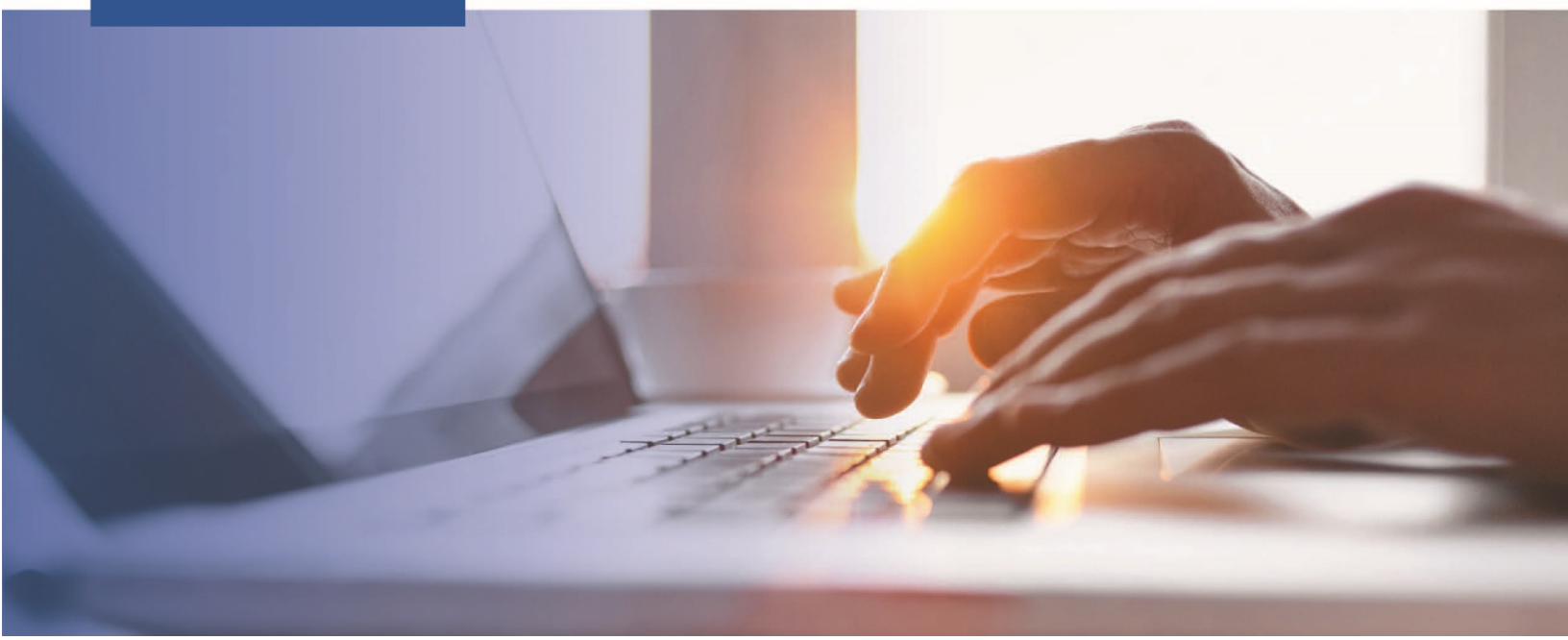
Cincom



Cincom® MANTIS®

WebSphere® MQ Programming

P39-1365-13

Version 3.5.01 and 6.5.01

Cincom® MANTIS®
WebSphere® MQ Programming
Publication Number P39-1365-13

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 45246-3732
USA

PHONE: +1 513 612 2300
FAX: +1 513 612 2000
WORLD WIDE WEB: http://www.cincom.com

Attention:
Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

External Web site disclaimer:
Cincom Systems, Inc. does not own, nor does it warrant the accuracy, adequacy or completeness of, the information and materials contained in linked documentation, and expressly disclaims liability for any errors or omissions in the information and materials. No warranty of any kind, implied, express or statutory, including but not limited to the warranties of non-infringement of third party rights, title, merchantability, fitness for a particular purpose and freedom from computer virus, is given in conjunction with the information and materials contained in this linked documentation. Nothing contained herein constitutes nor is intended to constitute an offer, inducement, promise, or contract of any kind. The data, information, and materials contained herein are for informational purposes only and are not represented to be error free. Any links are provided as a courtesy. They are not intended to nor do they constitute an endorsement by Cincom Systems, Inc. of the linked materials.

Release information for this manual

Cincom® MANTIS® WebSphere® MQ Programming, P39-1365-13, is dated June 1, 2019. This document supports release 3.5.01 and 6.5.01 of MANTIS®.

Cincom Technical Support for MANTIS®

To contact support, please visit https://supportWeb.cincom.com.

Contents

1. Overview

This chapter provides a detailed description of MQSeries support, in order to help you develop a MANTIS application.

Description of WebSphere MQ Programming

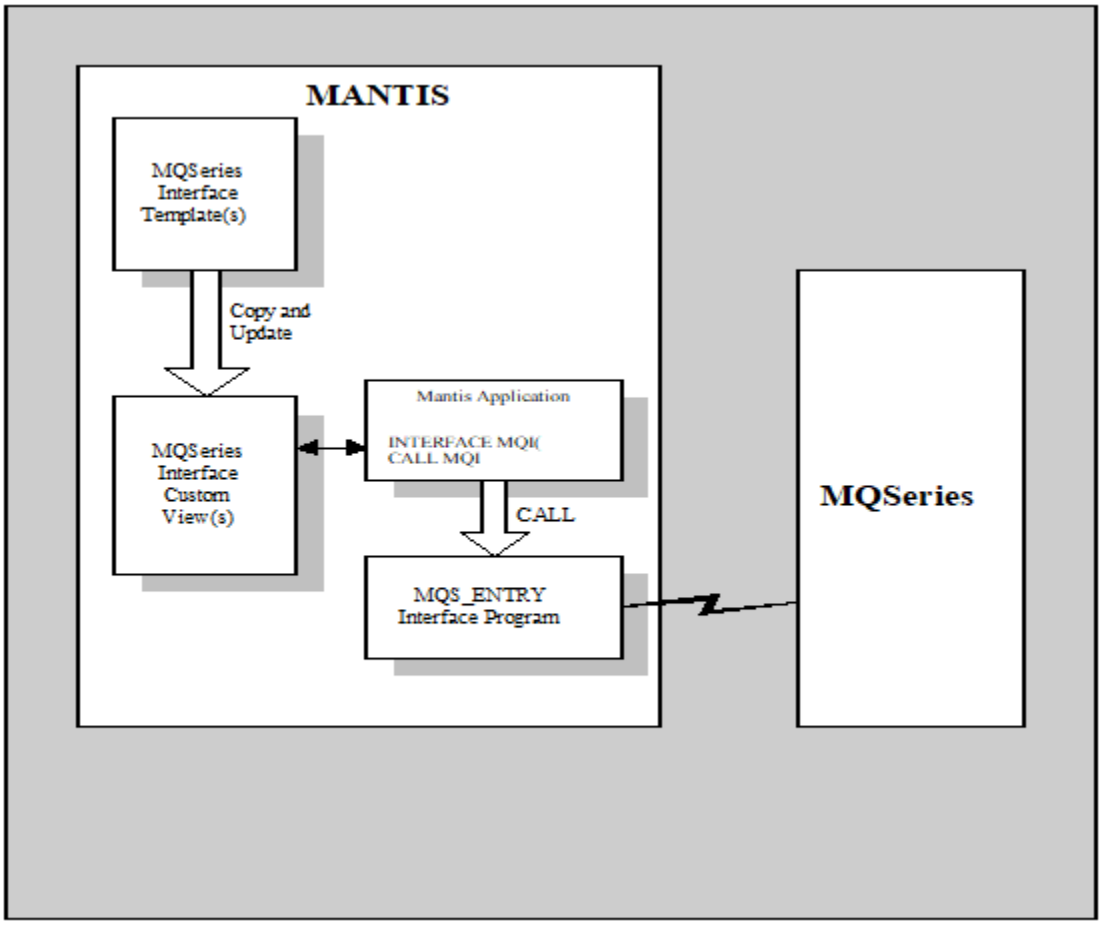
WebSphere MQ Programming is a MANTIS feature that adds support for IBM's MQSeries messaging product. This feature enables MANTIS application programmers to send MQSeries messages to, and receive MQSeries messages from, any local or remote machine that supports MQSeries.

Cincom has implemented WebSphere MQ Programming differently from other MANTIS facilities. It is implemented as the following:

- **A set of Interfaces.** These are stored under the MASTER user and serve as templates.
- **A generalized Interface program.** This program is invoked when a MANTIS program CALLs one of the Interfaces.

Development cycle figure

Below is a figure representing a typical development cycle that uses MANTIS and the MQSeries Interface:



Using an Interface layout as a template for your Interface

The Interface layouts that (along with the generalized Interface program) make up WebSphere MQ Programming are templates for your Interfaces. To use an Interface layout as a template, perform the following:

1. Use the Interface Design Facility Library Functions to fetch the appropriate Interface template.
2. As appropriate for your application, customize the copy of the Interface layout
   For example, a possible change you can make is to add fields for user data at the end of the Interface layout.
3. Use Library Functions to save the Interface template under a new name.

Generalized Interface program

The major component of MQSeries support is a generalized Interface program. For all environments, the name of the "Program to be Called" is MANTISMQ. This name is then processed as follows, according to operating system:

- **Mainframe.** Consider the following:
  - In CICS—The name MANTISMQ is translated internally to CSOXWMQS.
  - In BATCH—The name MANTISMQ is translated to a program called CSOXBMQS.
- **UNIX.** MANTISMQ is the entry point name, which is located in a shared library called libmanmqs.so.

Under both mainframe CICS and UNIX, this program performs the following:

1. Processes the requests from the MANTIS application
2. Makes the corresponding MQI calls to MQSeries

MQI and WebSphere MQ Programming

Cincom has modeled WebSphere MQ Programming after the standard MQSeries API called "MQI." Application programmers who are familiar with MQI in either the C or COBOL programming languages will find this feature to be simple and easy to use.

Reference materials

Programmers who are not familiar with the MANTIS INTERFACE and MANTIS CALL statements should refer to the following:

- For information on Interface design:
  - If you use OpenVMS or UNIX—*MANTIS Facilities*, P39-1301.
  - If you use z/OS or z/VSE—*MANTIS Facilities for z/OS and z/VSE*, P39-5301.
- For information on the CALL and INTERFACE statements:
  - If you use OpenVMS or UNIX—*MANTIS Language*, P39-1311.
  - If you use z/OS or z/VSE—*MANTIS Language for z/OS and z/VSE*, P39-5302.

Internal Interfaces for MQSeries support

MANTIS provides the following internal Interfaces for MQSeries support.

These internal Interfaces are located on the MASTER user library.

Interface	Description	Comments
MQBEGIN	Starts a unit of work.	♦ Optional. ♦ Unix only.
MQCOMMIT	Establishes a sync point and ends a unit of work.	♦ Optional. ♦ Unix and Mainframe Batch only.
MQCONNECT	Opens an MQSeries object and connects to it.	♦ This interface requires special initialization, performed with the “INITCONN” function. ♦ Creates a handle that all other Interfaces use.
MQDISCONNECT	Closes an MQSeries object and disconnects from it.	
MQEXIT	Closes all open handles.	♦ Optional. ♦ This Interface does not correspond to an MQSeries function; this Interface is an extra Cincom feature.
MQGET	Reads an MQSeries message.	♦ The user should perform the following: 1. Modify the Interface. 2. Save the Interface under a different name. ♦ This interface requires special initialization, performed with the “INITGET” function.
MQPUT	Sends an MQSeries message.	♦ The user should perform the following: 1. Modify the Interface. 2. Save the Interface under a different name. ♦ This interface requires special initialization, performed with the “INITPUT” function.

Interface	Description	Comments
MQROLLBACK	Reverse GETs and PUTs back to a prior sync point or a unit of work begin.	♦ Optional. ♦ Unix and Mainframe Batch only.
MQTM	Retrieves the z/OS MQTM messages or the UNIX MQTMC2 Trigger message.	♦ Optional. ♦ Unix and Mainframe CICS only.

2. Fundamental usage

This chapter provides a detailed discussion of common fields in MQSeries Interface views. All MQSeries Interface views start with the same common fields. Although the Interface type may prefix each of these fields, the order and meaning of the fields remains the same across all MQSeries Interface layouts.

Common fields in MQSeries Interface views

See the following syntax definition:

CALL mqinterface(function, handle, compcode, reason, ident, dmplength, dmpfilename ...)

For explanations of the parameters in this syntax definition, see the rest of this chapter.

function

Description *Required.* The desired operation request.

Format 3–8 character text expression.

