

# Cincom

## **AD/ADVANTAGE**

MANTIS WebSphere MQ Programming

P39-1365-00



---

# AD/Advantage<sup>®</sup> MANTIS WebSphere MQ Programming

**Publication Number P39-1365-00**

© 2001 Cincom Systems, Inc.  
All Rights Reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage<sup>®</sup>

C+A-RE<sup>™</sup>

CINCOM<sup>®</sup>

Cincom Encompass<sup>®</sup>

Cincom Smalltalk<sup>™</sup>

Cincom SupportWeb<sup>®</sup>

CINCOM SYSTEMS<sup>®</sup>



gOOj<sup>™</sup>

iD CinDoc<sup>™</sup>

iD CinDoc Web<sup>™</sup>

iD Consulting<sup>™</sup>

iD Correspondence<sup>™</sup>

iD Correspondence Express<sup>™</sup>

iD Environment<sup>™</sup>

iD Solutions<sup>™</sup>

Intelligent Document Solutions<sup>™</sup>

Intermax<sup>™</sup>

MANTIS<sup>®</sup>

Socrates<sup>®</sup>

Socrates<sup>®</sup> XML

SPECTRA<sup>™</sup>

SUPRA<sup>®</sup>

SUPRA<sup>®</sup> Server

Visual Smalltalk<sup>®</sup>

VisualWorks<sup>®</sup>

Acucobol, Inc.

AT&T

Compaq Computer Corporation

Data General Corporation

Gupta Technologies, Inc.

International Business Machines Corporation

JSB Computer Systems Ltd.

Micro Focus, Inc.

Microsoft Corporation

Systems Center, Inc.

TechGnosis International, Inc.

The Open Group

UNIX System Laboratories, Inc.

or of their respective companies.

Cincom Systems, Inc.

55 Merchant Street

Cincinnati, Ohio 45246-3732

U. S. A.

PHONE: (513) 612-2300

FAX: (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.

---

---

## Release information for this manual

*AD/Advantage MANTIS WebSphere MQ Programming*, P39-1365-00, is dated October 30, 2001. This document supports Release 5.5.01 of MANTIS.

### We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. At your convenience, please take the [survey](#) provided with the online documentation.

### Cincom Technical Support for AD/Advantage

*All customers*

Web: <http://supportweb.cincom.com>

*U. S. A. customers*

Phone: 1-800-727-3525

FAX: (513) 612-2000

Attn: AD/Advantage Support

Mail:

Cincom Systems, Inc.

Attn: AD/Advantage Support

55 Merchant Street

Cincinnati, OH 45246-3732

U. S. A.

*Customers outside U. S. A.*

All:

Visit the support links at

<http://www.cincom.com> to find

contact information for your nearest

Customer Service Center.



# Contents

<b>About this book</b>	<b>ix</b>
Using this document .....	ix
MANTIS overview .....	ix
Document organization .....	x
Conventions .....	xi
MANTIS documentation series .....	xiv
MANTIS for Mainframe documentation series .....	xiv
MANTIS for OpenVMS/UNIX documentation series .....	xvi
IBM MQSeries documentation series .....	xvii
Educational material .....	xvii
<b>Overview</b>	<b>19</b>
Introduction to WebSphere MQ Programming .....	19
Description of WebSphere MQ Programming .....	19
Development cycle figure .....	20
Using an Interface layout as a template for your Interface .....	21
Generalized Interface program .....	21
MQI and WebSphere MQ Programming .....	22
Reference materials .....	22
Internal Interfaces for MQSeries support .....	23
<b>Fundamental usage</b>	<b>25</b>
Common fields in MQSeries Interface views .....	25
<b>Field naming conventions</b>	<b>29</b>
Field prefixes .....	29
Introduction .....	29
Adding another level of prefixing .....	30
Different kinds of fields, requiring different actions .....	30

<b>Errors</b>	<b>31</b>
Each possible error condition or warning condition .....	31
<b>Constants</b>	<b>39</b>
Including MQ_INIT in a user program.....	39
Categories of MQSeries constants in MQ_INIT.....	40
<b>Building a MANTIS MQSeries application</b>	<b>43</b>
Introduction .....	43
Creating programs that use CALLS to the MQSeries Interfaces.....	44
Creating a program that reads a message queue .....	44
Creating a program that writes to a message queue.....	45
Initializing Interfaces .....	46
Initializing Interfaces that do not require special initialization.....	46
Initializing Interfaces that require special initialization .....	47
Using the MQSeries Interface layouts .....	48
Introduction .....	48
Using the MQBEGIN Interface to start a unit of work .....	50
Using the MQCOMMIT Interface to establish a sync point and commit all previous message GETs and PUTs .....	52
Using the MQCONNECT Interface to open and connect to an MQSeries object.....	54
Using the MQDISCONNECT Interface to close and disconnect from an MQSeries object .....	58
Using the MQEXIT Interface to close all open handles .....	60
Using the MQGET Interface to read an MQSeries message .....	62
Using the MQPUT Interface to send an MQSeries message.....	66
Using the MQROLLBACK Interface rollback to a previous sync point and reverse all previous message GETs and PUTs.....	70
Using the MQTM Interface to map the MQSeries trigger data to the MANTIS MQTM Interface .....	72
<b>MQSeries/MANTIS triggering</b>	<b>75</b>
General MANTIS trigger considerations .....	76
Procedure for using MANTIS as a trigger handler.....	76
Programs that illustrate the trigger-handling process .....	76
Writing a MANTIS application program to handle the triggered event .....	77
Sample program for sending a message to a trigger queue .....	77
Sample program for handling an MQSeries trigger event .....	77

UNIX MQSeries/MANTIS trigger considerations.....	78
The trigger.sh script as a model for your trigger handler .....	78
Steps required for trigger handling .....	78
Procedure for constructing a trigger handler .....	78
MQSeries and MANTIS procedure for handling the triggered event .....	79
OS/390 MQSeries/MANTIS trigger considerations .....	82
The CSOXTRIG front-end application as a model for your trigger handler ..	82
Steps required for trigger handling .....	82
Procedure for constructing a trigger handler .....	83
MQSeries and MANTIS procedure for handling the triggered event .....	84
<b>MQSeries/MANTIS example programs</b>	<b>87</b>
MQ_INIT .....	88
MQ_SAMPLE .....	88
Uses for MQ_SAMPLE.....	88
Queue used for sending and receiving messages .....	88
UNIX screen shot of MQ_SAMPLE .....	89
OS/390 CICS and COMMIT and ROLLBACK functions .....	89
MQ_HANDLER.....	90
Abilities necessary for any handler to possess .....	90
MEMADDR argument to MQ_HANDLER .....	90
Running MQ_HANDLER interactively vs. running it automatically .....	90
MQ_TRIGGER .....	91
Introduction to MQ_TRIGGER .....	91
MQ_TRIGGER sample output screen.....	91
Defining the trigger and associated initiation queue for the MQ_TRIGGER and the MQ_HANDLER programs .....	92
GETERR(2033) .....	94
<b>MQSeries/MANTIS diagnostic considerations</b>	<b>95</b>
Diagnosing a MANTIS program error.....	95
Dumping MQSeries Interface views .....	97
Introduction to the “DUMP” value .....	97
UNIX sample of a dumped MQCONNECT Interface .....	97
Procedure for dumping the failing Interface layout .....	98
System-specific dump file descriptions .....	99
Dump length .....	100

