS remains active.
- A IBM MQ abend occurs. CICS remains active.
- A CICS abend occurs. The queue manager remains active.

What happens when you stop a queue manager

When a queue manager stops normally, IBM MQ stops all activity in an orderly way. You can stop IBM MQ by using either quiesce, force, or restart mode.

The effects are given in Table 13 on page 121.

Table 13. Stopping a queue manager in QUIESCE, FORCE, and RESTART mode

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

With CICS, a current thread runs only to the end of the unit of work. With CICS, stopping a queue manager in quiesce mode stops the CICS-MQ adapter, and so, if an active task contains more than one unit of work, the task does not necessarily run to completion.

When you stop a queue manager, in any mode, the steps are as follows:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart relative byte address (RBA) is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the IBM MQ bootstrap data set (BSDS) is updated.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create indoubt units of work for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

Quiesce mode does not affect indoubt units of work. Any unit of work that is indoubt remains indoubt.

Automatic reconnection and resynchronization

When CICS is connected to IBM MQ and the queue manager stops, the CICS-MQ adapter tries to reconnect after the stoppage has been detected. The connect request uses the same connect parameters that were used in the previous connect request.

If you have specified a single queue manager for the connection, CICS waits for 10 seconds after the stoppage is detected, and then tries to reconnect. If the queue manager has not been restarted within the 10 seconds, the connect request is deferred until the queue manager is restarted, when CICS reconnects automatically.

If you have specified a queue-sharing group for the connection, CICS tries to reconnect immediately when the stoppage is detected. The actions that CICS takes to restore the connection depend on whether there are outstanding units of work for the last queue manager, and the setting that you have specified for resynchronization.

If you have specified a queue-sharing group for the connection, you can select appropriate resynchronization actions for CICS by using the RESYNCMEMBER attribute of the MQCONN resource definition. Resynchronization takes place when the connection to IBM MQ is lost and CICS is holding outstanding units of work for the last queue manager. You can choose whether CICS waits to reconnect to the same queue manager, or whether CICS makes one attempt to reconnect to the same queue manager, but if that attempt fails, connects to a different eligible queue manager in the group. A queue manager is eligible for connection to a CICS region if it is currently active on the same LPAR as the CICS region.

Alternatively if IBM MQ supports group units of recovery for CICS, you can use the **RESYNCMEMBER(GROUPRESYNC)** option. With the GROUPRESYNC option, CICS connects to any local IBM MQ queue manager in the queue-sharing group and resolves all outstanding indoubt units of work regardless of which queue manager it was previously connected to. To resolve the outstanding units of work CICS must reconnect by using the same MQCONN resource definition that was used for the previous connection; for more information, see How indoubt units of work are resolved by CICS.

What happens when the CICS-MQ adapter restarts

Whenever a connection is broken, the adapter must go through a restart phase during the reconnect process. The restart phase resynchronizes resources. Resynchronization between CICS and IBM MQ enables indoubt units of work to be identified and resolved.

Resynchronization can be caused by these requests:

- An explicit request from the distributed queuing component
- An implicit request when a connection is made to IBM MQ

If the resynchronization is caused by connecting to IBM MQ, the sequence of events is as follows:

1. The connection process obtains a list of unit of work (UOW) IDs that IBM MQ thinks are indoubt.
2. The UOW IDs are displayed on the console in DFHMQ0313I messages.
3. The UOW IDs are passed to CICS.
4. CICS initiates a resynchronization task (CRSY) for each indoubt UOW ID.
5. The result of the task for each indoubt UOW is displayed on the console.

You need to check the messages that are displayed during the connect process:

DFHMQ0313I
Shows that a UOW is in doubt.

DFHMQ0400I
Identifies the UOW and is followed by one of these messages:

- DFHMQ0402I or DFHMQ0403I shows that the UOW was resolved successfully (committed or backed out).

- DFHMQ0404E, DFHMQ0405E, DFHMQ0406E, or DFHMQ0407E shows that the UOW was not resolved.

DFHMQ0409I
Shows that all UOWs were resolved successfully.

DFHMQ0408I
Shows that not all UOWs were resolved successfully.

DFHMQ0314I
Warns that UOW IDs highlighted with a * are not resolved automatically. These UOWs must be resolved explicitly by the distributed queuing component when it is restarted.

The total number of DFHMQ0313I messages should equal the total number of DFHMQ0402I plus DFHMQ0403I messages. If the totals are not equal, the connection process cannot resolve some UOWs. Those UOWs that cannot be resolved are caused by problems with CICS (for example, a cold start) or with IBM MQ, or by distributing queuing. When these problems have been fixed, you can initiate another resynchronization by disconnecting and then reconnecting.

Alternatively, you can resolve each outstanding UOW yourself using the RESOLVE INDOUBT command and the UOW ID shown in message DFHMQ0400I. You must then initiate a disconnect and a connect to clean up the *unit of work descriptors* in CICS. You must know the correct outcome of the UOW to resolve UOWs manually.

All messages that are associated with unresolved UOWs are locked by IBM MQ and no Batch, TSO, or CICS task can access them.

If CICS fails and an emergency restart is necessary, *do not* vary the GENERIC APPLID of the CICS system. If you do, and then reconnect to IBM MQ, data integrity with IBM MQ cannot be guaranteed, because IBM MQ treats the new instance of CICS as a different CICS (because the APPLID is different). Indoubt resolution is then based on the wrong CICS log.

Do not change the setting for RESYNCMEMBER when units of work are outstanding in IBM MQ as this means that the units of work cannot be resolved. A unit of work held in CICS is identified with a resource manager qualifier. When RESYNCMEMBER(GROUPRESYNC) is used the qualifier is the name of the queue-sharing group, otherwise the qualifier used is the name of the individual queue manager.

How indoubt units of work are resolved by CICS

One of the functions of the CICS-MQ adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager ends abnormally while connected to CICS, CICS might commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is *indoubt*.

IBM MQ cannot resolve these indoubt units of work (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

If CICS reconnects to the same IBM MQ queue manager, a process to resolve indoubt units of work is initiated during startup of the CICS-MQ adapter. The steps in this process are as follows:

- The CICS-MQ adapter requests a list of indoubt units of work for this connection ID from IBM MQ.
- The adapter receives the list of indoubt units of work, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each indoubt unit of work.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. If any units of work are shunted indoubt, that is, CICS itself cannot resolve them, they are resolved separately. For more information about shunted units of work, see Shunted units of work

The resolution of indoubt units of work has no effect on CICS resources. IBM MQ considers CICS to be the recovery coordinator, and, when WebSphere MQ restarts, it automatically commits or backs out each unit, depending on whether a log record marked the beginning of the commit. The existence of indoubt objects does not lock CICS resources while WebSphere MQ is being reconnected.

Unresolved units of work

If you are using a IBM MQ queue-sharing group for the connection, and you have specified RESYNCMEMBER(NO) on the MQCONN definition for the connection, CICS makes only one attempt to reconnect to the same queue manager. If that attempt fails, CICS connects to any eligible member of the queue-sharing group. A queue manager is eligible for connection to a CICS region if it is currently active on the same LPAR as the CICS region. When CICS connects to a different queue manager, indoubt units of work cannot be resolved, and CICS issues the warning message DFHMQ2064 if any unresolved units of work remain.