Options The allowable function strings are:

- "GET"
- "MQGET"
- "PUT"
- "MQPUT"
- "BEGIN"*†
- "MQBEGIN"†
- "COMMIT"*†b
- "MQCMIT"*†b
- "CONNECT"
- "MQCONN"
- "DISCONN"
- "MQDISC"
- "ROLLBACK"*†b
- "MQBACK"*†b
- "DUMP"
- "EXIT"*†
- "INITPUT"
- "INITGET"
- "INITCONN"
- "INITMQTM"*†

* Unix
† Mainframe CICS
b Mainframe Batch

Consideration An invalid function is a function that does not equal one of the strings listed above (for example, NULL is an invalid function). If the function is invalid, MANTIS returns an error in the REASON and COMPCODE fields.

handle

Description *Required.* The MQSeries handle that MQSeries sets and uses for subsequent MQI calls.

Format A MANTIS symbolic name, defined as a BIG.

Consideration Retain this value for subsequent calls.

compcode

Description *Required.* The MQSeries compcode, where MQSeries sets a high-level completion code for the prior CALL.

Format A MANTIS symbolic name, defined as a BIG.

Consideration The MQSeries Interface sets this value to:

- **1.** Successful.
- **2.** Warning.
- **3.** Error.

General Considerations

- All Interfaces call the same entry point, which is named MQS_ENTRY. The FUNCTION field in each Interface points the MANTIS/MQSeries executable to the appropriate handler.
- Prior to making any Interface calls, set all information pertaining to the call in the Interface layout that includes the following:
  - HANDLE (usually)
  - FUNCTION
- After the call completes, check the Interface symbolic name for an indication of a warning or error. If the Interface symbolic name returns a non-NULL value, interrogate the REASON field.
- You may connect to multiple MQSeries objects. When you do this, the called Interface program generates multiple HANDLES (one HANDLE for each connection). You can disconnect from these handles by explicitly calling the MQDISCONNECT Interface that has the corresponding HANDLE and settings. Another disconnection method is to call the MQEXIT routine, which disconnects the application from all connected objects.
- For UNIX, to preserve data integrity, when MANTIS terminates (either normally or abnormally), MANTIS automatically calls the MQEXIT Interface as though the Interface was called by the user's application.

---

reason

Description *Required.* The MQSeries reason code that MQSeries sets. If a non-zero COMPCODE is returned, the *reason* field further defines the problem.

This field will contain the exact MQSeries reason code when an error occurs. For errors detected internally by the MQSeries Interface, a negative reason code will be returned. These codes will be further defined later in this documentation.

Format A MANTIS symbolic name, defined as a BIG.

Consideration For a description of each reason code, refer to *MQSERIES Application Programming Reference*, SC33-1673.

---

ident

Description *Internal.* The Interface identification field.

Format A MANTIS symbolic name, defined as a BIG.

Consideration This field is used internally to identify the Interface type and requires no user modification.

---

dmplength

Description *Optional.* The length of the Interface layout to dump.

Format A MANTIS symbolic name, defined as a BIG.

Consideration For more information, see 9. MQSeries/MANTIS diagnostic considerations.

dmpfilename

Description *Optional.* Name of the dump file being generated.

Format A MANTIS symbolic name, defined as a TEXT for a length of 80 bytes.

Consideration For more information, see 9. MQSeries/MANTIS diagnostic considerations.

3. Field naming conventions

Field prefixes

For most Interface layouts, many fields within the Interface layout have the same prefix. For example, in the MQPUT Interface layout, many fields are prefixed with “MQMD_”. This prefix precedes the name of each field in the MQMD structure layout for an MQPUT MQI call to MQSeries.

You will find many similarly named fields in MQSeries Interface layouts. To prevent the automatic mapping feature of MANTIS from reusing these fields, all fields in all Interface layouts are prefixed to indicate in which Interface they are contained.

For example, consider the MQPUT Interface layout. Since FUNCTION, HANDLE, REASON, COMPCODE, IDENT, DMPLENGTH, and DMPFILENAME can be found in all views, these fields are all prefixed with PUT:

- PUT_FUNCTION
- PUT_HANDLE
- PUT_REASON
- PUT_COMPCODE
- PUT_IDENT
- PUT_DMPLENGTH
- PUT_DMPFILENAME

Adding another level of prefixing

You can add another level of prefixing in order to keep the functions separate. To do so, use PREFIX on the INTERFACE statement defining the Interface layout:

```
10 INTERFACE QUEUE1("MQPUT1",PASSWORD,PREFIX)
20 INTERFACE QUEUE2("MQPUT2",PASSWORD,PREFIX)
30 QUEUE1_PUT_FUNCTION="CONNECT"
```

Different kinds of fields, requiring different actions

Although many fields exist in the MQGET and MQPUT Interface views, you need not set all of these fields prior to the Interface call. This is because fields that are sent to MQSeries from the MANTIS application are classified as one of the following:

- **Inbound field.** The MANTIS application programmer must fill in these fields.
- **Outbound field.** These fields return data to the MANTIS application. The application may need to interrogate outbound fields.
- **Inbound/Outbound field.** These fields offer 2-way communication. In some cases, the MANTIS application programmer must fill in these fields. In other cases, the application may need to interrogate these fields.

The function that is being performed dictates which fields in the Interface layout must be set or interrogated. To learn more about which fields must be set for a given function, refer to *MQSERIES Application Programming Reference*, SC33-1673.

4. Errors

After returning from any of the Interface calls, check the INTERFACE symbolic variable for error conditions or warning conditions.

General error categories

There are only 3 possible values that can be returned by any of the MQSeries Interfaces, as described in the following table:

Symbolic name value	Interface name	Situation	Recommended Response
" "	All	Function successfully completed.	None required.
WARNING	All	A warning occurred during the MQSeries API call of the Interface routine.	Compare the value of the REASON field to the REASON field value listed in *MQSERIES Application Programming Reference*, SC33-1673.
ERROR	All	An error occurred during the processing of the user's request. It could have been an internal error or one generated by an MQSeries API call.	Review the value of the REASON field in the view. If the value is positive, compare the value to the *MQSERIES Application Programming Reference*, SC33-1673. If the value is negative, compare the value to one of the possible values listed below.

Negative REASON codes

These codes can be generated from any of the MQSeries Interface calls. They are restricted to the internal workings of the MQSeries Interface and exclude any of the possible error conditions returned by the MQSeries APIs.

The following table lists each possible negative REASON value. The vast majority of these values should never be encountered but are listed here for completeness. For each possible error condition, the table lists the following:

- Name of Interface in which this error condition or warning condition occurs
- Situation in which this error condition or warning condition occurs
- Recommended response to this error condition or warning condition

For information on positive REASON codes (those generated by the MQSeries API call), refer to *MQSERIES Application Programming Reference*, SC33-1673.

REASON value	Interface name	Situation	Recommended response
-1	All	An error occurred during initial memory allocation.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-2	All	The Interface layout supplied an invalid value in the FUNCTION field.	Correct the FUNCTION field so that it contains one of the following values: ♦ BEGIN or MQBEGIN ♦ COMMIT or MQCMIT ♦ CONNECT or MQCONN ♦ DISCONN or MQDISC ♦ DUMP ♦ EXIT ♦ GET or MQGET ♦ INITCONN ♦ INITGET ♦ INITMQTM ♦ INITPUT ♦ PUT or MQPUT ♦ ROLLBACK or MQBACK

REASON value	Interface name	Situation	Recommended response
-3	All	The Interface layout supplied an invalid value of ZERO in the HANDLE field.	Perform the following: Correct the value in the HANDLE field so that it matches one of the values returned by the MQCONNECT call. Check the spelling of the HANDLE variable. Check to see if you have assigned a valid handle value to the HANDLE variable. If not, assign a valid handle.

REASON value	Interface name	Situation	Recommended response
-4	All	The Interface layout supplied an invalid NONZERO value in the HANDLE field. This may be the result of one of the following: ♦ A handle was closed and disconnected. ♦ The program modified the variable holding the handle. ♦ *For Mainframe CICS Pseudo-Conversational mode:* A screen converse occurred in between an MQSeries connection and the current use of this handle, causing the loss of the connection.	Correct the value in the HANDLE field so that it matches one of the values returned by the MQCONNECT call.
-5	All	The DUMP function failed because you did not set the interface type to a valid type.	***Note:*** *For the recommended response, see 9. MQSeries/MANTIS diagnostic considerations.*
-6	All	The DUMP function failed because of one of the following: ♦ The Interface type was not identified. ♦ The DMPLENGTH field was set to ZERO.	***Note:*** *For the recommended response, see 9. MQSeries/MANTIS diagnostic considerations.*

REASON value	Interface name	Situation	Recommended response
-7	All	The DUMP function failed because a write error occurred. This may have resulted from one of the following: ♦ You have insufficient I/O or access privileges to perform the write operation on the dump file. ♦ Exhaustion of space on the device where the dump file is supposed to be stored.	***Note:*** *For the recommended response, see 9. MQSeries/MANTIS diagnostic considerations.*
-8	All	The DUMP function failed because an open failure occurred on the dump file name in the view. This may have resulted from one of the following: ♦ The DMFILENAME field contained a blank or invalid file name. ♦ The dump file name had insufficient privileges.	***Note:*** *For the recommended response and the correct DMFILENAME syntax, see 9. MQSeries/MANTIS diagnostic considerations.*
-10	All	An error occurred during initial internal processing of the user handle.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.

REASON value	Interface name	Situation	Recommended response
-14	All	A memory free error occurred after an MQSeries object was closed.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-15	All	A "memory free" error occurred when an attempt was made to free an internal MQSeries object during a call to either the MQDISCONNECT or MQEXIT interface.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-16	All	A "memory free" error occurred when an attempt was made to free an internal MQSeries object during a call to either the MQDISCONNECT or MQEXIT interface.	***Note:*** *This error should never occur but has been documented for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-19 ***Note:*** *Mainframe only.*	MQBEGIN	An MQSeries begin was attempted in an environment that does not support it.	***Note:*** *The MQSeries begin is not available for the mainframe environments. Refer to* MQSeries Application Programming Reference, *SC33-1673.*
-22 ***Note:*** *Mainframe CICS only.*	MQCOMMIT	An MQSeries commit was attempted in an environment that does not support it.	***Note:*** *The MQSeries commit is not available for the mainframe CICS environments. Refer to* MQSeries Application Programming Reference, *SC33-1673.*

REASON value	Interface name	Situation	Recommended response
-23	All	A memory free error occurred while the MQSeries Interface was terminating.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-26	MQCONNECT	A memory allocation error occurred after an MQseries object was opened.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-27	MQCONNECT	A memory allocation error occurred after an MQSeries object was connected.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-29 ***Note:*** *Mainframe CICS only.*	MQROLLBACK	An MQSeries ROLLBACK was attempted in an environment that does not support it.	***Note:*** *The MQSeries rollback is not available for the mainframe CICS environments. Refer to* MQSeries Application Programming Reference*, SC33-1673.*
-34 ***Note:*** *Mainframe only.*	ALL	The MANTIS program attempted to access the MQSeries Interface, without having been authorized to do so.	***Note:*** *The installation options for this version of MANTIS do not allow access to the MQSeries Interface.* If you would like more information, contact your Master User.

REASON value	Interface name	Situation	Recommended response
-35 ***Note:*** *Mainframe CICS only.*	ALL	A memory allocation error occurred while the MQSeries Interface environment was being initialized.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-36 ***Note:*** *Mainframe CICS only.*	MQTM	The call to the MQTM Interface was made, but the TM_MEMADDR field, located in the Interface layout, was not set to a valid memory address that contained the MQSeries MQTM record.	***Note:*** *For the recommended response, see 7. MQSeries/MANTIS triggering.*
-37 ***Note:*** *Mainframe CICS only.*	MQTM	An attempt was made to free the memory that the TM_MEMADDR field, located in the MQTM Interface layout, specified. This may have been the result of one of the following: ♦ In the TM_MEMADDR field, which is located in the MQTM view, there was an invalid address. ♦ A CICS program caused memory corruption.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-38 ***Note:*** *Mainframe Batch only.*	MQTM	MQSeries triggering is not supported in the Mainframe batch environments.	***Note:*** *For the recommended response, see 7. MQSeries/MANTIS triggering.*

REASON value	Interface name	Situation	Recommended response
-39 ***Note:*** *UNIX only.*	MQTM	A call to the MQTM Interface was made without the MQTMC2 environment variable having been set.	Correct the process that is invoked to handle the message that invoked the trigger. This process must pass the MQTMC2 record to MANTIS by setting the record in the MQTMC2 environment variable before MANTIS is executed. ***Note:*** *For more information, see 7. MQSeries/MANTIS triggering.*
-40	MQCONNECT	Memory chains within the MANTIS process were corrupt during the MQCONNECT routine call that attempted to acquire memory for internal purposes.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-41	MQCONNECT	Memory chains within the MANTIS process were corrupt during the MQCONNECT routine call that attempted to acquire memory for internal purposes.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.
-42	MQDISCONNECT MQEXIT	A memory free error occurred while the MQSeries Interface was being terminated.	***Note:*** *This error should never occur but is documented here for the sake of completeness.* Perform the following: Document the problem. Contact Cincom's technical support.