<b>General UNIX and OS/390 considerations</b>	<b>101</b>
Installation considerations.....	101
UNIX.....	101
OS/390.....	102
MQCONNECT .....	103
UNIX.....	103
OS/390.....	104
MQDISCONNECT .....	105
UNIX.....	105
OS/390.....	105
MQGET .....	106
UNIX.....	106
OS/390.....	106
MQPUT .....	107
UNIX.....	107
OS/390.....	107
MQROLLBACK.....	108
UNIX.....	108
OS/390.....	108
MQCOMMIT.....	109
UNIX.....	109
OS/390.....	109
MQBEGIN.....	110
UNIX.....	110
OS/390.....	110
MQEXIT .....	111
UNIX.....	111
OS/390.....	111
MQTM .....	112
UNIX.....	112
OS/390.....	112

# About this book

---

## Using this document

This guide describes MANTIS WebSphere MQ Programming. This is a MANTIS feature that adds support for the IBM 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.

## MANTIS overview

MANTIS<sup>®</sup> is an application development system that consists of the following:

- ◆ **A programming language.**
- ◆ **Design facilities.** For example:
  - Screens
  - Files

The MANTIS system is designed to increase your productivity in all areas of application development, from initial design through production and maintenance. MANTIS is a part of AD/Advantage, which offers additional tools for application development.

## Document organization

Below is a summary for each chapter in this guide:

### **Chapter 1—Overview**

Briefly describes WebSphere MQ Programming.

### **Chapter 2—Fundamental usage**

Describes MQSeries initial settings and parameters.

### **Chapter 3—Field naming conventions**

Describes MQSeries field prefixes and kinds of MQSeries fields.

### **Chapter 4—Errors**

Describes MANTIS errors.

### **Chapter 5—Constants**

Describes MQSeries's functions in the MQ\_INIT program.

### **Chapter 6—Building a MANTIS MQSeries application**

Describes how you can CALL the MQSeries Interfaces, initialize the MQSeries Interfaces, and use the MQSeries Interfaces.

### **Chapter 7—MQSeries/MANTIS triggering**

Describes how to use MANTIS as an MQSeries trigger handler.

### **Chapter 8—MQSeries/MANTIS example programs**

Describes the MQ\_INIT, MQ\_SAMPLE, MQ\_HANDLER, and MQ\_TRIGGER programs.

### **Chapter 9—MQSeries/MANTIS diagnostic considerations**

Describes how to diagnose a MANTIS program error and dump MQSeries Interface views.

### **Chapter 10—General UNIX and OS/390 considerations**

Describes considerations for installation, MQCONNECT, MQDISCONNECT, MQGET, MQPUT, MQROLLBACK, MQCOMMIT, MQBEGIN, MQEXIT, and MQTM.



Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter FIRST, LAST, or a value for <i>begin</i>.</p>	<pre>{ FIRST   begin   LAST }</pre>
<p><u>Underlining</u> (In syntax)</p>	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not specify ON, OFF, or a row and column destination, the system defaults to ON.</p>	<pre>SCROLL [ ON         OFF         [row][,col] ]</pre>
	<p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either PRO or PROTECTED.</p>	<pre>PROTECTED</pre>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.</p>	<pre>(argument, ...)</pre>

Convention	Description	Example
UPPERCASE	<p>Indicates MANTIS reserved words. You must enter them exactly as they appear.</p> <p>The example indicates that you must enter CONVERSE exactly as it appears.</p>	CONVERSE <i>name</i>
<i>Italics</i>	<p>Indicate variables you replace with a value, a column name, a file name, and so on.</p> <p>The example indicates that you can supply a name for the program.</p>	COMPOSE [ <i>program-name</i> ]
Punctuation marks	<p>Indicate required syntax that you must code exactly as presented.</p> <p>( ) parentheses  . period  , comma  : colon  ; semicolon  ' single quotation mark  " " double quotation marks</p>	[LET] <sub>v</sub> [ <sup>(i)</sup> <sub>(i, j)</sub> ] [ROUNDED( <i>n</i> )] = <i>e1</i> [, <i>e2</i> , <i>e3</i> ...]

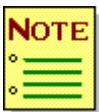
## MANTIS documentation series

### MANTIS for Mainframe documentation series

The MANTIS documentation series contains the following guides:

#### MASTER User tasks

- ◆ *MANTIS Installation, Startup, and Configuration, MVS/ESA, OS/390, P39-5018*
- ◆ *MANTIS Installation, Startup, and Configuration, VSE/ESA, P39-5019*
- ◆ *MANTIS Administration, OS/390, VSE/ESA, P39-5005*
- ◆ *MANTIS Messages and Codes, OS/390, VSE/ESA, P39-5004\**
- ◆ *MANTIS Administration Tutorial, OS/390, VSE/ESA, P39-5027*
- ◆ *MANTIS XREF Administration, OS/390, VSE/ESA, P39-0012*



Manuals marked with an asterisk (\*) are listed twice because you can use them for multiple tasks.

---

## General use

- ◆ *MANTIS Quick Reference, OS/390, VSE/ESA*, P39-5003
- ◆ *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001
- ◆ *MANTIS Language, OS/390, VSE/ESA*, P39-5002
- ◆ *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013
- ◆ *MANTIS Messages and Codes, OS/390, VSE/ESA*, P39-5004\*
- ◆ *AD/Advantage Programming*, P39-7001
- ◆ *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028
- ◆ *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105
- ◆ *MANTIS XREF, OS/390, VSE/ESA, OpenVMS*, P39-0011
- ◆ *MANTIS Entity Transformers*, P39-0013
- ◆ *MANTIS DL/I Programming, OS/390, VSE/ESA*, P39-5008
- ◆ *MANTIS SAP Facility, OS/390, VSE/ESA*, P39-7000
- ◆ *MANTIS WebSphere MQ Programming*, P39-1365
- ◆ *MANTIS Application Development Tutorial, OS/390, VSE/ESA*, P39-5026



---

Manuals marked with an asterisk (\*) are listed twice because you can use them for multiple tasks.