Under some circumstances, CICS reconnects to the same queue manager but cannot run the IBM MQ process to resolve indoubt units of work. In that situation CICS issues one of the following error messages:

- DFHMQ0404E
- DFHMQ0405E
- DFHMQ0406E
- DFHMQ0407E

followed by the message DFHMQ0408I.

If any unresolved units remain after restart, resolve them by the methods described in "How to resolve CICS units of work manually" on page 124.

IBM MQ supports group units of recovery for CICS, and using WebSphere MQ 7.1, or higher, you can use the new option RESYNCMEMBER(GROUPRESYNC). With the GROUPRESYNC option, CICS connects to any local IBM MQ queue manager in the queue-sharing group. The queue manager is chosen by IBM MQ and it asks CICS to resolve all indoubt units of work regardless of which queue manager has the indoubt unit of work. Do not change the setting for RESYNCMEMBER when units of work are outstanding in IBM MQ as this means that the units of work cannot be resolved. A unit of work held in CICS is identified with a resource manager qualifier. When RESYNCMEMBER(GROUPRESYNC) is used the qualifier is the name of the queue-sharing group, otherwise the qualifier used is the name of the individual queue manager.

- If you specify RESYNCMEMBER(GROUPRESYNC) and the previous connection used an MQCONN definition with RESYNCMEMBER(YES) or RESYNCMEMBER(NO) and there are indoubt units of work outstanding in IBM MQ, those indoubt units of work cannot be resolved without reverting to the previous setting of RESYNCMEMBER. CICS issues the warning message DFHMQ2065.
- If you specify RESYNCMEMBER(YES) or RESYNCMEMBER(NO) and the previous connection used an MQCONN definition with RESYNCMEMBER(GROUPRESYNC) and there are indoubt units of work outstanding in IBM MQ, those indoubt units of work cannot be resolved without reverting to the previous setting of RESYNCMEMBER. CICS issues the warning message DFHMQ2066.

How to resolve CICS units of work manually

If the CICS-WebSphere MQ adapter ends abnormally, CICS and WebSphere MQ build indoubt lists either dynamically or during restart, depending on which subsystem caused the abend. When CICS connects to WebSphere MQ, one or more units of work might not have been resolved. Any units of work that CICS cannot resolve must be resolved manually using WebSphere MQ commands.

About this task

If some units of work have not been resolved, one of the following messages is sent to the console:

- DFHMQ0404E
- DFHMQ0405E
- DFHMQ0406E
- DFHMQ0407E
- DFHMQ0408I
- DFHMQ2064

- DFHMQ2065
- DFHMQ2066

CICS retains details of units of work that were not resolved during connection startup. An entry is purged when it no longer appears on the list presented by WebSphere MQ.

Any units of work that CICS cannot resolve must be resolved manually using WebSphere MQ commands. This manual procedure is rarely used, because it is required only where operational errors or software problems have prevented automatic resolution. Any inconsistencies found during indoubt resolution must be investigated.

To resolve the units of work:

Procedure

1. Obtain a list of the units of work from WebSphere MQ using the following command:

   ```
   +CSQ1 DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
   ```

   You receive the following message:

   ```
   CSQM201I +CSQ1 CSQMDRTC  DISPLAY CONN DETAILS
   CONN(BC85772CBE3E0001)
   EXTCONN(C3E2D8C3C7D9F0F94040404040404040)
   TYPE(CONN)
   CONNOPTS(
    MQCNO_STANDARD_BINDING
    )
   UOWLOGDA(2005-02-04)
   UOWLOGTI(10.17.44)
   UOWSTDA(2005-02-04)
   UOWSTTI(10.17.44)
   UOWSTATE(UNRESOLVED)
   NID(IYRCSQ1 .BC8571519B60222D)
   EXTURID(BC8571519B60222D)
   QMURID(0000002BDA50)
   URTYPE(CICS)
   USERID(MQTEST)
   APPLTAG(IYRCSQ1)
   ASID(0000)
   APPLTYPE(CICS)
   TRANSID(GP02)
   TASKNO(0000096)
    END CONN DETAILS
   ```

   For CICS connections, the NID (origin identifier) consists of the CICS applid and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is stored in records written to both the CICS system log and the WebSphere MQ log at syncpoint processing time. This value is referred to in CICS as the *work token*.
2. Scan the CICS log for entries related to a particular unit of work:
   a) Look for a PREPARE record for the task-related installation where the work token field (JCSRMTKN) equals the value obtained from the network ID.

      The network ID is supplied by WebSphere MQ in the DISPLAY CONN command output.

   b) The PREPARE record in the CICS log for the units of work provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

   c) You can use the CICS journal print utility DFHJUP when scanning the log.

      For details of using this program, see Reading log streams using batch jobs (DFHJUP).
3. Scan the WebSphere MQ log for records with the NID related to a particular unit of work. Then use the URID from this record to obtain the rest of the log records for this unit of work.

   You can use the WebSphere MQ print log records program (CSQ1LOGP) to scan the log.

   When scanning the WebSphere MQ log, note that the WebSphere MQ startup message CSQJ001I provides the start RBA for this session.

4. If you need to, perform indoubt resolution in WebSphere MQ. WebSphere MQ can be directed to take the work action for a unit of work using a WebSphere MQ RESOLVE INDOUBT command. To recover all threads associated with a specific *connection-name*, use the NID(*) option.

   The command produces one of the following messages showing whether the thread is committed or backed out:

```
CSQV414I +CSQ1 THREAD network-id COMMIT SCHEDULED
CSQV415I +CSQ1 THREAD network-id ABORT SCHEDULED
```

Results

When performing indoubt resolution, CICS and the adapter are not aware of the commands to WebSphere MQ to commit or back out units of work, because only WebSphere MQ resources are affected. However, CICS keeps details about the indoubt threads that were not resolved by WebSphere MQ. This information is purged either when the list presented is empty or when the list does not include a unit of work of which CICS has details.

When triggering does not work

A program is not triggered if the trigger monitor cannot start the program or the queue manager cannot deliver the trigger message. For example, the applid in the process object must specify that the program is to be started in the background; otherwise, the trigger monitor cannot start the program.

If a trigger message is created but cannot be put on the initiation queue (for example, because the queue is full or the length of the trigger message is greater than the maximum message length specified for the initiation queue), the trigger message is put instead on the dead-letter (undelivered message) queue.

If the put operation to the dead-letter queue cannot complete successfully, the trigger message is discarded and a warning message is sent to the z/OS console or to the system operator or is put on the error log.

Putting the trigger message on the dead-letter queue might generate a trigger message for that queue. This second trigger message is discarded if it adds a message to the dead-letter queue.

If the program is triggered successfully but abends before it receives the message from the queue, use a trace utility (for example, CICS AUXTRACE if the program is running under CICS) to find the cause of the failure.

Chapter 10. Troubleshooting the CICS-MQ bridge

There are certain actions that the CICS-MQ bridge takes in the event of an error, and the actions you can take to identify and resolve problems with the bridge.