REASON value	Interface name	Situation	Recommended response
-43 ***Note:*** *Mainframe CICS only.*	MQCONNECT	Memory for tracking OPEN handles could not be obtained. This may have been the result of one of the following: ♦ Many calls were made to the MQCONNECT interface, without a call being made to the MQDISCONNECT interface. ♦ A program in Program Design was re-run too many times without either a call to MQDISCONNECT in Conversational mode or a COMMIT occurring.	Because mainframe CICS MANTIS can only concurrently connect to the same MQSeries Queue Manager 50 times, you must limit the number of calls to the MQCONNECT interface to 50 or less. ***Note:*** *For more information on MQCONNECT for CICS Mainframe, see* 7. MQSeries/MANTIS triggering*.*

5. Constants

MQSeries's many capabilities and options result in the availability of numerous constants for use in MQSeries's various functions. Cincom has duplicated these constants into a program called MQ_INIT, which is located under the MASTER user.

Including MQ_INIT in a user program

Include MQ_INIT in any user program by performing one of the following:

- Copying all or part of MQ_INIT into your program during editing
- Including MQ_INIT in your program as a COMPONENT

Categories of MQSeries constants in MQ_INIT

MQ_INIT contains the categories of MQSeries constants that are listed in the following table.

Category of MQSeries constants	Description
Completion Codes	Constants that can be used with any Interface to test the value of the COMPCODE field for normal, warning, and error statuses.
MQCLOSE Options	Constants defining options in the MQDISCONNECT Interface that can be used during the CLOSE of an MQSeries object.
MQGMO Constants for GET	Constants that can be used in the MQGET Interface and consist of the following: ♦ MQGMO Get Message Options ♦ Match Options ♦ Group Status ♦ Segment Status ♦ Segmentation and Expiry
MQMD Constants for PUT	Constants that can be used in the MQPUT Interface and consist of the following: ♦ Structure ID ♦ Version number ♦ Report Options ♦ Message Type ♦ Feedback Values ♦ Encoding Values ♦ Coded Character ♦ Set Identifiers ♦ Format Values ♦ Priority Values ♦ Persistence Values ♦ Message Identifier Values ♦ Message Correlation Identifier ♦ Put Application Types ♦ Put Message Flags
MQOPEN Object Type Definitions	Constants that define the object type being opened and used in the MQCONNECT Interface.
MQPMO Constants for PUT	Constants that can be used in the MQPUT Interface and consist of the following: ♦ MQPMO Version Number ♦ MQPMO Structure Length ♦ MQPMO Put-Message Options ♦ MQPMO Message Record Fields
Reason Codes	Constants that can be used with any Interface to test the value of the REASON field for any of the errors generated by MQSeries.

6. Building a MANTIS MQSeries application

This chapter describes how to perform the following:

- **Create programs that use CALLs to the MQSeries Interfaces**
- **Initialize the MQSeries Interfaces**
- **Use the MQSeries Interfaces**

Creating a program that reads a message queue

To create a program that reads a message queue, perform the following:

1. Define your message layouts to MANTIS. To accomplish this, perform the following:
   a. On the MANTIS Facility Selection Menu, select the Design an Interface option.
   b. Load the MQGET Interface design template.
   c. Add your message data fields to the end of the Interface layout.
   d. Save the new Interface description under a different name.
   e. If your program will be reading more than one message layout, repeat steps B–D.
2. Write a program that contains the following structure for your application logic and your Interface CALLs:

Pseudo-code	Comment
MQCONNECT	
MQINITGET	
MQBEGIN	Your program may or may not require this Interface CALL.
Loop	
MQGET	
MQCOMMIT or MQROLLBACK	Your program may or may not require this Interface CALL.
Endloop	
MQDISCONNECT or MQEXIT	

Creating a program that writes to a message queue

To create a program that writes to a message queue, perform the following:

1. Define your message layouts to MANTIS. To accomplish this, perform the following:
   a. On the MANTIS Facility Selection Menu, select the Design an Interface option.
   b. Load the MQPUT Interface design template.
   c. Add your message data fields to the end of the Interface layout.
   d. Save the new Interface description under a different name.
   e. If your program will be reading more than one message layout, repeat steps B–D.
2. Write a program that contains the following structure for your application logic and your Interface CALLs:

Pseudo-code	Comment
MQCONNECT	
MQINITPUT	
MQBEGIN	Your program may or may not require this Interface CALL.
Loop	
MQPUT	
MQCOMMIT or MQROLLBACK	Your program may or may not require this Interface CALL.
Endloop	
MQDISCONNECT or MQEXIT	

Initializing Interfaces that do not require special initialization

Use the CLEAR statement to initialize Interfaces that do not require special initialization. These Interfaces are:

- MQBEGIN
- MQCOMMIT
- MQDISCONNECT
- MQEXIT
- MQROLLBACK
- MQTM

How to use the CLEAR statement to initialize an Interface

To use the CLEAR statement, place it on a program line, followed by an Interface name. Below is sample code that includes the CLEAR statement and an Interface name:

```
10 INTERFACE MQCOMMIT("MASTER:MQCOMMIT",PASSWORD)
...
1000 CLEAR MQCOMMIT
1010 COM_FUNCTION="COMMIT"
1020 COM_HANDLE=SAVE_HANDLE1
1030 CALL MQCOMMIT
```

What the CLEAR statement does

When the program executes the CLEAR statement, the CLEAR statement performs the following:

1. Clears all fields in the Interface.
2. Sets all TEXT fields in the Interface to NULL.
3. Sets all numeric fields in the Interface to ZERO.

When you execute the CLEAR statement, fields defined in the Interface have initial values of NULL or ZERO. This follows normal MANTIS rules. The exception occurs when you auto-map fields to variables to which you have already assigned values.

Initializing Interfaces that require special initialization

Use CALLs to special functions in order to initialize Interfaces that require special initialization. These Interfaces are:

- **MQCONNECT**. Call the INITCONN function to initialize this Interface layout. See Initializing the MQCONNECT Interface.
- **MQGET**. Call the INITGET function to initialize this Interface layout. See Initializing the MQGET Interface to its default usable state.
- **MQPUT**. Call the INITPUT function to initialize this Interface layout. See Initializing the MQPUT Interface to its default usable state.

For information on the default state for each of these Interface layouts, refer to *MQSERIES Application Programming Reference*, SC33-1673.

Sample code for initializing an Interface that requires special initialization

Below is sample code for specifying the INITPUT function (used for initializing the MQPUT Interface layout):

```
10 INTERFACE QUEUE1("MQPUT1",PASSWORD)
20 CALL QUEUE1("INITPUT")
30 IF PUT_COMPCODE<>ZERO
40 .DO ERROR_ROUTINE
50 END
```

Using the MQSeries Interface layouts

The MQSeries Interface layouts, which are described in this section, are designed for use with the MQSeries Interface program. They are located in the MASTER library

These Interface layouts are listed in the following table:

For information on using these Interface layouts in programs, see Creating a program that reads a message queue and Creating a program that writes to a message queue.

Interface layout name	Description	Used as-is?	Requires special initialization?	Comments
MQBEGIN	Starts a unit of work.	Yes	No	N/A
MQCOMMIT	Commits messages that were sent and received during a unit of work.	Yes	No	N/A
MQCONNECT	Opens and connects to an MQSeries object.	Yes	Yes	Returns handle for use on other Interfaces. **Note:** Requires special initialization prior to its use (“INITCONN”).
MQDISCONNECT	Closes and disconnects from an MQSeries object.	Yes	No	N/A
MQEXIT	Closes all open handles.	Yes	No	N/A
MQGET	Reads an MQSeries message.	No	Yes	To modify this Interface, perform the following: 1. Copy this Interface. 2. Add your message data layout to the end of the copy. **Note:** Requires special initialization prior to its use (“INITGET”).
MQPUT	Sends an MQSeries message.	No	Yes	To modify this Interface, perform the following: 1. Copy this Interface. 2. Add your message data layout to the end of the copy. **Note:** Requires special initialization prior to its use (“INITPUT”).
MQROLLBACK	Resets rollback to prior synch point.	Yes	No	N/A

Interface layout name	Description	Used as-is?	Requires special initialization?	Comments
MQTM	Maps the MQSeries trigger data.	Yes	No	N/A

Using the MQBEGIN Interface to start a unit of work

The MQBEGIN Interface enables the user program to start a unit of work, as defined by MQSeries and its mqbegin API.

Changing the MQBEGIN Interface layout

Altering data already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQBEGIN Interface layout figure

The Interface layout is shown below:

```
Page No : 1  :                  Interface Area Layout          2001/09/26
MASTER:MQBEGIN                                                 07:53:57
Element Count : 7                                 Element Size : 108
ELEM  -------------NAME------------- TYPE  FORMAT  LEN SIGN  DEC DIM -ATTRIBUTE-
  1   BEG_FUNCTION                  TEXT  TEXT    8
  2   BEG_HANDLE                    BIG   BINARY  4
  3   BEG_COMPCODE                  BIG   BINARY  4
  4   BEG_REASON                    BIG   BINARY  4   YES
  5   BEG_IDENT                     BIG   BINARY  4
  6   BEG_DMPLENGTH                 BIG   BINARY  4
  7   BEG_DMPFILENAME               TEXT  TEXT    80
```

Initializing the MQBEGIN Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, perform the following:

1. Set BEG_HANDLE to a valid handle returned by the MQCONNECT Interface.
2. Set BEG_FUNCTION to the string value “BEGIN” or “MQBEGIN”.

Sample code. See the following sample code for examples of setting BEG_HANDLE and BEG_FUNCTION:

```
230 INTERFACE MQBEGIN("MASTER:MQBEGIN",PASSWORD)
240 BEG_FUNCTION="BEGIN"
250 BEG_HANDLE=SAVE_HANDLE1
260 CALL MQBEGIN
```

Description of sample code. The above MANTIS program performs the following:

3. Loads the MQBEGIN Interface.
4. Sets the required fields (BEG_FUNCTION and BEG_HANDLE).

5. Calls the Interface to begin a unit of work based on the object pointed to by the HANDLE that was returned by a previous MQCONNECT (CON_HANDLE or another variable assigned its value).

MQBEGIN Interface and Transaction Server

MQBEGIN is not supported in Transaction Server. In Transaction Server, transaction support of MQSeries messages falls under normal MANTIS transaction guidelines.

Using the MQCOMMIT Interface to establish a sync point and commit all previous message GETs and PUTs

The MQCOMMIT Interface enables you to establish a sync point and to commit all previous message GETs and PUTs in the manner that MQSeries and its mqcmit API define.

Changing the MQCOMMIT Interface layout

Altering data already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQCOMMIT Interface area layout figure

The Interface area layout is shown below:

```
Page No : 1   :                  Interface Area Layout          2001/09/26
MASTER:MQCOMMIT                                                 07:54:44
Element Count : 7                                  Element Size : 108
ELEM  -------------NAME------------- TYPE  FORMAT  LEN SIGN  DEC DIM -ATTRIBUTE-
  1   COM_FUNCTION                  TEXT  TEXT    8
  2   COM_HANDLE                    BIG   BINARY  4
  3   COM_COMPCODE                  BIG   BINARY  4
  4   COM_REASON                    BIG   BINARY  4   YES
  5   COM_IDENT                     BIG   BINARY  4
  6   COM_DMPLENGTH                 BIG   BINARY  4
  7   COM_DMPFILENAME               TEXT  TEXT    80
```

Initializing the MQCOMMIT Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, perform the following:

1. Set COM_FUNCTION to the string value “COMMIT” or “MQCMIT”.
2. Set COM_HANDLE to a valid handle that was returned by the MQCONNECT Interface.

Sample code. See the following sample code:

```
230 INTERFACE MQCOMMIT("MASTER:MQCOMMIT",PASSWORD)
240 COM_FUNCTION="COMMIT"
250 COM_HANDLE=SAVE_HANDLE1
260 CALL MQCOMMIT
```

Description of sample code. The above MANTIS program performs the following:

3. Loads the MQCOMMIT Interface.
4. Sets the required fields (COM_FUNCTION and COM_HANDLE).

5. Calls the Interface, in order to establish a sync point based on the object that is pointed to by the HANDLE that was returned by a previous MQCONNECT (CON_HANDLE).

MQCOMMIT and Transaction Server

MQCOMMIT is not supported in Transaction Server. In Transaction Server, transaction support of MQSeries messages falls under normal MANTIS transaction guidelines.

Using the MQCONNECT Interface to open and connect to an MQSeries object

The MQCONNECT Interface enables the user program to open and connect to an MQSeries object via the mqconn and mqopen APIs. Not all calls to the MQCONNECT interface result in an mqconn call; one qconn call is made for each unique object. Nevertheless, every call to the MQCONNECTY interface will yield a call to the mqopen.