---

## MANTIS for OpenVMS/UNIX documentation series

The MANTIS documentation series contains the following guides:

### Getting started

- ◆ *MANTIS 2.8.01 Installation and Startup, OpenVMS/UNIX, P39-0027\**

### General use

- ◆ *MANTIS Facilities, OpenVMS/UNIX, P39-1300\**
- ◆ *MANTIS Language, OpenVMS/UNIX, P39-1310*
- ◆ *MANTIS Messages and Codes, OpenVMS/UNIX, P39-1330*
- ◆ *MANTIS Application Development Tutorial, OpenVMS/UNIX, P39-1340*
- ◆ *MANTIS SUPRA SQL Programming, OpenVMS/UNIX, P39-1345*
- ◆ *AD/Advantage Programming, P39-7001*
- ◆ *AD/Advantage MANTIS Entity Transformers, P39-0013*
- ◆ *AD/Advantage Component Management Facility, P19-2131*
- ◆ *MANTIS Oracle Programming, UNIX, P39-1355*
- ◆ *MANTIS DB2 Programming, UNIX, P39-1360*
- ◆ *MANTIS WebSphere MQ Programming, P39-1365*

### Master User tasks

- ◆ *MANTIS Facilities, OpenVMS/UNIX, P39-1300\**
- ◆ *MANTIS Administration, OpenVMS/UNIX, P39-1320*
- ◆ *MANTIS 2.8.01 Installation and Startup, OpenVMS/UNIX, P39-0027\**



---

Manuals marked with an asterisk (\*) are listed twice because you can use them for multiple tasks.

---

## **IBM MQSeries documentation series**

MANTIS programmers who are unfamiliar with the MQI interface should consult the following IBM MQSeries manuals:

- ◆ *MQSeries Application Programming Reference*, SC33-1673
- ◆ *MQSeries Application Programming Guide*, SC33-0807
- ◆ *MQSeries System Administration*, SC33-1873
- ◆ *MQSeries, An Introduction to Messaging and Queuing*, GC33-0805
- ◆ *MQSeries Messages*, GC33-1876

## **Educational material**

MANTIS educational material is available from your regional Cincom education department.



# 1

## Overview

---

### Introduction to WebSphere MQ Programming

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

#### Description of WebSphere MQ Programming

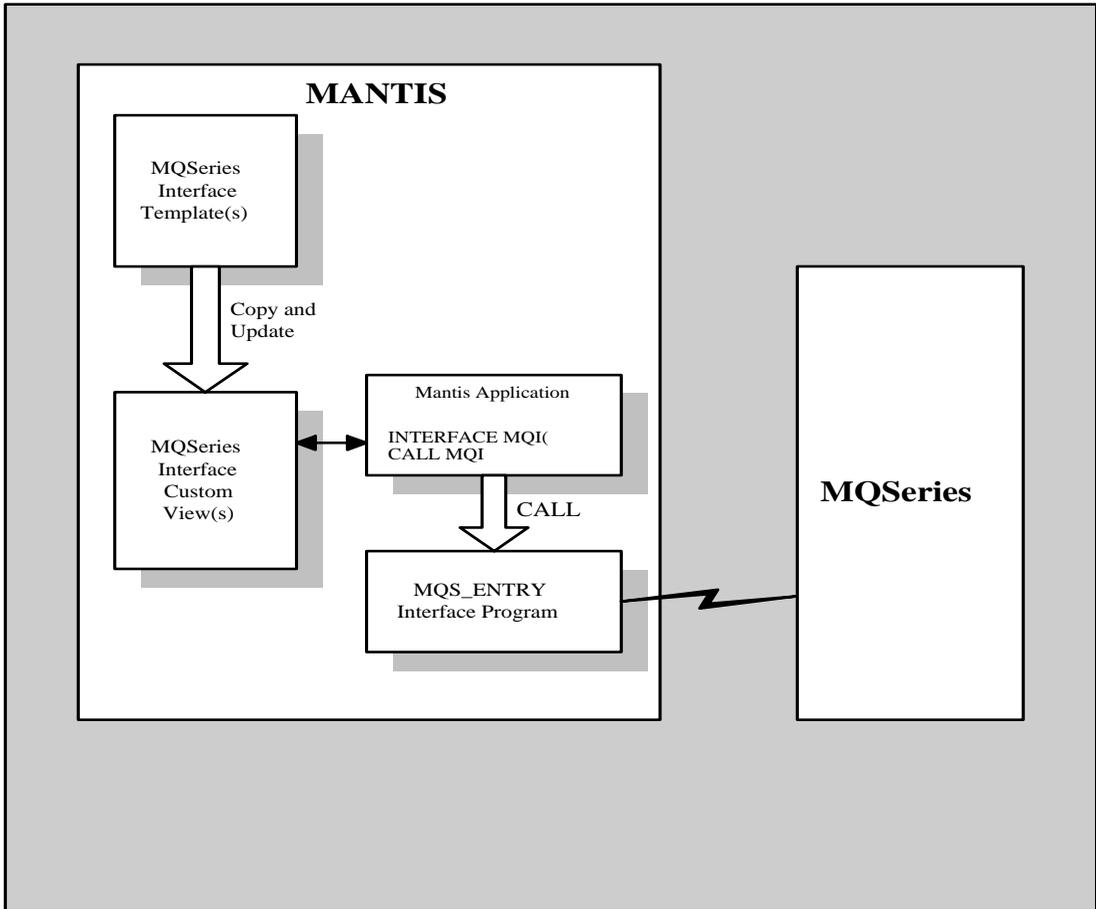
WebSphere MQ Programming is a MANTIS feature that adds support for IBM's MQSeries messaging product. This feature enables MANTIS application programmers to send MQSeries messages to, and receive MQSeries messages from, any local or remote machine that supports MQSeries.

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. This program is named differently under mainframe CICS and UNIX:

- ◆ **Mainframe CICS.** A CICS transaction called CSOXWMQS.
- ◆ **UNIX.** A shared library called libmq5.so. It has an entry point called MQS\_Entry.

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, OpenVMS/UNIX*, P39-1300.
  - If you use OS/390 or VSE/ESA—*MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.
- ◆ For information on the CALL and INTERFACE statements:
  - If you use OpenVMS or UNIX—*MANTIS Language OpenVMS/UNIX*, P39-1310.
  - If you use OS/390 or VSE/ESA—*MANTIS Language, OS/390, VSE/ESA*, P39-5002.