How does the CICS-MQ bridge handle errors?

The CICS-MQ bridge allows an IBM MQ application to run a CICS DPL program or a CICS 3270 transaction by putting (MQPUT) an MQ request message on an MQ bridge queue. The CICS-MQ bridge issues an MQGET request to retrieve the message from the queue, and then passes the message content, that is the message without the MQ message headers, to the target program or transaction.

Typically, the application or transaction processes the message and returns its response to the CICS-MQ bridge. If the request message specifies a reply-to queue, the CICS-MQ bridge adds MQ response message headers to this response and puts this MQ response message, through an MQPUT request, on the reply-to queue.

When the application or transaction detects an error (for example, missing or incorrect data in the request), it can return a reply message to the CICS-MQ bridge. The CICS-MQ bridge handles this error response the same way it handles a normal response.

The CICS-MQ bridge can handle some errors that the IBM MQ application cannot or does not handle. For example:

- The message header specifies incorrect character coding.
- The message includes a message header that does not conform to MQ or MQ bridge requirements.
- The transaction or its program is not installed.
- Security credentials for the message do not permit the transaction to proceed.
- An exception condition causes abnormal termination (ABEND) of the application or of the bridge itself.

In general, the CICS-MQ bridge performs the following steps in case of such errors:

1. Write a DFHMQ07nn message to the CSMT transient data queue, the CICS job log, or both.
2. Take a transaction dump.
3. Issue SYNCPOINT ROLLBACK.
4. Handle the request message that the bridge does not retry.

For details, see "What actions does the CICS-MQ bridge perform in case of an error" on page 128.

How does the MQ bridge perform unit of work rollback?

A standard in-syncpoint MQGET is reversible. When the unit of work issues SYNCPOINT, MQ completes removing the message from the queue. When the unit of work issues SYNCPOINT ROLLBACK, MQ returns the message to the queue.

For transient conditions, this mechanism allows the CICS-MQ bridge to recover automatically. MQ returns the request message to the bridge queue and the CICS-MQ bridge gets and processes the same message again.

But for non-transient conditions, the mechanism uses system resources pointlessly, repeatedly reprocessing the same request message until (for example) MQ expiry processing deletes the message.

To help resolve this problem, the CICS-MQ bridge can limit the number of retries. This gives the CICS-MQ bridge an opportunity to recover automatically from transient conditions and avoids endless retries for non-transient conditions.

The backout threshold (BOTHRESH) attribute of the bridge queue specifies the maximum number of retries, that is, the maximum number of times SYNCPOINT ROLLBACK can return the same message to

the bridge queue. The default is zero, which means SYNCPOINT ROLLBACK never returns a message to the bridge queue.

How does the CICS-MQ bridge handle messages that it does not retry?

When the CICS-MQ bridge cannot simply put a message back to the bridge queue (for example, a message with a backout count greater than the backout threshold), the bridge will dispose of the message as follows:

1. The bridge first tries to put the message to the backout requeue queue, if a backout requeue queue has been defined to the queue manager.
2. When step 1 fails or is not possible, the bridge tries to use the dead letter queue instead if one has been defined to the queue manager.
3. When step 2 fails or is not possible, the bridge checks the persistence attribute of the message.
   - If the message is not persistent, the bridge discards the message
   - If the message is persistent, the bridge abnormally terminates.

How can you debug the CICS-MQ bridge?

When the CICS-MQ bridge does not process the messages as expected, or the bridge task abends or ends unexpectedly, you can follow the troubleshooting guidelines in “Debugging the CICS-MQ bridge” on page 130 to debug the bridge.

What actions does the CICS-MQ bridge perform in case of an error

When the CICS-MQ bridge detects an error, it will perform a series of actions.

Figure 12 on page 129 illustrates the series of actions CICS-MQ performs to handle an error.

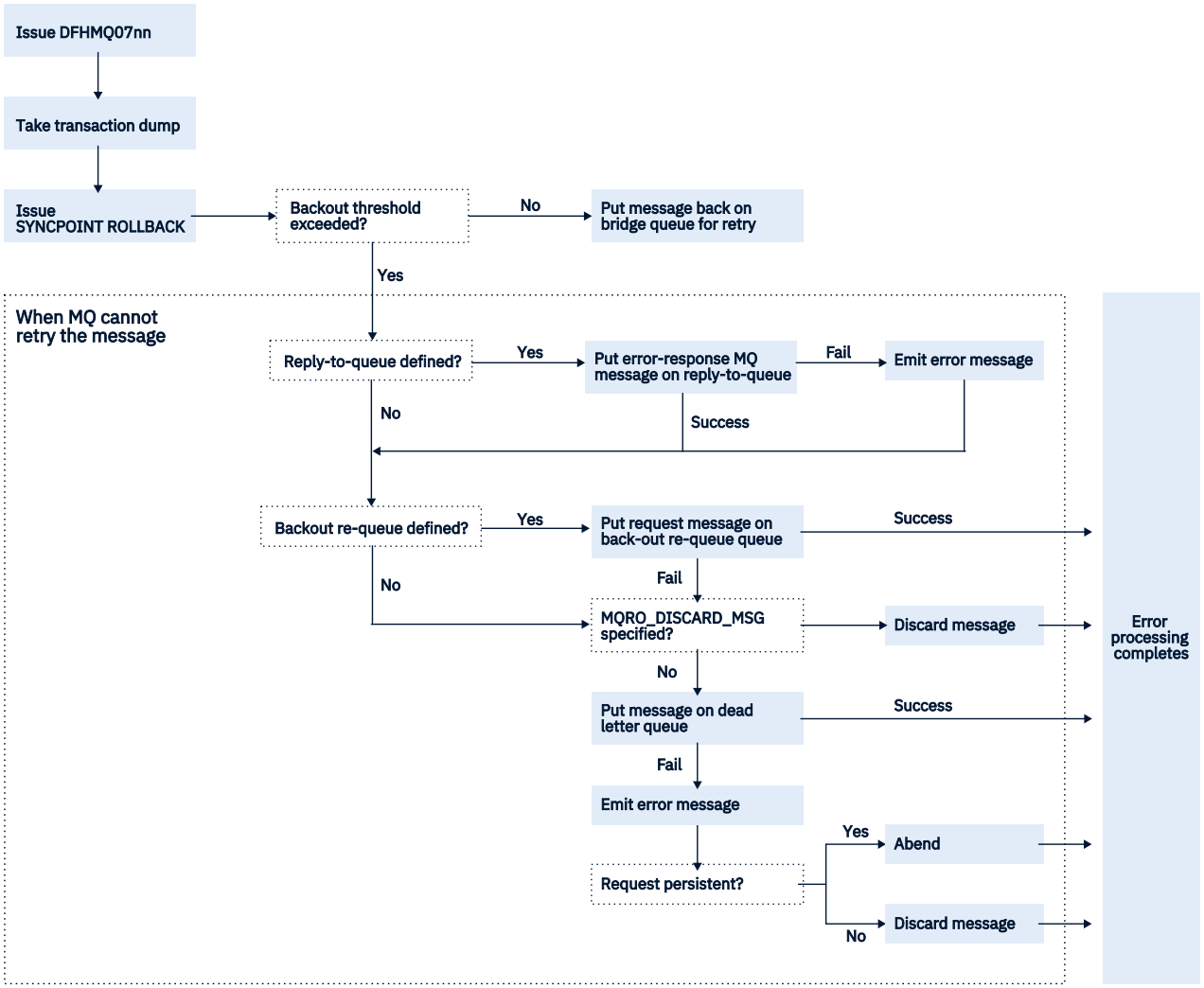


Figure 12. CICS-MQ error handling workflow

When an error occurs, the CICS-MQ bridge performs the following actions:

1. Write a DFHMQ07nn message to the CSMT transient data queue, the CICS job log, or both.
2. Take a transaction dump.
3. Issue SYNCPOINT ROLLBACK.
4. If the backout count for the message is not yet at the backout threshold, the SYNCPOINT ROLLBACK processing puts the request message back on the bridge queue so that the CICS-MQ bridge can get the request message again and retry processing it. Processing for this error is complete.

   If the backout count for the message exceeds the backout threshold, continue with step "5" on page 130 and step "6" on page 130.
5. If the request message specifies a reply-to queue, build an error-response MQ message and put it on the reply-to queue. If the put operation for this error-response message fails, emit an error message. Processing for this error is complete.

   Note:

   - This error-response MQ message includes an MQCIH even if the request message does not.
   - The CICS-MQ bridge checks for and processes the MQCIH_PASS_EXPIRATION and MQRO_PASS_DISCARD_AND_EXPIRY options in the request message.
6. If there is a backout re-queue queue defined for the bridge queue, put the request message on the back-out re-queue queue. If the put operation is successful, processing for this error is complete.

   If the put operation fails or if there is no backout re-queue queue defined for the bridge queue, continue with step "7" on page 130.
7. Put the message on the dead letter queue unless the request message specifies MQRO_DISCARD_MSG, which instructs the bridge not to put the message on the dead letter queue.

   If the put operation fails, the bridge emits an error message, and either terminates abnormally (ABEND) if the request is persistent, or discards the message if the message is nonpersistent.

   Processing for this error is complete.

   Note: The request message is added with an MQDLH when put on the dead letter queue. The resulting message might be truncated if it is too long for the dead letter queue.

Debugging the CICS-MQ bridge

When the CICS-MQ bridge does not process the messages as expected, or the bridge task abends or ends unexpectedly, these troubleshooting guidelines help you debug the bridge.

What's the error that occurred?

- "Message is PUT to the bridge request queue, but is not processed by the bridge monitor" on page 130
- "Inbound message is taken from the request queue by the bridge monitor, but the CICS DPL program or CICS transaction fails to run" on page 131
- "The CICS-MQ bridge task abends" on page 131
- "The CICS-MQ bridge monitor ends unexpectedly" on page 132

Message is PUT to the bridge request queue, but is not processed by the bridge monitor

1. Check that the bridge monitor is running. Issue **CEMT I TASK** and look for CKBR or whatever other transaction identifier you are using for the bridge monitor.

   If the bridge monitor is not running and you are expecting it to be triggered, make sure that the triggering options on the bridge request queue are correct. Use a queue attribute of TRIGTYPE(FIRST).

If the bridge monitor was running but is no longer running, check the output in the CICS CSMT and joblog on all CICS regions where you expect bridge monitors to be running, to see if an error has caused the bridge monitor to terminate.

2. If the bridge request queue is defined with QSGDISP(SHARED), check that it also specifies INDXTYPE(CORRELID).
3. Browse the inbound message that is not being processed and check that the values of MQMD.MsgId and MQMD.CorrelId are correct. If this is the first message in a unit of work or a pseudoconversation, MQMD.CorrelId must be set to a value of MQCI_NEW_SESSION and MQMD.MsgId must be set to MQMI_NONE (binary zeros).
4. If this message is not the first one in a unit of work or pseudoconversation, ensure that your application has checked previous reply messages adequately for possible errors. As a minimum, it should check the following fields in the MQCIH:
   - MQCIH.ReturnCode
   - MQCIH.CompCode
   - MQCIH.TaskEndStatus
   - MQCIH.AbendCode
   - MQCIH.ErrorOffset

Go to top...

Inbound message is taken from the request queue by the bridge monitor, but the CICS DPL program or CICS transaction fails to run

1. Check the output in the CICS MSGUSR log. This output nearly always reports the reason why the DPL program or transaction failed to run. The common reasons for this are as follows:
   - Program or transaction not defined to CICS. Use CEDA to define the program or transaction and run your bridge application again.
   - Insufficient authority to run the program or transaction. Details of how to control the level of authentication used by the CICS-MQ bridge are given in Setting up the CICS-MQ bridge.
2. Check the message that is sent to the reply queue by the bridge monitor. If an error has occurred, it is likely that the MQCIH.Format field is set to MQFMT_STRING and an error message is appended to the MQCIH in place of a vector.
3. Check the dead letter queue to see if a reply message has been sent there by the bridge monitor. If it has, and the values of MQMD.MsgId and MQMD.CorrelId are correct, check the value of MQDLH.Reason. This value is typically set to a feedback code that indicates the reason for the failure.

   For information on feedback codes, including those specific to the CICS-MQ bridge, see CICS messages.

Go to top...

The CICS-MQ bridge task abends

Abend codes are set in outbound messages in field MQCIH.AbendCode. In addition, the output in the CICS MSGUSR log reports abend codes for failing bridge tasks.

Abends are documented in CICS messages.

You can deal with some common abend codes as follows:

ABRG
An invalid bridge facility token was specified in an inbound message. Your first inbound message must always specify a value of MQCFAC_NONE in field MQCIH.Facility, and a nonzero value in MQCIH.FacilityKeepTime. CICS returns a facility token in field MQCIH.Facility, and you can use this value in all subsequent inbound messages in the pseudoconversation.

ABXH
Either `brmq_re_buffer_indicator` was set to N, when a receive with the buffer option was specified, or `brmq_re_buffer_indicator` was set to Y and a receive (without the buffer option) specified.

MBRJ
The MQCIH has invalid data. Check the values in the MQCIH field by field to find the one that is out of range. MBRJ can also be caused by a length mismatch; for example, when the value of `brmq_vector_length` and the length of the data vector do not agree, or the CICS headers and vector do not contain enough data.

MBRN
The message is shorter than expected. Every vector structure has one or two data length fields. The first is the first fullword field in the standard header for all vectors, and it should be equal to the overall length of the vector including the variable length data. Some vectors also contain another fullword length field that gives just the length of the variable length data. If these values indicate more data than there is, the bridge task abends with MBRN.

MBRO and MBRP
The vector structure (not the variable length data) contains an error. The MQCIH field ERROROFFSET gives the offset of the field in error. Check the values of the fields in the vector against the permitted values, as described in Link3270 message formats.

Go to top...