Changing the MQCONNECT Interface layout

Altering data already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQCONNECT Interface layout figure

The Interface layout is shown below:

```
Page No : 1   :              Interface Area Layout              2000/11/26
MASTER:MQCONNECT                                                19:05:14
Element Count : 13                                 Element Size : 272
ELEM  -------------NAME------------- TYPE   FORMAT  LEN SIGN  DEC DIM -ATTRIBUTE-
  1   CON_FUNCTION                   TEXT   TEXT    8
  2   CON_HANDLE                     BIG    BINARY  4
  3   CON_COMPCODE                   BIG    BINARY  4
  4   CON_REASON                     BIG    BINARY  4   YES
  5   CON_IDENT                      BIG    BINARY  4
  6   CON_DMPLENGTH                  BIG    BINARY  4
  7   CON_DMPFILENAME                TEXT   TEXT    80
  8   CON_QMGRNAME                   TEXT   TEXT    48
  9   CON_OPTIONS                    BIG    BINARY  4
 10   MQOD_OBJECTTYPE                BIG    BINARY  4
 11   MOOD_OBJECTNAME                TEXT   TEXT    48
 12   MQOD_OBJECTQMGRNAME            TEXT   TEXT    48
 13   MQOD_ALTERNATEUSERID           TEXT   TEXT    12
```

For information on elements 7–12 in the preceding figure, refer to *MQSeries Application Programming Reference*, SC33-1673. These elements are the following:

- QMGRNAME
- OPTIONS
- OBJECTTYPE
- OBJECTNAME
- OBJECTQMGRN_
- ALTERNATEUSE

Initializing the MQCONNECT Interface

Special initialization. This Interface requires special initialization—that is, before you can use this Interface, you must call a special function in order to initialize the Interface to its default usable state.

Initialization steps. Accomplish the special initialization by performing the following:

1. Place the "INITCONN" string in the FUNCTION field.
2. Call the MQCONNECT Interface.

To review the settings for the default state of mqconn, refer to *MQSERIES Application Programming Reference*, SC33-1673. Once the INITCONN function initializes the mqconn to its default values, the application can change them prior to calling the MQCONNECT Interface to connect and open the MQSeries object.

Sample code. See the following sample code for examples of placing the INITCONN string in the FUNCTION field and calling the MQCONNECT Interface:

```
10 INTERFACE MQCONNECT("MASTER:MQCONNECT",PASSWORD)
15 CON_FUNCTION="INITCONN"
20 CALL MQCONNECT
```

Description of sample code. The above MANTIS program performs the following:

1. Loads the MQCONNECT Interface.
2. Sets the required field (CON_FUNCTION).
3. Calls the MQCONNECT Interface.

Connecting to the MQSeries object

Connection procedure. To open the MQSeries object and connect to it, perform the following:

1. Change the CON_FUNCTION to "CONNECT" or "MQCONN".
2. Call the INTERFACE.

Sample code. See the following sample code:

```
25 CON_FUNCTION="CONNECT"
30 CALL MQCONNECT
```

Description of sample code. The above MANTIS program performs the following:

1. Sets the required field (CON_FUNCTION).
2. Calls the MQCONNECT Interface.

   Once the MQCONNECT call returns successfully, CON_HANDLE contains a value to use for all input and output to that object.

Using the same MQCONNECT Interface to open multiple objects

Connection procedure. To use the same MQCONNECT Interface to open multiple objects, perform the following:

1. Save the CON_HANDLE into another SMALL or BIG variable (for example, SAVE_HANDLE1).
2. Reinitialize the Interface with a CALL using function INITCONN.
3. Call the MQCONNECT Interface.

Sample code. See the following sample code:

```
10 INTERFACE MQCONNECT("MASTER:MQCONNECT",PASSWORD)
20 SMALL SAVE_HANDLE1
```

```
30 SMALL SAVE_HANDLE2
40 CON_FUNCTION="INITCONN"
50 CALL MQCONNECT
60 CON_FUNCTION="CONNECT"
70 CALL MQCONNECT
80 SAVE_HANDLE1=CON_HANDLE
90 CON_FUNCTION="INITCONN"
100 CALL MQCONNECT
110 CON_FUNCTION="CONNECT"
120 CALL MQCONNECT
130 SAVE_HANDLE2=CON_HANDLE
```

Comment on the preceding code. When dealing with queues, you may want to have multiple contexts in order to establish different currencies. For example, the same queue can be opened multiple times in a program:

- Opened once for browsing
- Opened once for destructive reading
- Opened once for writing

You can use the same MQCONNECT Interface and save separate handles to each connection.

Using the MQDISCONNECT Interface to close and disconnect from an MQSeries object

The MQDISCONNECT Interface enables the user program to close and disconnect from an MQSeries object via the mqclose and mqdisc APIs. Every call to the MQDISCONNECT interface will result in the call to mqclose. The last mqclose on a particular connected MQSeries object will result in a call to the mqdisc API.

Changing the MQDISCONNECT Interface layout

Altering data already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQDISCONNECT Interface area layout figure

The Interface area layout is shown below:

```
Page No : 1  :                  Interface Area Layout          2001/09/26
MASTER:MQDISCONNECT                                            07:48:33
Element Count : 8                                  Element Size : 112
ELEM  -------------NAME------------- TYPE  FORMAT  LEN SIGN  DEC DIM -ATTRIBUTE-
  1   DIS_FUNCTION                  TEXT  TEXT    8
  2   DIS_HANDLE                    BIG   BINARY  4
  3   DIS_COMPCODE                  BIG   BINARY  4
  4   DIS_REASON                    BIG   BINARY  4   YES
  5   DIS_IDENT                     BIG   BINARY  4
  6   DIS_DMPLENGTH                 BIG   BINARY  4
  7   DIS_DMPFILENAME               TEXT  TEXT    80
  7   DIS_OPTIONS                   BIG   BINARY  4
```

Initializing the MQDISCONNECT Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, perform the following:

1. Set DIS_FUNCTION to the string value of “DISCONNECT” or “MQDISC”.
2. Set DIS_HANDLE to a valid handle returned by the MQCONNECT Interface.
3. Set DIS_OPTIONS to any needed disconnect options.

Sample code. See the following sample code:

```
100 INTERFACE MQDISCONNECT("MASTER:MQDISCONNECT",PASSWORD )
110 DIS_FUNCTION="DISCONNECT"
120 DIS_HANDLE=SAVE_HANDLE1
130 CALL MQDISCONNECT
```

Description of sample code. The above MANTIS program performs the following:

1. Loads the MQDISCONNECT Interface.
2. Sets the required fields.
3. Calls the Interface, in order to close and disconnect the MQSeries object pointed to by the HANDLE that was returned by a previous MQCONNECT.

Using the MQEXIT Interface to close all open handles

The MQEXIT Interface enables the user program to close all open handles with one call to MQEXIT.

Changing the MQEXIT Interface layout

Altering data already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQEXIT Interface layout figure

The Interface layout is shown below:

```
Page No : 1  :                  Interface Area Layout          2001/09/26
MASTER:MQEXIT                                                  07:55:47
Element Count : 7                                 Element Size : 108
ELEM  -------------NAME------------- TYPE  FORMAT  LEN SIGN  DEC DIM -ATTRIBUTE-
  1   EXT_FUNCTION                  TEXT  TEXT    8
  2   EXT_HANDLE                    TEXT  TEXT    32
  3   DIS_COMPCODE                  BIG   BINARY  4
  4   DIS_REASON                    BIG   BINARY  4   YES
  5   DIS_IDENT                     BIG   BINARY  4
  6   DIS_DMPLENGTH                 BIG   BINARY  4
  7   DIS_DMPFILENAME               TEXT  TEXT    80
```

Initializing the MQEXIT Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, set EXT_FUNCTION to the string value “EXIT”.

UNIX Only: During MANTIS termination, whether it's for NORMAL or FATAL processing, BATCH or INTERACTIVE, if there are any connected resources, the MANTIS nucleus calls MQEXIT.

Sample code. See the following sample code:

```
340 INTERFACE MQEXIT("MASTER:MQEXIT",PASSWORD)
350 EXT_FUNCTION="EXIT"
360 CALL MQEXIT
```

Description of sample code. The above MANTIS program performs the following:

1. Loads the MQEXIT Interface.
2. Sets the EXT_FUNCTION as required.
3. Calls the Interface, in order to close all open handles, created from prior calls, to the MQCONNECT Interface.

Using the MQGET Interface to read an MQSeries message

The MQGET Interface enables you to read an MQSeries message via the mqget API.

Changing the MQGET Interface layout

Changing fields that are already present in this Interface layout. Do not change (Alter, Insert, or Delete) the fields that already exist in this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user message data to the end of this Interface layout. You must add user message data to the end of this Interface layout. Before you can read an MQSeries message via the mqget API, perform the following:

1. Use the Interface Design Facility Library Functions to fetch the MQGET Interface template.
2. Select the Update Area Layout option.
3. Add your message data layout to the end of the MQGET Interface template.
4. Use Library Functions to save the MQGET Interface template under a new name.

Do not alter MQGET Interface template by replacing it.

Each different message layout requires its own MQGET Interface and is modeled on MASTER:MQGET.

MQGET Interface layout figure

The Interface layout is shown below:

```
Page No : 1 :              Interface Area Layout          2001/09/26
MASTER:MQGET                                              07:52:06
Element Count : 46                        Element Size : 556
ELEM -------------NAME------------- TYPE FORMAT LEN SIGN DEC DIM -ATTRIB

1     GET_FUNCTION                  TEXT TEXT   8
2     GET_HANDLE                    BIG  BINARY 4
3     GET_COMPCODE                  BIG  BINARY 4
4     GET_REASON                    BIG  BINARY 4   YES
5     GET_IDENT                     BIG  BINARY 4
6     GET_DMPLENGTH                 BIG  BINARY 4
7     GET_DMPFILENAME               TEXT TEXT   80
8     GET_MSGLENGTH                 BIG  BINARY 4
9     G_MQMD_VERSION                BIG  BINARY 4   YES
10    G_MQMD_REPORT                 BIG  BINARY 4   YES
11    G_MQMD_MSGTYPE                BIG  BINARY 4   YES
12    G_MQMD_EXPIRY                 BIG  BINARY 4   YES
13    G_MQMD_FEEDBACK               BIG  BINARY 4   YES
14    G_MQMD_ENCODING               BIG  BINARY 4   YES
15    G_MQMD_CODEDCHAR              BIG  BINARY 4   YES
16    G_MQMD_FORMAT                 TEXT TEXT   8
17    G_MQMD_PRIORITY               BIG  BINARY 4   YES
18    G_MQMD_PERSISTEN              BIG  BINARY 4
19    G_MQMD_MSGID                  TEXT TEXT   24
20    G_MQMD_CORRELID               TEXT TEXT   24
21    G_MQMD_BACKOUTCO              BIG  BINARY 4   YES
22    G_MQMD_REPLYTOQ               TEXT TEXT   48
23    G_MQMD_REPLYTOQM              TEXT TEXT   48
24    G_MQMD_USERIDENT              TEXT TEXT   12
25    G_MQMD_ACCOUNTIN              TEXT TEXT   32
26    G_MQMD_APPLIDENT              TEXT TEXT   32
27    G_MQMD_PUTAPPLTY              BIG  BINARY 4   YES
28    G_MQMD_PUTAPPLNA              TEXT TEXT   28
29    G_MQMD_PUTDATE                TEXT TEXT   8
30    G_MQMD_PUTTIME                TEXT TEXT   8
31    G_MQMD_APPLORIGI              TEXT TEXT   4
32    G_MQMD_GROUPID                TEXT TEXT   24
33    G_MQMD_MSGSEQNUM              BIG  BINARY 4   YES
34    G_MQMD_MSGFLAGS               BIG  BINARY 4   YES
35    G_MQMD_ORIGINALL              BIG  BINARY 4   YES
36    MQGMO_VERSION                 BIG  BINARY 4   YES
37    MQGMO_OPTIONS                 BIG  BINARY 4   YES
38    MQGMO_WAITINTERV              BIG  BINARY 4   YES
39    MQGMO_RESOLVEDQN              TEXT TEXT   48
40    MQGMO_MATCHOPTIO              BIG  BINARY 4   YES
41    MQGMO_GROUPSTATU              TEXT TEXT   1
42    MQGMO_SEGMENTSTA              TEXT TEXT   1
43    MQGMO_SEGMENTATI              TEXT TEXT   1
44    MQGMO_RESERVED1               TEXT TEXT   1
45    MQGMO_MSGTOKEN                TEXT TEXT   16
46    MQGMO_RETURNEDLE              BIG  BINARY 4   YES
Add your message fields here
```

Initializing the MQGET Interface to its default usable state

Special initialization. This Interface requires special initialization—that is, before you can use this Interface, you must call a special function in order to initialize the Interface to its default usable state.

The initialization process will not alter any of the user fields that you added to the end of the Interface layout.

To review the settings for the default state of an MQGET Interface, refer to *MQSERIES Application Programming Reference*, SC33-1673.

Initialization procedure. To initialize the Interface, perform the following:

1. Set the FUNCTION field to the “INITGET” string.
2. Call the Interface.

Sample code. See the following sample code:

```
140 INTERFACE MQGET1("MQGET1",PASSWORD)
150 GET_FUNCTION="INITGET"
160 CALL MQGET1
```

Description of sample code. The above MANTIS program performs the following:

1. Loads the MQGET1 Interface.
2. Sets the GET_FUNCTION variable.
3. Calls the MQGET1 Interface.

Changing the initialized Interface in order to receive a message

Once you have initialized the Interface, you can set the GET_FUNCTION and GET_HANDLE variables before you call the MQGET Interface, so that you are able to receive a message.