## 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.	<ul style="list-style-type: none"> <li>◆ Optional.</li> <li>◆ Unix only.</li> </ul>
MQCOMMIT	Establishes a sync point and ends a unit of work.	<ul style="list-style-type: none"> <li>◆ Optional.</li> <li>◆ Unix only.</li> </ul>
MQCONNECT	Opens an MQSeries object and connects to it.	<ul style="list-style-type: none"> <li>◆ This interface requires special initialization, performed with the "INITCONN" function.</li> <li>◆ Creates a handle that all other Interfaces use.</li> </ul>
MQDISCONNECT	Closes an MQSeries object and disconnects from it.	
MQEXIT	Closes all open handles.	<ul style="list-style-type: none"> <li>◆ Optional.</li> <li>◆ Unix Only.</li> <li>◆ Only this Interface does not correspond to an MQSeries function; this Interface is an extra Cincom feature.</li> </ul>

Interface	Description	Comments
MQGET	Reads an MQSeries message.	<ul style="list-style-type: none"> <li>◆ The user should perform the following:               <ol style="list-style-type: none"> <li>1. Modify the Interface.</li> <li>2. Save the Interface under a different name.</li> </ol> </li> <li>◆ This interface requires special initialization, performed with the "INITGET" function.</li> </ul>
MQPUT	Sends an MQSeries message.	<ul style="list-style-type: none"> <li>◆ The user should perform the following:               <ol style="list-style-type: none"> <li>1. Modify the Interface.</li> <li>2. Save the Interface under a different name.</li> </ol> </li> <li>◆ This interface requires special initialization, performed with the "INITPUT" function.</li> </ul>
MQROLLBACK	Reverse GETs and PUTs back to a prior sync point or a unit of work begin.	<ul style="list-style-type: none"> <li>◆ Optional.</li> <li>◆ Unix only.</li> </ul>
MQTM	Retrieves either the MVS MQTM messages or the UNIX MQTMC2 Trigger message.	Optional.