The CICS-MQ bridge monitor ends unexpectedly

Some errors can cause the bridge monitor transaction, CKBR, to end unexpectedly. If you are using triggered queues to start the monitor, and messages remain on the bridge request queue, the CKTI transaction might try to restart CKBR. If the original error persists, CKBR failures might loop. To halt the loop, set off the `TriggerControl` attribute of the request queue while you diagnose and fix the underlying problem.

The bridge monitor can fail if it does not have sufficient authority to access the queues or CICS transactions, if it cannot write to the dead letter queue or it encounters problems when running CICS or IBM MQ services.

Go to top...

Chapter 11. MQCIH – CICS-MQ bridge header

The MQCIH structure describes the information that can be present at the start of a message sent to the CICS-MQ bridge through IBM MQ for z/OS.

Availability

AIX, HP-UX, z/OS, Solaris, Linux, Windows, and IBM MQ clients connected to these systems.

For C++ applications, the ImqCICSBridgeHeader class encapsulates specific features of the MQCIH data structure. See “IBM MQ C++ message header for the CICS-MQ bridge” on page 145.

Format name

MQFMT_CICS.

Version

The current version of MQCIH is MQCIH_VERSION_2. The header, COPY, and INCLUDE files provided for the supported programming languages contain the most recent version of MQCIH, with the initial value of the Version field set to MQCIH_VERSION_2.

The fields CursorPosition, ErrorOffset, InputItem, and Reserved4 are not present if the version is less than MQCIH_VERSION_2.

Character set and encoding

Special conditions apply to the character set and encoding used for the MQCIH structure and application message data:

- Applications that connect to the queue manager that owns the CICS-MQ bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager, because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the CICS-MQ bridge queue converts the MQCIH structure.
- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. You cannot use the CodedCharSetId and Encoding fields in the MQCIH structure to specify the character set and encoding of the application message data. You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Usage

If the application requires values that are the same as the initial values, and the bridge is running with AUTH=LOCAL or AUTH=IDENTIFY, you can omit the MQCIH structure from the message. In all other cases, the structure must be present.

The bridge accepts either a version-1 or a version-2 MQCIH structure, but for 3270 transactions, you must use a version-2 structure.

The application must ensure that fields documented as request fields have appropriate values in the message sent to the bridge; these fields are used as input to the bridge.

Fields documented as response fields are set by the CICS-MQ bridge in the reply message that the bridge sends to the application. Error information is returned in the ReturnCode, Function, CompCode, Reason, and AbendCode fields, but not all of them are set in all cases. The following table shows which fields are set for different values of ReturnCode.

Table 14. Contents of error information fields in MQCIH structure				
ReturnCode	***Function***	***CompCode***	***Reason***	***AbendCode***
MQCRC_OK	–	–	–	–
MQCRC_BRIDGE_ERROR	–	–	MQFB_CICS_*	–
MQCRC_MQ_API_ERROR MQCRC_BRIDGE_TIMEOUT	MQ call name	MQ *CompCode*	MQ *Reason*	–
MQCRC_CICS_EXEC_ERROR MQCRC_SECURITY_ERROR MQCRC_PROGRAM_NOT_AVAILABLE MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	–
MQCRC_BRIDGE_ABEND MQCRC_APPLICATION_ABEND	–	–	–	CICS ABCODE

Initial values and language declarations

This table shows the initial values of the fields in the CICS-MQ bridge header (MQCIH) and their order in the MQCIH structure.

Table 1. Initial values of fields in MQCIH		
Field name	**Name of constant**	**Value of constant**
StrucId	MQCIH_STRUC_ID	'CIHb'
Version	MQCIH_VERSION_2	2
StrucLength	MQCIH_LENGTH_2	180
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQCIH_NONE	0
ReturnCode	MQCRC_OK	0
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
UOWControl	MQCUOWC_ONLY	273
GetWaitInterval	MQCGWI_DEFAULT	-2
LinkType	MQCLT_PROGRAM	1
OutputDataLength	MQCODL_AS_INPUT	-1
FacilityKeepTime	None	0
ADSDescriptor	MQCADSD_NONE	0
ConversationalTask	MQCCT_NO	0
TaskEndStatus	MQCTES_NOSYNC	0
Facility	MQCFAC_NONE	Nulls
Function	MQCFUNC_NONE	Blanks

Table 1. Initial values of fields in MQCIH

(continued)

Field name	Name of constant	Value of constant
AbendCode	None	Blanks
Authenticator	None	Blanks
Reserved1	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
RemoteSysId	None	Blanks
RemoteTransId	None	Blanks
TransactionId	None	Blanks
FacilityLike	None	Blanks
AttentionId	None	Blanks
StartCode	MQCSC_NONE	Blanks
CancelCode	None	Blanks
NextTransactionId	None	Blanks
Reserved2	None	Blanks
Reserved3	None	Blanks
CursorPosition	None	0
ErrorOffset	None	0
InputItem	None	0
Reserved4	None	0

Notes:

1. The symbol ␣ represents a single blank character.
2. In the C programming language, the macro variable MQCIH_DEFAULT contains the values listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQCIH MyCIH = {MQCIH_DEFAULT};
```

AbendCode (MQCHAR4)

The value returned in this field is significant only if the ReturnCode field has the value MQCRC_APPLICATION_ABEND or MQCRC_BRIDGE_ABEND. If it does, AbendCode contains the CICS ABCODE value.

This field is a response field. The length of this field is given by MQ_ABEND_CODE_LENGTH. The initial value of this field is 4 blank characters.

ADSDescriptor (MQLONG)

This indicator specifies whether to send ADS descriptors on SEND and RECEIVE BMS requests.

The following values are defined:

MQCADSD_NONE
Do not send or receive ADS descriptors.

MQCADSD_SEND
Send ADS descriptors.

MQCADSD_RECV
Receive ADS descriptors.

MQCADSD_MSGFORMAT
Use message format for the ADS descriptors. This option sends or receives the ADS descriptors using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

Set the *ADSDescriptor* field as follows:

- If you are not using ADS descriptors, set the field to MQCADSD_NONE.
- If you are using ADS descriptors with the *same* CCSID in each environment, set the field to the sum of MQCADSD_SEND and MQCADSD_RECV.
- If you are using ADS descriptors with *different* CCSIDs in each environment, set the field to the sum of MQCADSD_SEND, MQCADSD_RECV, and MQCADSD_MSGFORMAT.

This is a request field used only for 3270 transactions. The initial value of this field is MQCADSD_NONE.

AttentionId (MQCHAR4)

This field is the initial value of the AID key when the transaction is started.

The value is a 1-byte value, left justified.

This request field is used only for 3270 transactions. The length of this field is given by MQ_ATTENTION_ID_LENGTH. The initial value of this field is 4 blanks.

Authenticator (MQCHAR8)

This field holds a password or passticket. If user-identifier authentication is active for the CICS-MQ bridge, Authenticator is used with the user identifier in the MQMD identity context to authenticate the sender of the message.

This field is a request field. The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is 8 blanks.

CancelCode (MQCHAR4)

This field holds the abend code that is used to stop the transaction (typically a conversational transaction that is requesting more data). Otherwise, this field is set to blanks.

This field is a request field used only for 3270 transactions. The length of this field is given by MQ_CANCEL_CODE_LENGTH. The initial value of this field is 4 blanks.

CodedCharSetId (MQLONG)

This field is reserved; its value is not significant.

The initial value of this field is 0.

CompCode (MQLONG)

The value returned in this field depends on ReturnCode.

See Table 14 on page 134.

This field is a response field. The initial value of this field is MQCC_OK

ConversationalTask (MQLONG)

This indicator specifies whether to allow the task to issue requests for more information, or to abend the task.

The value must be one of the following:

MQCCT_YES
Task is conversational. Specify this value if multiple messages can be used to supply vectors for the translation.

MQCCT_NO
Task is not conversational. Specify this value if all the input vectors needed for this CICS transaction are supplied in the input message.

This request field is used only for 3270 transactions. The initial value of this field is MQCCT_NO.

CursorPosition (MQLONG)

This field holds the initial cursor position when the transaction is started. Subsequently, for conversational transactions, the cursor position is in the RECEIVE vector.

This request field is used only for 3270 transactions. The initial value of this field is 0. This field is not present if Version is less than MQCIH_VERSION_2.

Encoding (MQLONG)

This field is reserved; its value is not significant.

The initial value of this field is 0.

ErrorOffset (MQLONG)

This field holds the position of invalid data detected by the bridge exit. The field provides the offset from the start of the message to the location of the invalid data.

This response field is used only for 3270 transactions. The initial value of this field is 0. This field is not present if Version is less than MQCIH_VERSION_2.

Facility (MQBYTE8)

This field holds an 8-byte bridge facility token. A bridge facility token allows multiple transactions in a pseudo-conversation to use the same bridge facility (virtual 3270 terminal).

In the first, or only, message in a pseudoconversation, set a value of MQCFAC_NONE; this value instructs CICS to allocate a new bridge facility for this message. A bridge facility token is returned in reply messages when a nonzero FacilityKeepTime value is specified on the input message. Subsequent input messages in a pseudoconversation must then use the same bridge facility token.

The following special value is defined:

MQCFAC_NONE
No facility token specified.

For the C programming language, the constant MQCFAC_NONE_ARRAY is also defined; this constant has the same value as MQCFAC_NONE, but is an array of characters instead of a string.

This field is both a request and a response field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LENGTH. The initial value of this field is MQCFAC_NONE.

FacilityKeepTime (MQLONG)

This field specifies the length of time in seconds that the bridge facility is kept after the user transaction ends.

Specify a value that corresponds to the expected duration of a pseudoconversation. For pseudo-nonconversational transactions, the value must be zero. For nonconversational transactions, the value should be zero.

This request field is used only for 3270 transactions. The initial value of this field is 0.

FacilityLike (MQCHAR4)

This field specifies the name of an installed terminal that is to be used as a model for the bridge facility.

A value of blanks means that FacilityLike is taken from the bridge transaction profile definition, or a default value is used.

This request field is used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LIKE_LENGTH. The initial value of this field is 4 blanks.

Flags (MQLONG)

This field holds any flags for the message.

The value of the field must be one of the following:

MQCIH_NONE
No flags.

MQCIH_PASS_EXPIRATION
The reply message contains:

- The same expiry report options as the request message
- The remaining expiry time from the request message with no adjustment made for the processing time of the bridge

If you omit this value, the expiry time is set to unlimited.

MQCIH_REPLY_WITHOUT_NULLS
The reply message length of a CICS DPL program request is adjusted to exclude trailing nulls (X'00') at the end of the COMMAREA returned by the DPL program. If this value is not set, the nulls might be significant and the full COMMAREA is returned.

MQCIH_SYNC_ON_RETURN
The CICS link for DPL requests uses the SYNCONRETURN option. This option causes CICS to take a sync point when the program completes if it is shipped to another CICS region. The bridge does not specify to which CICS region to ship the request; that is controlled by the CICS program definition or workload balancing facilities.

This field is a request field. The initial value of this field is MQCIH_NONE.

Format (MQCHAR8)

This field holds the IBM MQ format name of the data that follows the MQCIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the Format field in MQMD.

This format name is also used for the reply message, if the ReplyToFormat field has the value MQFMT_NONE.

- For DPL requests, Format must be the format name of the COMMAREA.
- For 3270 requests, Format must be DFHMQDCI or CSCQBDCI, and the bridge sets the format to DFHMQDCO or CSCQBDCO for Reply messages.

The data-conversion exits for these formats must be installed on the queue manager where they are to run.

If the request message generates an error reply message, the error reply message has a format name of MQFMT_STRING.

This field is a request field. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Function (MQCHAR4)

The value returned in this field depends on ReturnCode.

See Table 14 on page 134. The following values are possible when Function contains an IBM MQ call name:

MQCFUNC_MQCONN
MQCONN call

MQCFUNC_MQGET
MQGET call

MQCFUNC_MQINQ
MQINQ call

MQCFUNC_MQOPEN
MQOPEN call

MQCFUNC_MQPUT
MQPUT call

MQCFUNC_MQPUT1
MQPUT1 call

MQCFUNC_NONE
No call

In all cases, for the C programming language the constants MQCFUNC_*_ARRAY are also defined; these constants have the same values as the corresponding MQCFUNC_* constants, but are arrays of characters instead of strings.

This field is a response field. The length of this field is given by MQ_FUNCTION_LENGTH. The initial value of this field is MQCFUNC_NONE.

GetWaitInterval (MQLONG)

This field applies only when UOWControl has the value MQCUOWC_FIRST. It allows the sending application to specify the approximate time in milliseconds that the MQGET calls issued by the bridge

wait for second and subsequent request messages for the unit of work started by this message. This value overrides the default wait interval used by the bridge.

You can use the following special values:

MQCGWI_DEFAULT
Default wait interval. This value causes the CICS-MQ bridge to wait for the period of time specified when the bridge was started.

MQWI_UNLIMITED
Unlimited wait interval.

This field is a request field. The initial value of this field is MQCGWI_DEFAULT.

InputItem (MQLONG)

This field is reserved. The value must be 0.

This field is not present if Version is less than MQCIH_VERSION_2.

LinkType (MQLONG)

This field indicates the type of object that the bridge tries to link.

The value must be one of the following:

MQCLT_PROGRAM
DPL program

MQCLT_TRANSACTION
3270 transaction

This field is a request field. The initial value of this field is MQCLT_PROGRAM.

NextTransactionId (MQCHAR4)

This field holds the name of the next transaction returned by the user transaction (normally by **EXEC CICS RETURN TRANSID**). If no next transaction exists, this field is set to blanks.

This response field is used only for 3270 transactions. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is 4 blanks.

OutputDataLength (MQLONG)