Procedure for receiving a message. To receive a message, perform the following:

1. Change the GET_FUNCTION variable to “GET” or “MQGET”.
2. Set the GET_HANDLE variable to a valid handle from a previous MQCONNECT.
3. Call the Interface.

Sample code. See the following sample code:

```
200 GET_FUNCTION="GET"
210 GET_HANDLE=SAVE_HANDLE2
220 CALL MQGET1
```

Description of sample code. The above MANTIS program performs the following:

1. Sets the GET_FUNCTION variable.
2. Sets the GET_HANDLE variable to a valid handle from a previous MQCONNECT.
3. Calls the MQGET1 Interface.

Using the MQPUT Interface to send an MQSeries message

The MQPUT Interface enables you to send an MQSeries message via the mqput API.

Changing the MQPUT Interface layout

Changing fields that are already present in this Interface layout. Do not change the sequence of fields in the beginning of this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user message data to the end of the Interface layout. You must add user message data to the end of this Interface layout. Before you can send an MQSeries message via the mqput API, perform the following:

1. Use the Interface Design Facility Library Functions to fetch the MQPUT Interface template.
2. Select the Update Area Layout option.
3. Add your message data layout to the end of the MQPUT Interface template.
4. Use Library Functions to save the MQPUT Interface template under a new name.

Do not alter the MQPUT Interface template by replacing it.

Each different message layout requires its own version of the MQPUT Interface and is modeled on MASTER:MQPUT.

MQPUT Interface layout figure

The Interface layout is shown below:

```
Page No : 1 :              Interface Area Layout          2001/09/26
 MASTER:MQPUT                                             07:50:13
 Element Count : 44                          Element Size : 592
 ELEM -------------NAME------------- TYPE FORMAT LEN SIGN DEC DIM -ATTRIB

 1     PUT_FUNCTION                  TEXT TEXT   8
 2     PUT_HANDLE                    BIG  BINARY 4
 3     PUT_COMPCODE                  BIG  BINARY 4
 4     PUT_REASON                    BIG  BINARY 4   YES
 5     PUT_IDENT                     BIG  BINARY 4
 6     PUT_DMPLENGTH                 BIG  BINARY 4
 7     PUT_DMPFILENAME               TEXT TEXT   80
 8     PUT_LENGTH                    BIG  BINARY 4
 9     P_MQMD_VERSION                BIG  BINARY 4   YES
 10    P_MQMD_REPORT                 BIG  BINARY 4   YES
 11    P_MQMD_MSGTYPE                BIG  BINARY 4   YES
 12    P_MQMD_EXPIRY                 BIG  BINARY 4   YES
 13    P_MQMD_FEEDBACK               BIG  BINARY 4   YES
 14    P_MQMD_ENCODING               BIG  BINARY 4   YES
 15    P_MQMD_CODEDCHAR              BIG  BINARY 4   YES
 16    P_MQMD_FORMAT                 TEXT TEXT   8
 17    P_MQMD_PRIORITY               BIG  BINARY 4   YES
 18    P_MQMD_PERSISTEN              BIG  BINARY 4   YES
 19    P_MQMD_MSGID                  TEXT TEXT   24
 20    P_MQMD_CORRELID               TEXT TEXT   24
 21    P_MQMD_BACKOUTCO              BIG  BINARY 4   YES
 22    P_MQMD_REPLYTOQ               TEXT TEXT   48
 23    P_MQMD_REPLYTOQM              TEXT TEXT   48
 24    P_MQMD_USERIDENT              TEXT TEXT   12
 25    P_MQMD_ACCOUNTIN              TEXT TEXT   32
 26    P_MQMD_APPLIDENT              TEXT TEXT   32
 27    P_MQMD_PUTAPPLTY              BIG  BINARY 4   YES
 28    P_MQMD_PUTAPPLNA              TEXT TEXT   28
 29    P_MQMD_PUTDATE                TEXT TEXT   8
 30    P_MQMD_PUTTIME                TEXT TEXT   8
 31    P_MQMD_APPLORIGI              TEXT TEXT   4
 32    P_MQMD_GROUPID                TEXT TEXT   24
 33    P_MQMD_MSGSEQNUM              BIG  BINARY 4   YES
 34    P_MQMD_MSGFLAGS               BIG  BINARY 4   YES
 35    P_MQMD_ORIGINALL              BIG  BINARY 4   YES
 36    MQPMO_VERSION                 BIG  BINARY 4   YES
 37    MQPMO_OPTIONS                 BIG  BINARY 4   YES
 38    MQPMO_KNOWNDESTC              BIG  BINARY 4   YES
 39    MQPMO_UNKNOWNDES              BIG  BINARY 4   YES
 40    MQPMO_INVALIDDES              BIG  BINARY 4   YES
 41    MQPMO_RESOLVEDQN              TEXT TEXT   48
 42    MQPMO_RESOLVEDQM              TEXT TEXT   48
 43    MQPMO_RECSPRESEN              BIG  BINARY 4   YES
 44    MQPMO_PUTMSGRECF              BIG  BINARY 4   YES
Add your message fields here
```

Initializing the MQPUT Interface to its default usable state

Special initialization. This Interface requires special initialization—that is, before you can use this Interface, you must call a special function in order to initialize the Interface to its default usable state.

The initialization process will not alter any of the user fields that you added to the end of the Interface layout.

To review the settings for the default state of an MQPUT, refer to *MQSERIES Application Programming Reference*, SC33-1673.

Initialization procedure. To initialize the Interface, perform the following:

1. Place the "INITPUT" string in the PUT_FUNCTION field.
2. Call the Interface.

Sample code. See the following sample code:

```
140 INTERFACE MQPUT1("MQPUT1",PASSWORD)
150 PUT_FUNCTION="INITPUT"
160 CALL MQPUT1
```

Description of sample code. The above MANTIS program performs the following:

1. Loads the MQPUT1 Interface.
2. Sets the PUT_FUNCTION variable.
3. Calls the MQPUT1 Interface

Changing the initialized Interface in order to send a message

Once you initialize the Interface, you can perform the procedure for sending a message.

Procedure for sending a message. To send a message, perform the following:

1. Change PUT_FUNCTION to "PUT" or "MQPUT".
2. Set the PUT_HANDLE variable to a valid handle from a previous MQCONNECT.
3. Set the PUT_LENGTH variable defining the length of your user data. This value does not include the length of the fields prior to your user data.
4. Call the Interface.

Sample code. See the following sample code:

```
170 PUT_FUNCTION="PUT"
180 PUT_HANDLE=SAVE_HANDLE2
190 PUT_LENGTH = 100
200 CALL MQPUT1
```

Description of sample code. The above MANTIS program performs the following:

1. Sets the PUT_FUNCTION variable.
2. Sets the PUT_HANDLE variable to a valid handle from a previous MQCONNECT.
3. Sets the PUT_LENGTH variable to 100. This is the total length of the user data area.
4. Calls the MQGET1 Interface.

Using the MQROLLBACK Interface to rollback to a previous sync point and reverse all previous message GETs and PUTs

The MQROLLBACK Interface enables you to perform the following:

1. Rollback to a previous sync point.
2. As MQSeries and its mqback API specify, reverse all previous message GETs and PUTs.

Changing the MQROLLBACK Interface layout

Changing fields that are already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQROLLBACK Interface layout figure

The Interface layout is shown below:

```
Page No : 1 :              Interface Area Layout          2001/09/26
MASTER:MQROLLBACK                                         07:55:47
Element Count : 7                          Element Size : 108
ELEM -------------NAME------------- TYPE FORMAT LEN SIGN DEC DIM -ATTRIB

1     ROL_FUNCTION                  TEXT TEXT   8
2     ROL_HANDLE                    BIG  BINARY 4
3     ROL_COMPCODE                  BIG  BINARY 4
4     ROL_REASON                    BIG  BINARY 4   YES
5     ROL_IDENT                     BIG  BINARY 4
6     ROL_DMPLENGTH                 BIG  BINARY 4
7     ROL_DMPFILENAME               TEXT TEXT   80
```

Initializing the MQROLLBACK Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, perform the following. These steps will establish a sync point based on the object (CON_HANDLE), pointed to by the HANDLE, that was returned by a previous MQCONNECT:

1. Set the ROL_HANDLE variable to a valid handle that was returned by the MQCONNECT Interface.
2. Set the ROL_FUNCTION variable to the string “ROLLBACK” or “MQBACK”.
3. Call the Interface.

Sample code. See the following sample code:

```
270 INTERFACE MQROLLBACK("MASTER:MQROLLBACK",PASSWORD)
280 ROL_FUNCTION="ROLLBACK"
290 ROL_HANDLE=SAVE_HANDLE1
300 CALL MQROLLBACK
```

Description of sample code. The above MANTIS program code performs the following:

1. Loads the MQROLLBACK Interface.
2. Sets the required fields (ROL_FUNCTION and ROL_HANDLE).
3. Calls the Interface.

Transaction Server and MQROLLBACK

MQROLLBACK is not supported in Transaction Server. In Transaction Server, transaction support of MQSeries messages falls under normal MANTIS transaction guidelines.

Using the MQTM Interface to map the MQSeries trigger data to the MANTIS MQTM Interface

The MQTM Interface maps the MQSeries trigger data to the MANTIS MQTM Interface.

For more information on using MANTIS as an MQSeries trigger message handler, see 7. MQSeries/MANTIS triggering and 10. General UNIX and Mainframe considerations.

Changing the MQTM Interface layout

Changing fields that are already present in this Interface layout. Do not change this Interface layout. Doing so will corrupt the Interface and will result in the premature termination of MANTIS.

Adding user data to the end of this Interface layout. You need not add user data to this Interface layout.

MQTM Interface layout figure

The Interface layout is shown below:

```
Page No : 1 :              Interface Area Layout          2001/09/26
MASTER:MQTM                                               09:25:33
Element Count : 19                          Element Size : 844
ELEM -------------NAME------------- TYPE FORMAT LEN SIGN DEC DIM -ATTRIB

1    TM_FUNCTION                    TEXT TEXT   8
2    TM_HANDLE                      BIG  BINARY 4
3    TM_COMPCODE                    BIG  BINARY 4
4    TM_REASON                      BIG  BINARY 4   YES
5    TM_IDENT                       BIG  BINARY 4
6    TM_DMPLENGTH                   BIG  BINARY 4
7    TM_DMPFILENAME                 TEXT TEXT   80
8    TM_MEMADDR                     TEXT TEXT   4
9    TM_STRUCID                     TEXT TEXT   4
10   TM_VERSION                     BIG  BINARY 4
11   TM_QNAME                       TEXT TEXT   48
12   TM_PROCESSNAME                 TEXT TEXT   48
13   TM_TRIGGERDATA                 TEXT TEXT   64
14   TM_APPLTYPE                    BIG  BINARY 4
15   TM_APPLID1                     TEXT TEXT   128
16   TM_APPLID2                     TEXT TEXT   128
17   TM_ENVDATA                     TEXT TEXT   128
18   TM_USERDATA                    TEXT TEXT   128
19   TM_QMGRNAME                    TEXT TEXT   48
```

Initializing the MQTM Interface

Special initialization. This Interface does not require special initialization before you can use it.

Initialization procedure. To initialize this Interface, set TMC_FUNCTION to the string "INITMQTM".

Sample code. See the following sample code:

```
370 INTERFACE MQTM("MASTER:MQTM",PASSWORD)
380 TMC_FUNCTION = "INITMQTM"
390 CALL MQTM
```

Description of sample code. In order to retrieve the MQSeries trigger message data, the preceding MANTIS program performs the following:

4. Loads the MQTM Interface.
5. Sets the required field (TMC_FUNCTION).
6. Calls the Interface.

Upon successful return, there is sufficient information to enable the MANTIS program to connect to the appropriate queue and retrieve the message that triggered the event.

7. MQSeries/MANTIS triggering

This chapter describes how to use MANTIS as an MQSeries trigger handler. It is organized in the following sections:

- General MANTIS trigger considerations.
- UNIX MQSeries/MANTIS trigger considerations.
- Mainframe MQSeries/MANTIS trigger considerations.

Using MANTIS for triggering is not currently supported under Mainframe Batch.

General MANTIS trigger considerations

MQSeries can invoke MANTIS in the background, so that MANTIS can handle a particular message type being sent to a queue that is defined as a trigger queue.

To clarify: this interface is designed to be used in a background environment. No results, other than errors, will be returned to the application when the application is executed interactively.