# 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 ...)
```

---

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"
- ◆ "PUT"
- ◆ "BEGIN"\*†
- ◆ "COMMIT"\*†
- ◆ "CONNECT"
- ◆ "DISCONN"
- ◆ "ROLLBACK"\*†
- ◆ "DUMP"
- ◆ "EXIT"\*†
- ◆ "INITPUT"
- ◆ "INITGET"
- ◆ "INITCONN"
- ◆ "INITMQTM"\*†

\* Unix

† OS/390 CICS

**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 32-byte TEXT field.

**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. For UNIX users, another disconnection method is to call the MQEXIT routine that disconnects the application from all connected objects.
- ◆ 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.
- 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 [“MQSeries/MANTIS diagnostic considerations”](#) on page 95.

# 3

## Field naming conventions

### Field prefixes

#### Introduction

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 and DMPLength 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

## 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 Manual*, SC33-1673.

# 4

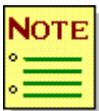
## Errors

After receiving return from all Interface calls, check the INTERFACE symbolic variable for error conditions or warning conditions.

### Each possible error condition or warning condition

The following table lists each possible error condition or warning condition (these are named in the “Symbolic name value” column). For each possible error condition or warning 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 further information on each error condition or warning condition in the following table, refer to *MQSERIES Application Programming Reference Manual*, SC33-1673.

Symbolic name value	Interface name	Situation	Recommended response
""	All	Function successfully completes.	None required.
BGNERR (UNIX only.)	MQBEGIN	An error occurs during the mqbegin API call in the MQBEGIN routine.	Compare the value of the REASON field in the view to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
BGNWRN (UNIX only.)	MQBEGIN	A warning occurs during the mqbegin API call in the MQBEGIN routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
CLOERR	MQDISCONNECT	An error occurs during the mqclose API call in the MQDISCONNECT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
CMTERR (UNIX only.)	MQCOMMIT	An error occurs during the mqcmit API call in the MQCOMMIT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
CMTWRN (UNIX only.)	MQCOMMIT	A warning occurs during the mqcmit API call in the MQCOMMIT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.

Symbolic name value	Interface name	Situation	Recommended response
CONERR	MQCONNECT	An error occurs during the mqconn API call in the MQCONNECT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
DMPERR	MQDUMP	An MQSeries Interface layout is dumped. Dump I/O may have failed because of one of the following: <ul style="list-style-type: none"> <li>◆ Output file could not be opened.</li> <li>◆ Data could not be written out.</li> <li>◆ The file was closed incorrectly.</li> </ul>	To find out what you should do, see “MQSeries/MANTIS diagnostic considerations” on page 95.
DMPINVID	MQDUMP	An MQSeries Interface layout is dumped. You did not set the Interface type to a valid type.	To find out what you should do, see “MQSeries/MANTIS diagnostic considerations” on page 95.
DMPINVSZ	MQDUMP	An MQSeries Interface layout is dumped. The Interface type was not identified and the DMPLENGTH field was set to ZERO.	To find out what you should do, see “MQSeries/MANTIS diagnostic considerations” on page 95.

<b>Symbolic name value</b>	<b>Interface name</b>	<b>Situation</b>	<b>Recommended response</b>
DSCERR	MQDISCONNECT	An error occurs during the mqdisc API call in the MQDISCONNECT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
DSCWRN	MQDISCONNECT	A warning occurs during the mqdisc API call in the MQDISCONNECT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
GETERR	MQGET	An error occurs during the mqget API call in the MQGET routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
GETWRN	MQGET	A warning occurs during the mqget API call in the MQGET routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.

Symbolic name value	Interface name	Situation	Response
INVHAN	All	<p>The Interface layout supplies an invalid value of ZERO in the HANDLE field.</p>	<p>Perform the following:</p> <ol style="list-style-type: none"> <li>1. Correct the HANDLE so that it matches one of the values returned by the MQCONNECT call.</li> <li>2. Check the spelling of the HANDLE variable.</li> <li>3. Ensure that you have assigned the HANDLE variable a valid handle value.</li> </ol>
INVHANDL	All	<p>The Interface layout supplies an invalid NONZERO value in the HANDLE field. This may be the result of one of the following:</p> <ul style="list-style-type: none"> <li>◆ A handle was closed and disconnected.</li> <li>◆ The program modified the variable holding the handle.</li> </ul>	<p>Correct the HANDLE so that it matches one of the values returned by an MQCONNECT call.</p>

Symbolic name value	Interface name	Situation	Recommended response
INVREQ	All	The Interface layout supplies an invalid value in the FUNCTION field.	<p>Correct the FUNCTION field so that it contains one of the following values:</p> <ul style="list-style-type: none"> <li>◆ GET</li> <li>◆ PUT</li> <li>◆ BEGIN</li> <li>◆ COMMIT</li> <li>◆ CONNECT</li> <li>◆ DISCONN</li> <li>◆ ROLLBACK</li> <li>◆ DUMP</li> <li>◆ EXIT</li> <li>◆ INITPUT</li> <li>◆ INITGET</li> <li>◆ INITCONN</li> <li>◆ INITMQTM</li> </ul>
MEMFAIL1	MQCONNECT	Memory chains within the MANTIS process are corrupt during the MQCONNECT routine call to acquire memory for internal purposes.	(This error should never occur but has been documented for completeness.) Document the problem and contact Cincom's technical support.
MEMFAIL2	MQCONNECT	Memory chains within the MANTIS process are corrupt during the MQCONNECT routine call to acquire memory for internal purposes.	(This error should never occur but has been documented for completeness.) Document the problem and contact Cincom's technical support.

Symbolic name value	Interface name	Situation	Recommended response
MQTMCERR (UNIX only.)	MQTM	The call to the MQTM Interface is made, but the MQTMC2 environment variable is not 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. For more information, see <a href="#">“MQSeries/MANTIS triggering”</a> on page 75.
MQTMERR (OS/390 only.)	MQTM	The call to the MQTM Interface is made, but the TM_MEMADDR field in the Interface layout is not set to a valid memory address containing the MQSeries MQTM record.	To find out what you should do, see <a href="#">“MQSeries/MANTIS triggering”</a> on page 75.
NOTAUTH (OS/390 only.)	All	The MANTIS program attempts to access the MQSeries Interface, but has not been authorized to do so.	Verify that the password on the INTERFACE statement matches the password in the Interface design. For more information, contact your Master User.
OPNERR	MQCONNECT	An error occurs during the mqopen API call in the MQCONNECT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.

Symbolic name value	Interface name	Situation	Recommended response
PUTERR	MQPUT	An error occurs during the mqput API call in the MQPUT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
PUTWRN	MQPUT	A warning occurs during the mqput API call in the MQPUT routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.
RBKERR (UNIX only.)	MQROLLBACK	An error occurs during the mqback API call in the MQROLLBACK routine.	Compare the value of the REASON field in the Interface layout to the value listed in <i>MQSERIES Application Programming Reference Manual</i> , SC33-1673.

# 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.

<b>Category of MQSeries constants</b>	<b>Description</b>
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: <ul style="list-style-type: none"><li>◆ MQGMO Get Message Options</li><li>◆ Match Options</li><li>◆ Group Status</li><li>◆ Segment Status</li><li>◆ Segmentation and Expiry</li></ul>

Category of MQSeries constants	Description
MQMD Constants for PUT	Constants that can be used in the MQPUT Interface and consist of the following: <ul style="list-style-type: none"> <li>◆ Structure ID</li> <li>◆ Version number</li> <li>◆ Report Options</li> <li>◆ Message Type</li> <li>◆ Feedback Values</li> <li>◆ Encoding Values</li> <li>◆ Coded Character</li> <li>◆ Set Identifiers</li> <li>◆ Format Values</li> <li>◆ Priority Values</li> <li>◆ Persistence Values</li> <li>◆ Message Identifier Values</li> <li>◆ Message Correlation Identifier</li> <li>◆ Put Application Types</li> <li>◆ Put Message Flags</li> </ul>
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: <ul style="list-style-type: none"> <li>◆ MQPMO Version Number</li> <li>◆ MQPMO Structure Length</li> <li>◆ MQPMO Put-Message Options</li> <li>◆ MQPMO Message Record Fields</li> </ul>
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

### Introduction

This chapter describes how to perform the following:

- ◆ **Create programs that use CALLS to the MQSeries Interfaces.** See “[Creating programs that use CALLS to the MQSeries Interfaces](#)” on page 44.