This field holds the length of the user data to be returned to the client in a reply message. This length includes the 8-byte program name.

If your request is passing data to the user program in a COMMAREA then the user program can return response data in the COMMAREA. The response data can be longer than the request data. If it is, then you must specify the length of the response data in OutputDataLength. This length includes the 8-byte program name.

If your request is passing data to the user program in a channel/container, then the user program returns any response data in a container. The container itself indicates the length of the data it holds, so that OutputDataLength is not used.

Note: The length of the user data in a message is the length of the message *excluding* the MQCIH structure.

If the length of the user data in the request message is smaller than OutputDataLength, the DATALENGTH option of the LINK command is used; this option allows the LINK to be function-shipped efficiently to another CICS region.

You can use the following special value:

MQCODL_AS_INPUT
Output length is same as input length.

This value might be needed even if no reply is requested, to ensure that the COMMAREA passed to the linked program is of sufficient size.

This request field is used only for DPL programs. The initial value of this field is MQCODL_AS_INPUT.

Reason (MQLONG)

The value returned in this field depends on ReturnCode.

See Table 14 on page 134.

This field is a response field. The initial value of this field is MQRC_NONE.

RemoteSysId (MQCHAR4)

This field holds the CICS system identifier of the CICS system that is processing the request. If this field is blank, the CICS system request is processed on the same CICS system as the bridge monitor. The SYSID used is returned in the Reply message.

For a 3270 pseudoconversation, all subsequent messages in the conversation must specify the remote SYSID returned in the initial reply. If specified, the SYSID must have these characteristics:

- Be active
- Have access to the IBM MQ Request queue
- Be accessible by the CICS ISC links from the CICS system where the bridge monitor is running

RemoteTransId (MQCHAR4)

This field is an optional request field. If specified, the field is used as the RTRANSID value of CICS START.

The length of this field is given by MQ_TRANSACTION_ID_LENGTH.

ReplyToFormat (MQCHAR8)

This field holds the IBM MQ format name of the reply message that is sent in response to the current message.

If your request is passing data to the user program in a COMMAREA, then the user program can return response data in the COMMAREA. The response data can be binary or character format; ReplyToFormat must contain the WebSphere® MQ format name to indicate which.

If your request is passing data to the user program in a channel and container, then the user program returns any response data in a container. The container itself indicates the format of the data it holds, so that ReplyToFormat is not used.

The rules for coding this field are the same as the rules for the Format field in MQMD.

This request field is used only for DPL programs. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Reserved1 (MQCHAR8)

This field is reserved. The value must be 8 blanks.

Reserved2 (MQCHAR8)

This field is reserved. The value must be 8 blanks.

Reserved3 (MQCHAR8)

This field is reserved. The value must be 8 blanks.

Reserved4 (MQLONG)

This field is reserved. The value must be 0.

This field is not present if Version is less than MQCIH_VERSION_2.

ReturnCode (MQLONG)

This field holds the return code from the CICS bridge monitor describing the outcome of the processing performed by the bridge.

The Function, CompCode, Reason, and AbendCode fields might contain additional information. See Table 14 on page 134.

The value is one of the following:

MQCRC_APPLICATION_ABEND
(5, X'005') Application ended abnormally.

MQCRC_BRIDGE_ABEND
(4, X'004') CICS bridge ended abnormally.

MQCRC_BRIDGE_ERROR
(3, X'003') CICS bridge detected an error.

MQCRC_BRIDGE_TIMEOUT
(8, X'008') Second or later message in the current unit of work not received within the specified time.

MQCRC_CICS_EXEC_ERROR
(1, X'001') EXEC CICS statement detected an error.

MQCRC_MQ_API_ERROR
(2, X'002') MQ call detected an error.

MQCRC_OK
(0, X'000') No error.

MQCRC_PROGRAM_NOT_AVAILABLE
(7, X'007') Program not available.

MQCRC_SECURITY_ERROR
(6, X'006') Security error occurred.

MQCRC_TRANSID_NOT_AVAILABLE
(9, X'009') Transaction not available.

This field is a response field. The initial value of this field is MQCRC_OK.

StartCode (MQCHAR4)

This indicator specifies whether the bridge emulates a terminal transaction or a transaction initiated with START.

The value must be one of the following:

MQCSC_START
Start

MQCSC_STARTDATA
Start data

MQCSC_TERMINPUT
Terminal input

MQCSC_NONE
None

In all cases, for the C programming language the constants MQCSC_*_ARRAY are also defined; these have the same values as the corresponding MQCSC_* constants, but are arrays of characters instead of strings.

In the response from the bridge, this field is set to the start code appropriate to the next transaction ID contained in the NextTransactionId field. The following start codes are possible in the response:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT

This field is both a request and a response field.

This field is used only for 3270 transactions. The length of this field is given by MQ_START_CODE_LENGTH. The initial value of this field is MQCSC_NONE.

StrucId (MQCHAR4)

This field holds an identifier for the CICS information header structure.

The value must be:

MQCIH_STRUC_ID
Identifier for CICS information header structure.

For the C programming language, the constant MQCIH_STRUC_ID_ARRAY is also defined; this constant has the same value as MQCIH_STRUC_ID, but is an array of characters instead of a string.

This field is a request field. The initial value of this field is MQCIH_STRUC_ID.

StrucLength (MQLONG)

This field specifies the length of the CICS information header structure.

The value must be one of the following:

MQCIH_LENGTH_1
Length of version-1 CICS information header structure.

MQCIH_LENGTH_2
Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

MQCIH_CURRENT_LENGTH
Length of current version of CICS information header structure.

This field is a request field. The initial value of this field is MQCIH_LENGTH_2.

TaskEndStatus (MQLONG)

This field shows the status of the user transaction at end of task.

One of these values is returned:

MQCTES_NOSYNC
Not synchronized.

The user transaction has not yet completed and has not taken a sync point. The MsgType field in MQMD is MQMT_REQUEST in this case.

MQCTES_COMMIT
Commit unit of work.

The user transaction has not yet completed, but has taken a sync point for the first unit of work. The MsgType field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_BACKOUT
Back out unit of work.

The user transaction has not yet completed. The current unit of work will be backed out. The MsgType field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_ENDTASK
End task.

The user transaction has ended (or abended). The MsgType field in MQMD is MQMT_REPLY in this case.

This field is a response field used only for 3270 transactions. The initial value of this field is MQCTES_NOSYNC.

TransactionId (MQCHAR4)

This field supplies a transaction identifier of the user transaction to be run in CICS, or a transaction code under which CICS programs are to be run.

If the LinkType field has the value `MQCLT_TRANSACTION`, TransactionId is the transaction identifier of the user transaction to be run; specify a nonblank value in this case.

If the LinkType field has the value `MQCLT_PROGRAM`, TransactionId is the transaction code under which all programs in the unit of work are to be run. If you specify a blank value, the CICS DPL bridge default transaction code (CKBP) is used. If the value is nonblank, you must either specify CKBP, or CKBC, or a local transaction ID that you have defined to CICS with DFHMQBP0 or DFHMQBP3 as its initial program. The TransactionId field applies only when the UOWControl field has the value `MQCUOWC_FIRST` or `MQCUOWC_ONLY`.

The TransactionId field is a request field. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is 4 blanks.

UOWControl (MQLONG)

This field controls the unit-of-work processing performed by the CICS bridge.

You can request the bridge to run a single transaction or one or more programs in a unit of work. The field indicates whether the CICS bridge starts a unit of work, performs the requested function in the current unit of work, or ends the unit of work by committing it or backing it out. Various combinations are supported, to optimize the data transmission flows.

The value must be one of the following:

MQCUOWC_ONLY
Start unit of work, perform function, then commit the unit of work.

MQCUOWC_CONTINUE
Additional data for the current unit of work (3270 only).

MQCUOWC_FIRST
Start unit of work and perform function.

MQCUOWC_MIDDLE
Perform function in current unit of work

MQCUOWC_LAST
Perform function, then commit the unit of work.

MQCUOWC_COMMIT
Commit the unit of work (DPL only).

MQCUOWC_BACKOUT
Back out the unit of work (DPL only).

This field is a request field. The initial value of this field is MQCUOWC_ONLY.

Version (MQLONG)

This field identifies the version of the CICS information header structure.

The value must be one of the following:

MQCIH_VERSION_1
Version-1 CICS information header structure.

MQCIH_VERSION_2
Version-2 CICS information header structure.

Fields that exist only in the more recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCIH_CURRENT_VERSION
Current version of CICS information header structure.

This field is a request field. The initial value of this field is MQCIH_VERSION_2.

IBM MQ C++ message header for the CICS-MQ bridge

C++ applications that send messages to the CICS-MQ bridge through IBM MQ for z/OS use the ImqCICSBridgeHeader class to interact with the MQCIH data structure.

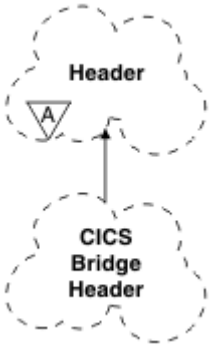
The following example shows how to add CICS-MQ bridge header information to a message:

```
ImqQueueManager mgr ;          // The queue manager.
ImqQueue queueIn ;             // Incoming message queue.
ImqQueue queueBridge ;         // CICS-MQ bridge message queue.
ImqMessage msg ;               // Incoming and outgoing message.
ImqCicsBridgeHeader header ;   // CICS-MQ bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS-MQ bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS-MQ bridge header information.  This will vary
```

```
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS-MQ bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

ImqCICSBridgeHeader class

The ImqCICSBridgeHeader class encapsulates specific features of the MQCIH data structure. Objects of this class are used by applications that send messages to the CICS-MQ bridge through IBM MQ for z/OS.



The object attributes for this class are as follows:

Figure 13. ImqCICSBridgeHeader class

ADS descriptor
Send/receive ADS descriptor. Set the descriptor with MQCADSD_NONE. The initial value is MQCADSD_NONE. The following additional values are possible:

- MQCADSD_NONE
- MQCADSD_SEND
- MQCADSD_RECV
- MQCADSD_MSGFORMAT

attention identifier
AID key. The field must be of length MQ_ATTENTION_ID_LENGTH.

authenticator
RACF password or PassTicket. The initial value contains blanks, of length MQ_AUTHENTICATOR_LENGTH.

bridge abend code
Bridge abend code, of length MQ_ABEND_CODE_LENGTH. The initial value is four blank characters. The value returned in this field depends on the return code.

bridge cancel code
Bridge abend transaction code. The field is reserved, must contain blanks, and be of length MQ_CANCEL_CODE_LENGTH.

bridge completion code
Completion code, which can contain either the IBM MQ completion code or the CICS EIBRESP value. The field has the initial value of MQCC_OK. The value returned in this field depends on the return code.

bridge error offset
Bridge error offset. The initial value is zero. This attribute is read-only.

bridge reason code
Reason code. This field can contain either the IBM MQ reason or the CICS EIBRESP2 value. The field has the initial value of MQRC_NONE. The value returned in this field depends on the return code.

bridge return code
Return code from the bridge. The initial value is MQCRC_OK.

conversational task
Whether the task can be conversational. The initial value is MQCCT_NO. The following additional values are possible:

- MQCCT_YES
- MQCCT_NO

cursor position
Cursor position. The initial value is zero.

facility keep time
CICS bridge facility release time.

facility like
Terminal emulated attribute. The field must be of length MQ_FACILITY_LIKE_LENGTH.

facility token
BVT token value. The field must be of length MQ_FACILITY_LENGTH. The initial value is MQCFAC_NONE.

function
Function, which can contain either the IBM MQ call name or the CICS EIBFN function. The field has the initial value of MQCFUNC_NONE, with length MQ_FUNCTION_LENGTH. The value returned in this field depends on the return code.

The following additional values are possible when function contains an IBM MQ call name:

- MQCFUNC_MQCONN
- MQCFUNC_MQGET
- MQCFUNC_MQINQ
- MQCFUNC_NONE
- MQCFUNC_MQOPEN
- MQCFUNC_PUT
- MQCFUNC_MQPUT1

get wait interval
Wait interval for an MQGET call issued by the bridge task. The initial value is MQCGWI_DEFAULT. The field applies only when UOW control has the value MQCUOWC_FIRST. The following additional values are possible:

- MQCGWI_DEFAULT
- MQWI_UNLIMITED

link type
Link type. The initial value is MQCLT_PROGRAM. The following additional values are possible:

- MQCLT_PROGRAM
- MQCLT_TRANSACTION

next transaction identifier
ID of the next transaction to attach. The field must be of length MQ_TRANSACTION_ID_LENGTH.

output data length
COMMAREA data length. The initial value is MQCODL_AS_INPUT.

reply-to format
Format name of the reply message. The initial value is MQFMT_NONE with length MQ_FORMAT_LENGTH.

start code
Transaction start code. The field must be of length MQ_START_CODE_LENGTH. The initial value is MQCSC_NONE. The following additional values are possible:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT
- MQCSC_NONE

task end status
Task end status. The initial value is MQCTES_NOSYNC. The following additional values are possible:

- MQCTES_COMMIT
- MQCTES_BACKOUT
- MQCTES_ENDTASK
- MQCTES_NOSYNC

transaction identifier
ID of the transaction to attach. The initial value must contain blanks, and must be of length MQ_TRANSACTION_ID_LENGTH. The field applies only when UOW control has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

UOW control
UOW control. The initial value is MQCUOWC_ONLY. The following additional values are possible:

- MQCUOWC_FIRST
- MQCUOWC_MIDDLE
- MQCUOWC_LAST
- MQCUOWC_ONLY
- MQCUOWC_COMMIT
- MQCUOWC_BACKOUT
- MQCUOWC_CONTINUE

version
The MQCIH version number. The initial value is MQCIH_VERSION_2. The only other supported value is MQCIH_VERSION_1.

Notices

This information was developed for products and services offered in the U.S.A. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 rights 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, MD-NC119
Armonk, NY 10504-1785
United States of America

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

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America

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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 5 are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS security
- Developing for external interfaces
- Reference: application developmenth
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 5 , but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 5 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services
- Customization Guide

- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- Supplied Transactions
- CICSPlex SM Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java™ Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 5 , but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Spring Boot is a trademark of Pivotal Software, Inc. in the U.S. and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (main interface):

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):

Depending upon the configurations deployed, this Software Offering may use session cookies that collect no personally identifiable information. These cookies cannot be disabled.

For CICS Explorer:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W