Procedure for using MANTIS as a trigger handler

To use MANTIS as a trigger handler, perform the following:

7. Use MQSeries queue definitions to associate MANTIS with a particular MQSeries queue and message type. For more information, refer to the following:
   - *MQSeries Application Programming Guide*, SC33-0807
   - One of the following platform-specific sections in this chapter. See UNIX MQSeries/MANTIS trigger considerations or Mainframe MQSeries/MANTIS trigger considerations.
8. Tell MANTIS which user, password, and program to execute in order to handle the trigger message. Each platform requires a different procedure. For platform-specific information, see one of the following:
   - UNIX MQSeries/MANTIS trigger considerations
   - Mainframe MQSeries/MANTIS trigger considerations

Programs that illustrate the trigger-handling process

MANTIS includes a set of programs to illustrate the trigger-handling process.

See 8. MQSeries/MANTIS example programs for descriptions of the following:

- How these trigger-handling programs work
- Front-end components needed to complete the trigger-handling process

The following two platform-specific sections describe components needed to complete the trigger-handling process.

Writing a MANTIS application program to handle the triggered event

Once you have customized the trigger-handler front-end for your environment, you must write a MANTIS application program to handle the triggered event.

This MANTIS application program must call the MQTM Interface in order to retrieve the following:

- **For UNIX users:** MQTMC2 record description.
- **For Mainframe CICS users:** MQTM record description.

For details on the MQTMC2 or MQTM record description, refer to *MQSERIES Application Programming Reference*, SC33-1673. Each of these two record descriptions contains detailed information about the message that caused the trigger event.

After the MANTIS application calls the MQTM Interface, it must perform the following:

9. Interrogate the fields.
10. Open the appropriate queue.
11. Retrieve the message.

Sample program for sending a message to a trigger queue

For an example of how to send a message to a queue defined as a trigger queue, study the MASTER:MQ_TRIGGER sample program (see MQ_TRIGGER).

Sample program for handling an MQSeries trigger event

For an example of how to handle an MQSeries trigger event, study the MASTER:MQ_HANDLER sample program (see MQ_HANDLER).

UNIX MQSeries/MANTIS trigger considerations

The trigger.sh script as a model for your trigger handler

Use the trigger.sh script as a model when you develop your own trigger handler. This script, one of the MQSeries/MANTIS example programs, is located in the $MANTIS_ROOT/libmqs directory that accompanies the UNIX version of MANTIS.

Steps required for trigger handling

For trigger handling, the following must occur:

12. You must associate trigger.sh, or a program that you have modeled after it, with the trigger queue.
13. MQSeries must execute trigger.sh, or a program that you have modeled after it, as the trigger handler.

Procedure for constructing a trigger handler

Perform the following in trigger.sh (or in a program that you have modeled on it):

14. Redirect all MANTIS terminal output to a log file.
15. this because MANTIS executes the user, password, and program in batch form, in the background.
16. Enter settings for the MANTIS working environment, such as MANTIS_PATCH, MANTIS_ROOT, MANTIS_CLASS, and so on.
17. Set the MQTMC2 environment variable to the trigger record passed from MQSeries.
18. Make any other changes specified by the documentation inside trigger.sh.

Since MQSeries invokes the trigger script and ultimately MANTIS, you must give MQSeries full READ/WRITE/EXECUTE privileges to all files within the $MANTIS_ROOT directory structure.

MQSeries and MANTIS procedure for handling the triggered event

Description of the procedure for handling the triggered event

MQSeries and MANTIS take the following steps (shown in Figure depicting the procedure for handling the triggered event) to cooperatively handle the triggered event:

19. An MQSeries-enabled application (MANTIS or any other application) sends a message to an application queue (this application queue must be defined as being trigger-enabled).

The following figure depicts this MQSeries-enabled application as residing on the same system as the application queue, but there are no restrictions on the type or location of the application or system. The only requirement is that MQSeries routes the message to the correct destination.

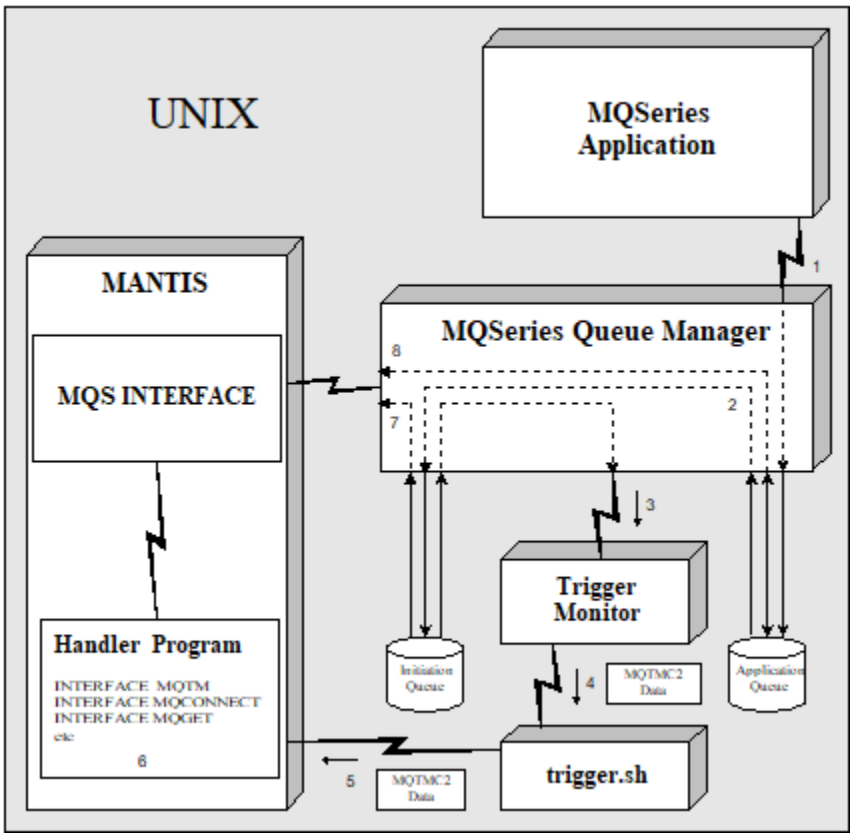
20. MQSeries copies the message to the initiation queue.
21. The trigger monitor program that has been watching the initiation queue performs the following:
    a. Detects an inbound message.
    b. Launches the appropriate application, trigger.sh, to handle the message (the MQTMC2 data structure is passed to trigger.sh as an argument).

22. Trigger.sh performs the following:
    a. Places the MQTMC2 data into the MQTMC2 environment variable.
    b. Sets up the MANTIS environment.
    c. Executes MANTIS in batch mode, using the appropriate USER, PASSWORD, and PROGRAM.
23. MANTIS signs on via the specified USER and PASSWORD.
24. The MANTIS program specified by trigger.sh in step 4 begins executing.

    This program is written to handle one or more trigger message types. This program uses the MQTM Interface to retrieve the MQTMC2 data that trigger.sh placed in the environment variable MQTMC2.
25. The MANTIS program performs the following:
    a. Connects to the initiation queue returned in the MQTM Interface.
    b. Retrieves the message that caused the trigger event.
26. The application may choose to connect to other application queues to send and receive messages; there are no restrictions on what the application does next.

Figure depicting the procedure for handling the triggered event

MQSeries and MANTIS take the following steps (see Description of the procedure for handling the triggered event) to cooperatively handle the triggered event:



Mainframe MQSeries/MANTIS trigger considerations

The CSOXTRIG front-end application as a model for your trigger handler

Use the CSOXTRIG front-end application (CICS transaction) as a model when you develop your own trigger handler. Cincom provides this application, one of the MQSeries/MANTIS example programs, in both executable and source forms.

Steps required for trigger handling

For trigger handling, the following must occur:

27. You must associate CSOXTRIG, or a program modeled after it, with the trigger queue.
28. MQSeries must execute CSOXTRIG, or a program modeled after it, as the trigger handler.

Procedure for constructing a trigger handler

Perform the following in CSOXTRIG (or in an application that is modeled after it):

29. Customize the BTRANID setting.
    This is the ID of the MANTIS background transaction that this front-end application will start.
30. Customize the BUID setting.
    This is the user that is executed in batch MANTIS.
31. Customize the BPSW setting.
    This is the password that is executed in batch MANTIS.
32. Customize the BTRIG setting.
    This is the program that is executed in batch MANTIS.

You need not customize the MSHMEM setting—it will be automatically set to the address of the Shared GETMAIN area that will contain the MQTM trigger to be passed from MQSeries.

33. For further setting changes, review the documentation within CSOXTRIG.

MQSeries and MANTIS procedure for handling the triggered event

Description of the procedure for handling the triggered event

MQSeries and MANTIS take the following steps (shown in Figure depicting procedure for handling the triggered event) to cooperatively handle the triggered event:

34. An MQSeries-enabled application (MANTIS or any other application) sends a message to an application queue (this application queue must be defined as being trigger-enabled).

The following figure depicts this MQSeries-enabled application as a UNIX application residing on the same system as the application queue, but there are no restrictions on the type or location of the application or system. The only requirement is that MQSeries routes the message to the correct destination.

35. MQSeries copies the message to the initiation queue.
36. The CICS trigger monitor program (CKTI) that has been watching the initiation queue performs the following:

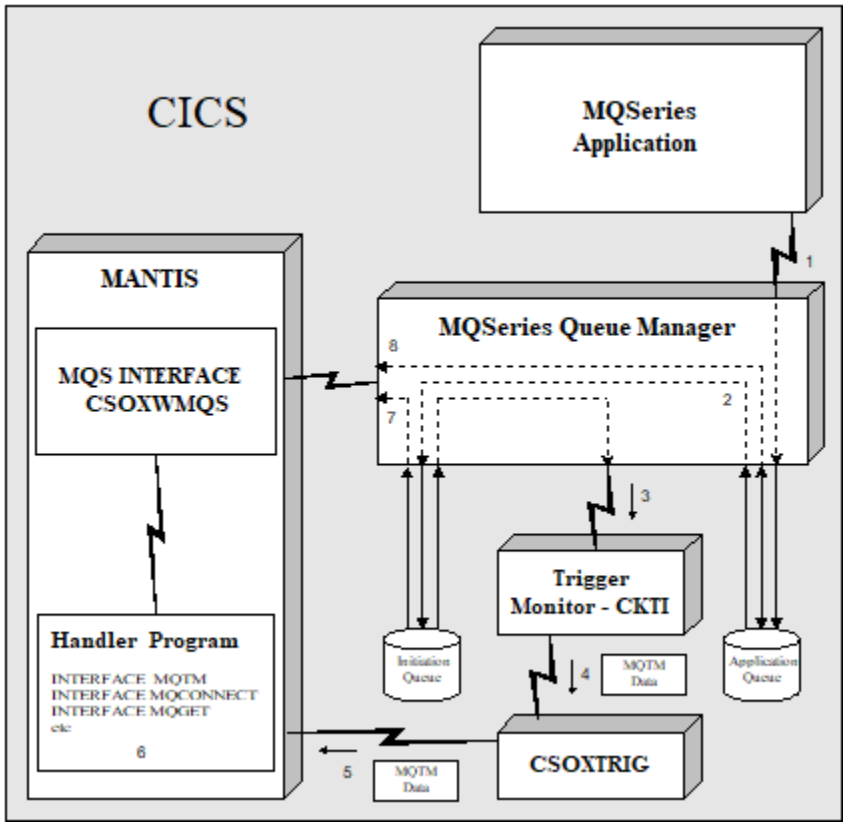
a. Detects an inbound message.
b. Launches the appropriate application, CSOXTRIG, to handle the message (the MQTM data structure is passed to CSOXTRIG as an argument on the CICS start).

37. CSOXTRIG performs the following:
    a. Places the MQTM data into a shared getmain area.
    b. Passes the getmain area address, USER, PASSWORD, and PROGRAM to MANTIS on the start.
38. MANTIS, running as a background MANTIS task, signs on via the specified USER and PASSWORD.
39. The specified MANTIS program begins executing.

    This program is written to handle one or more trigger message types. This program uses the MQTM Interface to retrieve the MQTM data that CSOXTRIG placed in the getmain area by CSOXTRIG.
40. The MANTIS program performs the following:
    a. Connects to the initiation queue returned in the MQTM Interface.
    b. Retrieves the message that caused the trigger event.
41. The application may choose to connect to other queues to send and receive messages; there are no restrictions on what the application does next.

Figure depicting procedure for handling the triggered event

MQSeries and MANTIS take the following steps (Description of the procedure for handling the triggered event) to cooperatively handle the triggered event:



8. MQSeries/MANTIS example programs

Cincom provides MANTIS example programs along with the MANTIS distribution. These programs test the MQSeries Interface and demonstrate its usage. Find them under the MASTER user.

The following sections describe these programs in more detail.

MANTIS program	Associated screen	Description
MQ_HANDLER	MQ_HANDLER	Working MANTIS program that serves as a message trigger handler.
MQ_HANDLER@	MQ_HANDLER	MANTIS source code for MQ_HANDLER.
MQ_INIT	N/A	MANTIS subroutine containing MQSeries constants.
MQ_SAMPLE	MQ_SAMPLE	Working MANTIS program that tests each MQSeries Interface.
MQ_SAMPLE@	MQ_SAMPLE	MANTIS source code for MQ_SAMPLE.
MQ_TRIGGER	N/A	Working MANTIS program that SENDs a message to trigger MQ_HANDLER.
MQ_TRIGGER@	N/A	MANTIS source code for MQ_TRIGGER.