- ◆ **Initialize the MQSeries Interfaces.** See “[Initializing Interfaces](#)” on page 46.
- ◆ **Use the MQSeries Interfaces.** See “[Using the MQSeries Interface layouts](#)” on page 48.

## Creating programs that use CALLS to 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	-----
Loop	-----
MQPUT	-----
MQCOMMIT or MQROLLBACK	-----
Endloop	-----
MQDISCONNECT or MQEXIT	-----

## Initializing Interfaces

### 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](#)” on page 55.
- ◆ **MQGET.** Call the INITGET function to initialize this Interface layout. See “[Initializing the MQGET Interface to its default usable state](#)” on page 64.
- ◆ **MQPUT.** Call the INITPUT function to initialize this Interface layout. See “[Initializing the MQPUT Interface to its default usable state](#)” on page 68.




---

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

### Introduction

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 programs that use CALLS to the MQSeries Interfaces](#)” on page 44.

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. <b>Note:</b> 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

Interface name	Description	Used as-is?	Requires special initialization?	Comments
MQGET	Reads an MQSeries message.	No	Yes	<p>To modify this Interface, perform the following:</p> <ol style="list-style-type: none"> <li>1. Copy this Interface.</li> <li>2. Add your message data layout to the end of the copy.</li> </ol> <p><b>Note:</b> Requires special initialization prior to its use ("INITGET").</p>
MQPUT	Sends an MQSeries message.	No	Yes	<p>To modify this Interface, perform the following:</p> <ol style="list-style-type: none"> <li>1. Copy this Interface.</li> <li>2. Add your message data layout to the end of the copy.</li> </ol> <p><b>Note:</b> Requires special initialization prior to its use ("INITPUT").</p>
MQROLLBACK	Resets rollback to prior synch point.	Yes	No	N/A
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 : 6		Element Size : 56						
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	BEG_FUNCTION	TEXT	TEXT	8				
2	BEG_HANDLE	TEXT	TEXT	32				
3	BEG_COMPCODE	BIG	BINARY	4				
4	BEG_REASON	BIG	BINARY	4				
5	BEG_IDENT	BIG	BINARY	4				
6	BEG_DMPLLENGTH	BIG	BINARY	4				

## 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".

**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:

1. Loads the MQBEGIN Interface.
2. Sets the required fields (`BEG_FUNCTION` and `BEG_HANDLE`).
3. 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 OS/390 Transaction Server

MQBEGIN is not supported in OS/390 Transaction Server. In OS/390 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 mqcmmit 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 : 6		Element Size : 56						
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	COM_FUNCTION	TEXT	TEXT	8				
2	COM_HANDLE	TEXT	TEXT	32				
3	COM_COMPCODE	BIG	BINARY	4				
4	COM_REASON	BIG	BINARY	4				
5	COM_IDENT	BIG	BINARY	4				
6	COM_DMPLLENGTH	BIG	BINARY	4				
4								

## 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".
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:

1. Loads the MQCOMMIT Interface.
2. Sets the required fields (COM\_FUNCTION and COM\_HANDLE).
3. 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 OS/390 Transaction Server

MQCOMMIT is not supported in OS/390 Transaction Server. In OS/390 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.

### 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:

ELEM	NAME	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	CON_FUNCTION	TEXT	TEXT	8				
2	CON_HANDLE	TEXT	TEXT	32				
3	CON_COMPCODE	BIG	BINARY	4				
4	CON_REASON	BIG	BINARY	4				
5	CON_IDENT	BIG	BINARY	4				
6	CON_DMPLLENGTH	BIG	BINARY	4				
7	CON_QMGRNAME	TEXT	TEXT	48				
8	CON_OPTIONS	BIG	BINARY	4				
9	MQOD_OBJECTTYPE	BIG	BINARY	4				
10	MQOD_OBJECTNAME	TEXT	TEXT	48				
11	MQOD_OBJECTQMGRNAME	TEXT	TEXT	48				
12	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:

- ◆ QMGRNAME
- ◆ OPTIONS
- ◆ OBJECTTYPE
- ◆ OBJECTNAME
- ◆ OBJECTQMGRNAME
- ◆ ALTERNATEUSERID

## 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 Manual*, 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."
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 in another TEXT variable for a length of 32 (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 TEXT SAVE_HANDLE1( 32)
30 TEXT SAVE_HANDLE2( 32)
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.

### 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 : 7		Element Size : 60						
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	DIS_FUNCTION	TEXT	TEXT	8				
2	DIS_HANDLE	TEXT	TEXT	32				
3	DIS_COMPCODE	BIG	BINARY	4				
4	DIS_REASON	BIG	BINARY	4				
5	DIS_IDENT	BIG	BINARY	4				
6	DIS_DMPLLENGTH	BIG	BINARY	4				
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."
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 : 6		Element Size : 56						
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	EXT_FUNCTION	TEXT	TEXT	8				
2	EXT_HANDLE	TEXT	TEXT	32				
3	EXT_COMPCODE	BIG	BINARY	4				
4	EXT_REASON	BIG	BINARY	4				
5	EXT_IDENT	BIG	BINARY	4				
6	EXT_DMPLLENGTH	BIG	BINARY	4				

## 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".



---

During MANTIS fatal processing, 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 the MQPUT 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 : 45						Element Size : 504		
ELEM	NAME	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	GET_FUNCTION	TEXT	TEXT	8				
2	GET_HANDLE	TEXT	TEXT	32				
3	GET_COMPCODE	BIG	BINARY	4				
4	GET_REASON	BIG	BINARY	4				
5	GET_IDENT	BIG	BINARY	4				
6	GET_DMPLLENGTH	BIG	BINARY	4				
7	GET_MSGLLENGTH	BIG	BINARY	4				
8	G_MQMD_VERSION	BIG	BINARY	4	YES			
9	G_MQMD_REPORT	BIG	BINARY	4	YES			
10	G_MQMD_MSGTYPE	BIG	BINARY	4	YES			
11	G_MQMD_EXPIRY	BIG	BINARY	4	YES			
12	G_MQMD_FEEDBACK	BIG	BINARY	4	YES			
13	G_MQMD_ENCODING	BIG	BINARY	4	YES			
14	G_MQMD_CODEDCHAR	BIG	BINARY	4	YES			
15	G_MQMD_FORMAT	TEXT	TEXT	8				
16	G_MQMD_PRIORITY	BIG	BINARY	4	YES			
17	G_MQMD_PERSISTEN	BIG	BINARY	4				
18	G_MQMD_MSGID	TEXT	TEXT	24				
19	G_MQMD_CORRELID	TEXT	TEXT	24				
20	G_MQMD_BACKOUTCO	BIG	BINARY	4	YES			
21	G_MQMD_REPLYTOQ	TEXT	TEXT	48				
22	G_MQMD_REPLYTOQM	TEXT	TEXT	48				
23	G_MQMD_USERIDENT	TEXT	TEXT	12				
24	G_MQMD_ACCOUNTIN	TEXT	TEXT	32				
25	G_MQMD_APPLIDENT	TEXT	TEXT	32				
26	G_MQMD_PUTAPPLTY	BIG	BINARY	4	YES			
27	G_MQMD_PUTAPPLNA	TEXT	TEXT	28				
28	G_MQMD_PUTDATE	TEXT	TEXT	8				
29	G_MQMD_PUTTIME	TEXT	TEXT	8				
30	G_MQMD_APPLORIGI	TEXT	TEXT	4				
31	G_MQMD_GROUPID	TEXT	TEXT	24				
32	G_MQMD_MSGSEQNUM	BIG	BINARY	4	YES			
33	G_MQMD_MSGFLAGS	BIG	BINARY	4	YES			
34	G_MQMD_ORIGINAL	BIG	BINARY	4	YES			
35	MQGMO_VERSION	BIG	BINARY	4	YES			
36	MQGMO_OPTIONS	BIG	BINARY	4	YES			
37	MQGMO_WAITINTERV	BIG	BINARY	4	YES			
38	MQGMO_RESOLVEDQN	TEXT	TEXT	48				
39	MQGMO_MATCHOPTIO	BIG	BINARY	4	YES			
40	MQGMO_GROUPSTATU	TEXT	TEXT	1				
41	MQGMO_SEGMENTSTA	TEXT	TEXT	1				
42	MQGMO_SEGMENTATI	TEXT	TEXT	1				
43	MQGMO_RESERVED1	TEXT	TEXT	1				
44	MQGMO_MSGTOKEN	TEXT	TEXT	16				
45	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 Manual*, 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".
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 : 43						Element Size : 540		
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	PUT_FUNCTION	TEXT	TEXT	8				
2	PUT_HANDLE	TEXT	TEXT	32				
3	PUT_COMPCODE	BIG	BINARY	4				
4	PUT_REASON	BIG	BINARY	4				
5	PUT_IDENT	BIG	BINARY	4				
6	PUT_DMPLLENGTH	BIG	BINARY	4				
7	PUT_LENGTH	BIG	BINARY	4				
8	P_MQMD_VERSION	BIG	BINARY	4	YES			
9	P_MQMD_REPORT	BIG	BINARY	4	YES			
10	P_MQMD_MSGTYPE	BIG	BINARY	4	YES			
11	P_MQMD_EXPIRY	BIG	BINARY	4	YES			
12	P_MQMD_FEEDBACK	BIG	BINARY	4	YES			
13	P_MQMD_ENCODING	BIG	BINARY	4	YES			
14	P_MQMD_CODEDCHAR	BIG	BINARY	4	YES			
15	P_MQMD_FORMAT	TEXT	TEXT	8				
16	P_MQMD_PRIORITY	BIG	BINARY	4	YES			
17	P_MQMD_PERSISTEN	BIG	BINARY	4	YES			
18	P_MQMD_MSGID	TEXT	TEXT	24				
19	P_MQMD_CORRELID	TEXT	TEXT	24				
20	P_MQMD_BACKOUTCO	BIG	BINARY	4	YES			
21	P_MQMD_REPLYTOQ	TEXT	TEXT	48				
22	P_MQMD_REPLYTOQM	TEXT	TEXT	48				
23	P_MQMD_USERIDENT	TEXT	TEXT	12				
24	P_MQMD_ACCOUNTIN	TEXT	TEXT	32				
25	P_MQMD_APPLIDENT	TEXT	TEXT	32				
26	P_MQMD_PUTAPPLTY	BIG	BINARY	4	YES			
27	P_MQMD_PUTAPPLNA	TEXT	TEXT	28				
28	P_MQMD_PUTDATE	TEXT	TEXT	8				
29	P_MQMD_PUTTIME	TEXT	TEXT	8				
30	P_MQMD_APPLORIGI	TEXT	TEXT	4				
31	P_MQMD_GROUPID	TEXT	TEXT	24				
32	P_MQMD_MSGSEQNUM	BIG	BINARY	4	YES			
33	P_MQMD_MSGFLAGS	BIG	BINARY	4	YES			
34	P_MQMD_ORIGINAL	BIG	BINARY	4	YES			
35	MQPMO_VERSION	BIG	BINARY	4	YES			
36	MQPMO_OPTIONS	BIG	BINARY	4	YES			
37	MQPMO_KNOWNDESTC	BIG	BINARY	4	YES			
38	MQPMO_UNKNOWNDES	BIG	BINARY	4	YES			
39	MQPMO_INVALIDDES	BIG	BINARY	4	YES			
40	MQPMO_RESOLVEDQN	TEXT	TEXT	48				
41	MQPMO_RESOLVEDQM	TEXT	TEXT	48				
42	MQPMO_RECSPRESEN	BIG	BINARY	4	YES			
43	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 Manual*, 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".
2. Set the PUT\_HANDLE variable to a valid handle from a previous MQCONNECT.
3. Call the Interface.

**Sample code.** See the following sample code:

```
170 PUT_FUNCTION="PUT"  
180 PUT_HANDLE=SAVE_HANDLE2  
190 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. Calls the MQGET1 Interface.

## Using the MQROLLBACK Interface 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 : 6	Element Size : 56							
ELEM	-----NAME-----	TYPE	FORMAT	LEN	SIGN	DEC	DIM	-ATTRIBUTE-
1	ROL_FUNCTION	TEXT	TEXT	8				
2	ROL_HANDLE	TEXT	TEXT	32				
3	ROL_COMPCODE	BIG	BINARY	4				
4	ROL_REASON	BIG	BINARY	4				
5	ROL_IDENT	BIG	BINARY	4				
6	ROL_DMPLNGTH	BIG	BINARY	4				

## 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".
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.

## OS/390 Transaction Server and MQROLLBACK

MQROLLBACK is not supported in OS/390 Transaction Server. In OS/390 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 “MQSeries/MANTIS triggering” on page 75 and “General UNIX and OS/390 considerations” on page 101.

### 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:

ELEM	NAME	TYPE	FORMAT	LEN	SIGN	DEC	DIM	ATTRIBUTE
1	TM_FUNCTION	TEXT	TEXT	8				
2	TM_HANDLE	TEXT	TEXT	32				
3	TM_COMPCODE	BIG	BINARY	4				
4	TM_REASON	BIG	BINARY	4				
5	TM_IDENT	BIG	BINARY	4				
6	TM_DMPLLENGTH	BIG	BINARY	4				
7	TM_MEMADDR	TEXT	TEXT	4				
8	TM_STRUCID	TEXT	TEXT	4				
9	TM_VERSION	BIG	BINARY	4				
10	TM_QNAME	TEXT	TEXT	48				
11	TM_PROCESSNAME	TEXT	TEXT	48				
12	TM_TRIGGERDATA	TEXT	TEXT	64				
13	TM_APPLTYPE	BIG	BINARY	4				
14	TM_APPLID1	TEXT	TEXT	128				
15	TM_APPLID2	TEXT	TEXT	128				
16	TM_ENVDATA	TEXT	TEXT	128				
17	TM_USERDATA	TEXT	TEXT	128				
18	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:

1. Loads the MQTM Interface.
2. Sets the required field (TMC\_FUNCTION).
3. 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” on page 76.
- ◆ “UNIX MQSeries/MANTIS trigger considerations” on page 78.
- ◆ “OS/390 MQSeries/MANTIS trigger considerations” on page 82.



---

Using MANTIS for triggering is not currently supported under OS/390 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.

### Procedure for using MANTIS as a trigger handler

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

1. 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”](#) on page 78 or [“OS/390 MQSeries/MANTIS trigger considerations”](#) on page 82.
2. 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”](#) on page 78
  - ◆ [“OS/390 MQSeries/MANTIS trigger considerations”](#) on page 82

### Programs that illustrate the trigger-handling process

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

See [“MQSeries/MANTIS example programs”](#) on page 87 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 OS/390 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 contain detailed information about the message that caused the trigger event.

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

1. Interrogate the fields.
2. Open the appropriate queue.
3. 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](#)” on page 91).

## 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](#)” on page 90).

## 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/libmq directory that accompanies the UNIX version of MANTIS.

### Steps required for trigger handling

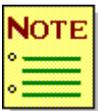
For trigger handling, the following must occur:

1. You must associate trigger.sh, or a program that you have modeled on it, with the trigger queue.
2. MQSeries must execute trigger.sh, or a program that you have modeled on 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):

1. Redirect all MANTIS terminal output to a log file.  
  
Do this because MANTIS executes the user, password, and program in batch form, in the background.
2. Enter settings for the MANTIS working environment, such as MANTIS\_PATCH, MANTIS\_ROOT, MANTIS\_CLASS, and so on.
3. Set the MQTMC2 environment variable to the trigger record passed from MQSeries.
4. 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.

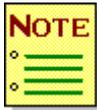
---

## 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](#)” on page 81) to cooperatively handle the triggered event:

1. 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.

---

2. MQSeries copies the message to the initiation queue.
3. 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).
4. `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.

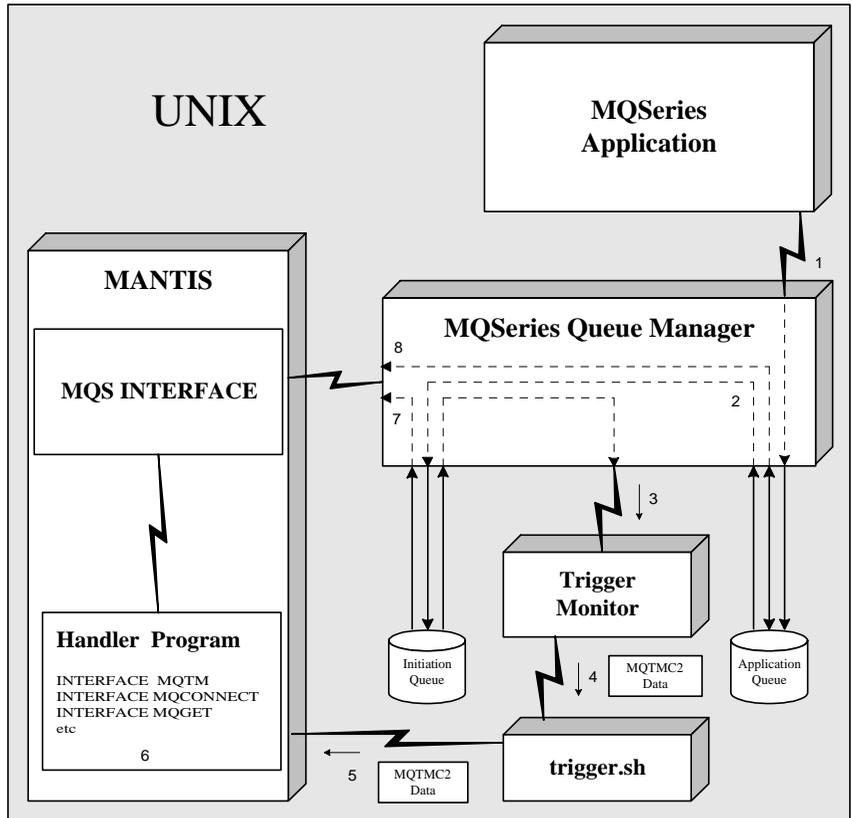
5. MANTIS signs on via the specified USER and PASSWORD.
6. The MANTIS program specified by trigger.sh in step 4 begins executing.