MQ_INIT

By itself, MQ_INIT only allocates and initializes MANTIS variables that are used as constants for the various MQSeries Interface fields.

MQ_INIT is built as a subroutine that can be copied into a user program. Use it as one of the following:

- External subroutine
- Internal subroutine
- Component

For more information on MQ_INIT, see 5. Constants.

MQ_SAMPLE

MQ_SAMPLE is an example program that performs the following:

42. Tests each MQSeries Interface.
43. For each Interface, displays either a “SUCCESSFUL” status or an “ERROR” status.

Uses for MQ_SAMPLE

You can use MQ_SAMPLE:

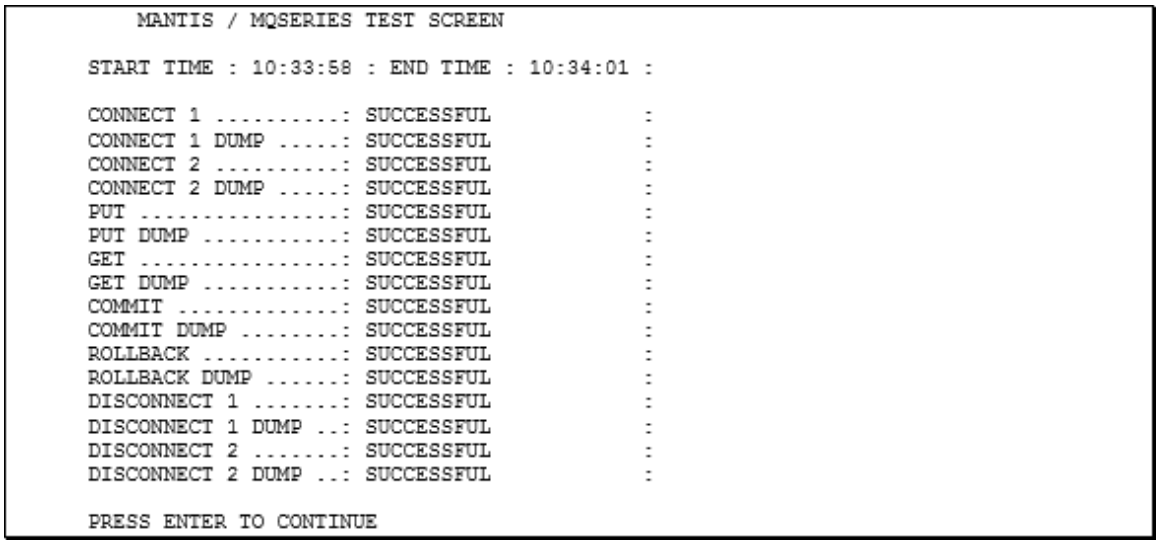
- As a test to find out whether the connection between MANTIS and MQSeries is working properly.
- As an example for developing MANTIS programs that perform MQSeries messaging.

Queue used for sending and receiving messages

MANTIS uses the SYSTEM.DEFAULT.LOCAL.QUEUE as the queue for sending and receiving messages. Therefore, you must be authorized to perform write operations to this queue. For SYSTEM.DEFAULT.LOCAL.QUEUE, you may substitute any queue to which you have write privileges.

UNIX screen shot of MQ_SAMPLE

The UNIX screen shot of MQ_SAMPLE below shows a “SUCCESSFUL” status for all MQSeries Interfaces.



MQ_SAMPLE’s errors for COMMIT and ROLLBACK functions under mainframe CICS

For the mainframe CICS environments, MQ_SAMPLE will generate an error (-22) for the COMMIT function and an error (-29) for the ROLLBACK function because they are not supported.

MQ_HANDLER

MQ_HANDLER is a MANTIS example program that demonstrates how MANTIS can be used as an MQSeries trigger handler.

Abilities necessary for any handler to possess

MQ_HANDLER demonstrates the abilities necessary for any handler to possess. A handler must perform the following:

44. Receive the Trigger Record by calling the MANTIS Interface MQTM (For Mainframe, this is MQTM; for Unix, this is MQTMC2). The MANTIS application can review the data in this Interface's fields, in order to determine the next step.
45. Using the MQCONNECT Interface, connect to the queue containing the message to which this program was triggered.
46. Retrieve the triggered message via the appropriate MQGET Interface. Each message type will require its own Interface that is modeled after the MQGET Interface.
47. Disconnect from the queue via the MQDISCONNECT Interface.
48. Take appropriate action by connecting, sending, or receiving other messages.

MEMADDR argument to MQ_HANDLER

Mainframe CICS uses the MEMADDR argument to MQ_HANDLER. The MEMADDR argument's purpose is to serve as a storage address of the MQTM record that CSOXTRIG passed to MANTIS. For more information, see Mainframe MQSeries/MANTIS trigger considerations.

Running MQ_HANDLER interactively vs. running it automatically

Cincom did not design MQ_HANDLER to be run interactively; rather, Cincom designed it to be executed by the MQSeries trigger monitor. If you attempt to run MQ_HANDLER interactively, MANTIS aborts. If the trigger queue and definitions are set up properly, MQSeries will execute MQ_HANDLER automatically when you interactively run MASTER:MQ_TRIGGER.

MQ_TRIGGER

MQ_TRIGGER

MQ_TRIGGER, a MANTIS example program, sends a message to a trigger-enabled queue called TRIGGER.QUEUE. MQ_TRIGGER works with the MQ_HANDLER program.

The MQ_HANDLER program performs the following:

- Serves as a trigger handler.
- Responds to the message that MQ_TRIGGER sends to TRIGGER.QUEUE.

Before using MQ_TRIGGER, review the material under MQ_HANDLER.

MQ_TRIGGER sample output screen

Below is an MQ_TRIGGER sample output screen:

```
                    INVOKE TRIGGER


THIS PROGRAM WILL SEND A MESSAGE TO THE TEST TRIGGER.QUEUE
THEREBY CAUSING THE MQSERIES TRIGGER MONITOR TO TRIGGER THE
MANTIS SUPPLIED FRONT END TO INITIALIZE THE ENVIRONMENT
NECESSARY TO ALLOW A BACKGROUND MANTIS TO PROCESS THE
MESSAGE SENT BY THIS PROCESS.

MESSAGE SENT: THIS IS THE TRIGGER MSG FROM MANTIS    :

MESSAGE SENT STATUS : SUCCESS :

MESSAGE RECV: * THIS IS THE TRIGGER RESPONSE MSG * 09:06:07 :

MESSAGE RECV STATUS : SUCCESS :

PROGRAM COMPLETE - PRESS ENTER TO CONTINUE
```

GETERR(2033)

Regarding the GETERR(2033) error message, consider the following:

- **Improperly configured trigger queues or definitions.** If the sent message works properly, most errors are related to the GET in MQ_TRIGGER and are usually due to improperly configured trigger queues or definitions. To locate the source of this error, perform the following:
  - For CICS—Review the CICS JOBLOG to see if the trigger monitor successfully started CSOXTRIG.
  - For both CICS and UNIX—Review the MANTIS DUMP file for indications of how far the MASTER:MQ_HANDLER program progressed before it failed. For more information on the MQSeries/MANTIS Dump mechanism, see 9. MQSeries/MANTIS diagnostic considerations.

- **Timing problem.** Occasionally, a timing problem causes error 2033. This happens because insufficient time was provided in which to invoke, execute, and receive the message from MQ_HANDLER. To resolve this problem, simply run MQ_TRIGGER again.

9. MQSeries/MANTIS diagnostic considerations

Diagnosing a MANTIS program error

When a MANTIS program causes errors while it is attempting to CONNECT, GET, PUT, etc, perform the following steps to diagnose the problem:

49. Check the Interface name for returned values.

    For detailed descriptions of these return values, see 4. Errors.

50. Review the COMPCODE field in the Interface layout.

    The COMPCODE field contains a number that represents problem severity. For more information about the COMPCODE field in the Interface views, see 2. Fundamental usage.

51. Review the REASON field in the Interface layout.

    The REASON field contains a NON-ZERO value if an error occurs. This value will be a positive value if the MQSeries API generated the error. A negative value in the REASON field represents an internal error in the interface. For more information on this number:

    - For positive REASON values, perform the following:
        - Compare this number with the MQRC_ values found in the MANTIS program MQ_INIT.
        - Look up a description of the number in *MQSERIES Application Programming Reference*, SC33-1673.
    - For negative REASON values, review Negative REASON codes

    For more information about the REASON field in the Interface views, see 2. Fundamental usage.

52. Perform a DUMP of the Interface layout. This may be required under severe conditions—especially when CINCOM Technical Support is involved. For more information on dumping MQSeries Interface views, see Dumping MQSeries Interface views.

Dumping MQSeries Interface views

Each MQSeries Interface layout contains a FUNCTION field that you must initialize to the function you would like to perform. For example:

- MQCONNECT contains a field called CON_FUNCTION. Set this field to “INITCONN” or “CONNECT”.
- MQDISCONNECT contains a field called DIS_FUNCTION. Set this field to “DISCONN”.

There’s an additional value to which you can set all FUNCTION fields: “DUMP”. “DUMP” signals the Interface subroutines to dump the contents of the Interface layout.

UNIX sample of a dumped MQCONNECT Interface

Below is a UNIX sample of an MQCONNECT Interface layout that has been dumped:

```
Dumping Connect View Dump Length = 228
  0x00000000 - 20202020 20202020 44554d50 20202020  *           DUMP      *
  0x00000010 - 00000002 20202020 20202020 20202020  * ....                 *
  0x00000020 - 20202020 20202020 20202020 20202020  *                       *
  0x00000030 - 00000000 00000000 00000001 00000000  * .... .... .... .... *
  0x00000040 - 20202020 20202020 20202020 20202020  *                       *
  0x00000050 - 20202020 20202020 20202020 20202020  *                       *
  0x00000060 - 20202020 20202020 20202020 20202020  *                       *
  0x00000070 - 00002011 00000001 53595354 454d2e44  * .. . .... SYST EM.D *
  0x00000080 - 45464155 4c542e4c 4f43414c 2e515545  * EFAU LT.L OCAL .QUE *
  0x00000090 - 55452020 20202020 20202020 20202020  * UE                   *
  0x000000a0 - 20202020 20202020 00000000 00000000  *           .... .... *
  0x000000b0 - 00000000 00000000 00000000 00000000  * .... .... .... .... *
  0x000000c0 - 00000000 00000000 00000000 00000000  * .... .... .... .... *
  0x000000d0 - 00000000 00000000 00000000 00000000  * .... .... .... .... *
  0x000000e0 - 00000000                             * ....                *
```

Procedure for dumping the failing Interface layout

Special initialization. Although an Interface does not require special initialization to dump, it makes sense to dump the Interface layout after it has failed to perform as intended. Dumping the layout, before calling the Interface as the view is intended to do, could result in error being returned in the REASON field.

Procedure for dumping the failing Interface. To dump the failing Interface layout, set the FUNCTION variable to the string “DUMP”.

Code sample. See the following code:

```
180  CON_FUNCTION="INITCONN"
190  CALL MQCONNECT
200  CON_FUNCTION="CONNECT"
210  MQOD_OBJECTNAME="SYSTEM.DEFAULT.LOCAL.QUEUE"
220  MQOD_OBJECTTYPE=MQOT_Q
230  CON_OPTIONS=MQOO_INPUT_AS_Q_DEF+MQOO_OUTPUT
240  CALL MQCONNECT
250  IF MQCONNECT<>""
260    CON_FUNCTION="DUMP"
270  . CON_DMPFILENAME="MMQD"
280    CALL MQCONNECT
290    STOP
300  END
```

Description of the code sample. The preceding MANTIS code performs the following:

53. Initializes the MQCONNECT Interface by performing the following:
    a. Specifying “INITCONN” in the FUNCTION field.
    b. Properly setting the remaining connection fields.
54. Resets the FUNCTION field to “CONNECT”.
55. Calls MQCONNECT to connect to the SYSTEM.DEFAULT.LOCAL.QUEUE.
56. Once control is returned, the Interface CALL is checked for errors by comparing the Interface name to “”.
57. If an error occurs, sets the FUNCTION field is set to “DUMP”.
58. Sets the DMPFILENAME to "MMQD".
59. Calls MQCONNECT again to dump the Internet layout.

System-specific dump file descriptions

Consider the dump file description that is relevant to your system. Dump file name: Each view has a field called DMPFILENAME. This field relates to the output of the “DUMP” function but is platform specific in it’s meaning. For all platforms, output of the “DUMP” is appended to the file, if it already exists. If you use the “DUMP” feature of an MQSeries interface the DMPFILENAME must be set, there is no default name.

- **UNIX.** The output file name may also contain a full path or default to the current working directory. Make sure you have write privileges to the file regardless of its location. If the dump file name does not already exist, MANTIS creates it.
- **Mainframe CICS.** The filename associated by the DMPFILENAME field refers to a Transient Data Queue. Therefore, the naming restrictions of a TD Queue apply. Before using the Queue name, it must be created with the following DCB:

```
Organization       PS
Record format      VBA
Record length      100
Block size     23476 (3380)
Allow at least 1 cylinder for primary Space allocation.
```

  In addition to the above DCB for the mmqd, the following CICS (Transaction Server) definitions are required:
  - NAME=MMQD
  - TYPE=EXTRA
  - DATABUFFERS=10
  - DDNAME=MMQD
  - OPENTIME=INITIAL
  - TYPEFILE=OUTPUT
- **z/OS Batch.** The filename associated by the DMPFILENAME field refers to a DD name in your JCL. Therefore, the naming restrictions of a DD Name apply. A dataset with 100 byte records and blocked accordingly would be sufficient.
- **z/VSE Batch.** The filename associated by the DMPFILENAME field refers to a DLBL name in your JCL. Therefore, the naming restrictions of a DLBL Name apply. A dataset with a block size of 4096 would be sufficient.

Dump length

The Interface type determines the dump length. Once an Interface has been used, the IDENT field in the Interface layout is set internally to signify Interface type. An attempt to dump this Interface before the Interface is used internally will generate an error code in the REASON field of that view.

Overriding the dump length with DMPLENGTH

You can override the dump length by entering a numeric size in the DMPLENGTH field. However, do not specify a size larger than the Interface itself; if you do, MANTIS may end prematurely. You must set DMPLENGTH for any dumps of MQGET and MQPUT Interfaces that include user data fields, because the dump routine does not know the exact size of the Interface layout (including the user data fields).

10. General UNIX and Mainframe considerations

MQSeries Client Configuration for UNIX MANTIS

For UNIX only, MQSeries applications can be built and then linked to client or server runtime libraries. The MQSeries support for UNIX Mantis has been built and linked to the client libraries. Whether the application has been linked with client or server libraries has little or no impact on the application API. However, it does have an impact on administering MQSeries and MQSeries-enabled applications.

To enable messaging between an MQSeries client (MANTIS) and server, an MQI Channel must be created. The MQ API makes no reference to this channel in any way; it's strictly an MQSeries administrative issue. Based on many factors, there are several ways to set up channels in an MQSeries client/server environment. Because of these factors, and also because of possible variations between customer environments, channel configuration recommendations are beyond the scope of this document. However, the creation of channels, as well as a properly configured MQSeries client/server environment, are absolute requirements for using the MANTIS/MQSeries interface.

For information on properly configuring an MQSeries client environment, refer to *WebSphere Clients*, GC34-6058.

Installation considerations

UNIX

Consider the following for MQSeries installation on the UNIX platform:

- **Things to verify after MQSeries is installed.** After MQSeries has been installed and is running properly, verify the following:
  - MANTIS is in proper working order.
  - The UNIX environment variable MANTIS_SHRLIB is pointing to $MANTIS_ROOT/libmqs. This enables MANTIS to load the MQSeries internal Interface.

For information on setting MANTIS_SHRLIB, refer to the "Logical names" topic in *MANTIS Administration*, P39-1321.

- **Enabling appropriate patches.** To authorize the use of MQSeries, enable the MANTIS Security patch (Option, Product 13). Before developing applications using MQSeries, consult Cincom's MANTIS technical support in order to get the MANTIS Security patch and any additional updates.
- **Installing and running MQSeries.** You must install and run MQSeries before MANTIS can communicate with the Queue Manager.
- **Other installation considerations.** If you followed the documented installation procedure, no other installation considerations are required. To find the documented installation procedure to follow, refer to the version of *MQSeries Quick Beginning Guide* for the appropriate platform.

Mainframe

Consider the following for MQSeries installation for Mainframe:

- **Enabling the MQSeries feature.** In the Mainframe environment, you must enable the MQSeries feature in order to run it.
- **Authorization error with MQSeries Interfaces.** If you ever receive an authorization error when you attempt to use any of the MQSeries Interfaces, consult your Mantis Administrator.
- **Under the z/OS operating system:**
  a. Make sure that the "C" runtime library is in your CICS start JCL, under the DFHRPL DD statement:

```
// DD  DISP=SHR,DSN=CEE.SCEERUN            'C' Runtime
```

  b. Add the MQSeries runtime to the CICS start JCL, under the DFHRPL DD statement:

```
// DD DISP=SHR,DSN=CSQ.V5R2M0.SCSQANLE
// DD DISP=SHR,DSN=CSQ.V5R2M0.SCSQCICS
// DD DISP=SHR,DSN=CSQ.V5R2M0.SCSQAUTH
// DD DISP=SHR,DSN=CSQ.V5R2M0.SCSQLOAD
```

- **Under the z/VSE operating system.** Make sure that the "C" runtime and MQSeries library are already allocated to your partition. If not, your CICS start JCL should reference them in the LIBDEF search chain statement:

```
PRD2.SCEEBASE
PRD2.MQSERIES
```

If you followed the documented installation procedure, no other installation considerations are required. To find the appropriate installation procedure, refer to the following:

- *MQSeries for z/OS and Planning Guide*, GC34-5650
- *MQSeries for z/OS System Setup Guide*, SC34-5651
- *MQSeries for z/OS System Administration Guide*, SC34-5652
- *MQSeries for VSE/ESA V2.1.1 System Management Guide*, GC34-5364

MQCONNECT

UNIX

Consider the following for the UNIX implementation of the MQCONNECT Interface:

- It has no restrictions.
- It conforms to the routine requirements for:
  - MQSeries MQI mqconn
  - MQSeries MQI mqopen
  - Multiple connections can be made to:
  - The same object
  - Different objects
  - Multiple Queue Managers

Mainframe

Consider the following for the Mainframe implementation of the MQCONNECT Interface:

- **Connections under CICS.** Consider the following restrictions:
  - Only one Queue Manager may be connected to at one time. However, multiple objects controlled by that queue manager may be connected to concurrently.
  - Under Mainframe CICS only—MANTIS imposes a restriction of 50 concurrent connections. Attempting to execute more than 50 concurrent connections results in an error -43 in the RESULT field of the MQCONNECT interface. Consider the following:
  - In CICS Conversational Mode—During testing of your MQSeries MANTIS program, if you repeatedly execute a program containing MQCONNECT without successfully calling MQEXIT or MQDISCONNECT, MANTIS will probably produce error -43.
  - In CICS Pseudo-Conversational mode—You would have to specifically call MQCONNECT 50 times in a row, without performing a screen converse, in order for MANTIS to produce error -43.
  - If error -43 is generated, perform one of the following:
    - Reduce the number of calls to MQCONNECT.
    - Issue the MQDISCONNECT interface call.
    - Issue the MQEXIT interface call.
- **CICS Pseudo-Conversational Mode restrictions.** Consider the following restrictions for this terminal mode:
  - When the user performs a MANTIS CONVERSE, MQSeries connections are lost and MQSeries connections are not maintained.
  - The user is responsible for reconnecting to the MQSeries Object after each CONVERSE.
  - Retaining a connection handle across a CONVERSE does not maintain the connection. If the user uses the connection handle after a CONVERSE, an Invalid Handle Error results.
- **CICS Conversational Mode restrictions.** CICS Conversational Mode does not have the same restrictions as CICS Pseudo-Conversational Mode. All connections are maintained across a CONVERSE.
- **MQCONNECT conforms to the requirements of the following routines:**
  - MQSeries MQI mqconn

- MQSeries MQI mqopen

MQDISCONNECT

UNIX

Consider the following for the UNIX implementation of the MQDISCONNECT Interface:

- It has no restrictions.
- It conforms to the routine requirements for the following:
  - MQSeries MQI mqclose
  - MQSeries MQI mqdisc

Mainframe

Consider the following for the Mainframe implementation of the MQDISCONNECT Interface:

- It has no restrictions.
- It conforms to the routine requirements for the following:
  - MQSeries MQI mqclose
  - MQSeries MQI mqdisc

MQGET

UNIX

Consider the following for the UNIX implementation of the MQGET Interfaces:

- It has no restrictions
- It conforms to the routine requirements for MQSeries MQI mqget

Mainframe

Consider the following for the Mainframe implementation of the MQGET Interface:

- **MQMD fields not used by the MQGET Interface.** Consider that for the Mainframe, several MQMD (Message Descriptor) fields do not exist. Although these fields appear in the MQGET Interface layout for the Mainframe, the MQSeries Interface does not use them. These fields are:
  - MsgSeqNumber
  - MsgFlags
  - OriginalLength
  - GroupId
- **MQGET conforms to the requirements of the MQSeries MQI mqget routine.**

MQPUT

UNIX

Consider the following for the UNIX implementation of the MQPUT Interface:

- It has no restrictions.
- It conforms to the routine requirements for MQSeries MQI mqput.

Mainframe

Consider the following for the Mainframe implementation of the MQPUT Interface:

- **MQMD fields not used by the MQSeries Interface.** For the Mainframe, four MQMD (Message Descriptor) fields do not exist. Although these four fields appear in the MQPUT Interface layout for the Mainframe, the MQSeries Interface does not use them. These fields are the following:
  - MsgSeqNumber
  - MsgFlags
  - OriginalLength
  - GroupId
- **MQPMO fields not used by the MQSeries Interface.** For the Mainframe, two MQPMO (Put Message Options) fields do not exist. Although these two fields appear in the MQPUT Interface layout for the Mainframe, they are not used by the MQSeries Interface:
  - RecsPresent
  - PutMsgRecFields
- **MQPUT conforms to the requirements of the MQSeries MQI mqput routine.**

MQROLLBACK

UNIX

Consider the following for the UNIX implementation of the MQROLLBACK Interface:

- It has no restrictions.
- It conforms to the routine requirements for MQSeries MQI mqback.

Mainframe

Consider the following for the mainframe implementation of the MQROLLBACK Interface:

- MQSeries MQI mqback is not supported in CICS. Using this Interface in CICS will generate a REASON code error of (-29) in the interface.
- To rollback a logical unit of work containing MQSeries updates, use MANTIS RESET. For more information on MANTIS RESET, *refer to MANTIS Language for z/OS and z/VSE*, P39-5302.
- For more information on COMMITting transactions and units of work, refer to *MQSERIES Application Programming Guide*, SC33-0807.

MQCOMMIT

UNIX

Consider the following for the UNIX implementation of the MQCOMMIT Interface:

- It has no restrictions.
- It conforms to the routine requirements for MQSeries MQI mqcmit.
- A CONVERSE on UNIX does not COMMIT the outstanding UNIT of WORK. Therefore, the MANTIS program must handle all COMMITs manually.

Mainframe

Consider the following for the Mainframe implementation of the MQCOMMIT Interface:

- MQSeries MQI mqcmit is not supported in CICS. Using this Interface in CICS will generate a REASON code error of (-22) in the interface.
- To commit a logical unit of work containing MQSeries updates, use MANTIS COMMIT. For more information on MANTIS COMMIT, *refer to MANTIS Language for z/OS and z/VSE*, P39-5302.
- If the MANTIS application is running in Pseudo-Conversational Mode, a CONVERSE, by default, performs the following:
  c. Ends the MANTIS transaction.
  d. Commits the Logical Unit of Work.
- For more information on Committing transactions and units of work, refer to *MQSERIES Application Programming Guide*, SC33-0807.

MQBEGIN

UNIX

Consider the following for the UNIX implementation of the MQCOMMIT Interface:

- It has no restrictions.
- It conforms to the routine requirements for MQSeries MQI mqbegin.

Mainframe

Consider the following for the Mainframe implementation of the MQCOMMIT Interface:

- The MQSeries MQI mqbegin is not supported in any Mainframe environment. Using this Interface on the Mainframe will generate a REASON code error of (-19) in the interface.
- Since, under CICS, all updateable MQSeries transactions are logged by default, an MQBEGIN is not required.
- For more information on COMMITting transactions and units of work, refer to MQSERIES Application Programming Guide, SC33-0807.

MQEXIT

UNIX and Mainframe

Consider the following for the implementation of the MQEXIT Interface:

The MQEXIT Interface is an add-on feature that enables MANTIS programmers to close all open connections with one call. It performs an mqclose and mqdisc on all connection handles.

MQTM

UNIX

Consider the following for the UNIX implementation of the MQTM Interface:

- For triggering, the MQTMC2 record is passed to UNIX MANTIS. Because the MQTM Interface contains all MQTMC2 fields, there are no restrictions.
- The TM_MEMADDR field is only used for Mainframe.

Mainframe

- Consider the following for the Mainframe of the MQTM Interface:
- For triggering, the MQTM record is passed to CICS MANTIS. The MQTM Interface layout contains all MQTMC2 fields. The only field in the Interface layout that is not usable by the Mainframe is TM_QMGRNAME. Although using this field does not generate an error, the MQSeries Interface does not use it. It exists only for compatibility with the UNIX version of MANTIS.
- TM_MEMADDR is a Mainframe-only field. It serves as a storage address of the MQTM record that CSOXTRIG passes to MANTIS. For more information, see 7. MQSeries/MANTIS triggering.

Index

R

T

U

W