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 .

7. 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.
8. 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” on page 79) to cooperatively handle the triggered event:



## OS/390 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:

1. You must associate CSOXTRIG, or a program modeled on it, with the trigger queue.
2. MQSeries must execute CSOXTRIG, or a program modeled on it, as the trigger handler.

## Procedure for constructing a trigger handler

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

1. Customize the BTRANID setting.

This is the ID of the MANTIS background transaction that this front-end application will start.

2. Customize the BUID setting.

This is the user that is executed in batch MANTIS.

3. Customize the BPSW setting.

This is the password that is executed in batch MANTIS.

4. 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.

---

5. 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](#)” on page 86) to cooperatively handle the triggered event:

1. 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.

---

2. MQSeries copies the message to the initiation queue.
3. 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).
4. 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.

5. MANTIS, running as a background MANTIS task, signs on via the specified USER and PASSWORD.

6. 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.

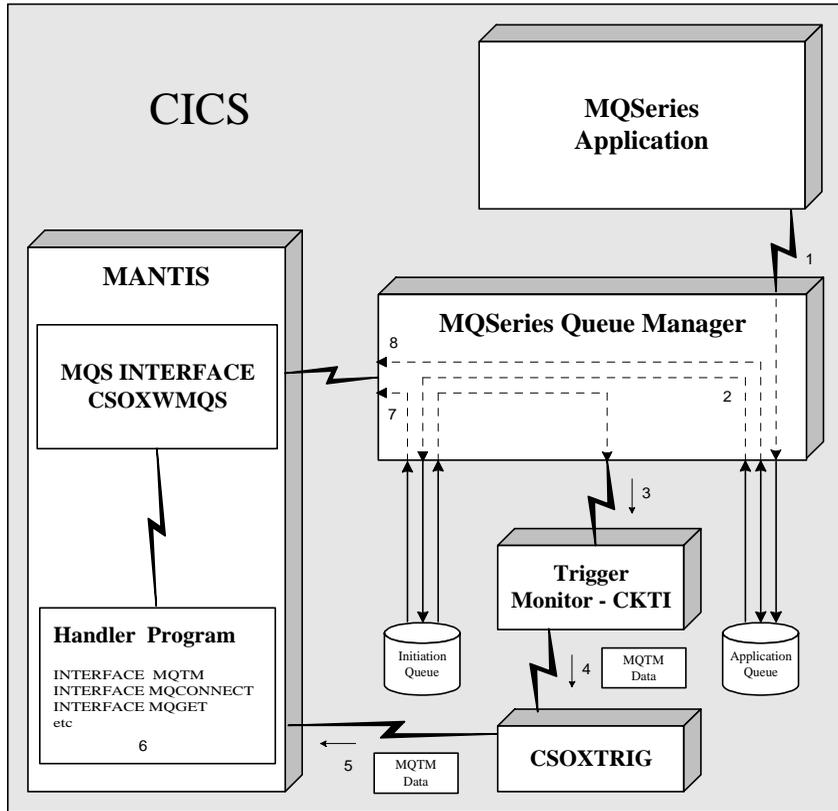
7. The MANTIS program performs the following:

A. Connects to the initiation queue returned in the MQTM Interface.

B. Retrieves the message that caused the trigger event.

8. 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” on page 84) 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.

<b>MANTIS program</b>	<b>Associated screen</b>	<b>Description</b>
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 “Constants” on page 39.

---

## MQ\_SAMPLE

MQ\_SAMPLE is an example program that performs the following:

1. Tests each MQSeries Interface.
2. 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.

```
MANTIS / MQSERIES TEST SCREEN

START TIME : 10:33:58 :           END TIME : 10:34:01 :

CONNECT 1 .....: SUCCESSFUL           :
CONNECT 1 DUMP .....: SUCCESSFUL           :
CONNECT 2 .....: SUCCESSFUL           :
CONNECT 2 DUMP .....: SUCCESSFUL           :
PUT .....: SUCCESSFUL           :
PUT DUMP .....: SUCCESSFUL           :
GET .....: SUCCESSFUL           :
GET DUMP .....: SUCCESSFUL           :
COMMIT .....: SUCCESSFUL           :
COMMIT DUMP .....: SUCCESSFUL           :
ROLLBACK .....: SUCCESSFUL           :
ROLLBACK DUMP .....: SUCCESSFUL           :
DISCONNECT 1 .....: SUCCESSFUL           :
DISCONNECT 1 DUMP ..: SUCCESSFUL           :
DISCONNECT 2 .....: SUCCESSFUL           :
DISCONNECT 2 DUMP ..: SUCCESSFUL           :

PRESS ENTER TO CONTINUE
```

## OS/390 CICS and COMMIT and ROLLBACK functions

For OS/390 CICS, MQ\_SAMPLE bypasses COMMIT and ROLLBACK functions 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:

1. Receive the Trigger Record by calling the MANTIS Interface MQTM (For OS/390, 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.
2. Using the MQCONNECT Interface, connect to the queue containing the message to which this program was triggered.
3. Retrieve the triggered message via the appropriate MQGET Interface. Each message type will require its own Interface that is modeled after the MQGET Interface.
4. Disconnect from the queue via the MQDISCONNECT Interface.
5. Take appropriate action by connecting, sending, or receiving other messages.

### MEMADDR argument to MQ\_HANDLER

OS/390 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 "[OS/390 MQSeries/MANTIS trigger considerations](#)" on page 82.

### 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 queues and definitions are set up properly, MQSeries will execute MQ\_HANDLER automatically when you interactively run MASTER:MQ\_TRIGGER.

---

## MQ\_TRIGGER

### Introduction to 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](#)” on page 90.

---

### 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 **      :
MESSAGE RECV STATUS : SUCCESS                               :

PROGRAM COMPLETE - PRESS ENTER TO CONTINUE

```

## Defining the trigger and associated initiation queue for the MQ\_TRIGGER and the MQ\_HANDLER programs

### For UNIX

**Procedure.** Define the following for the MQ\_TRIGGER and MQ\_HANDLER programs:

- ◆ Trigger
- ◆ Associated initiation queue

To accomplish this, use the sample UNIX MQSC definitions below.



---

For the APPLICID parameter, specify your installation's full path to the MANTIS front-end application.

---

**Sample UNIX MQSC definitions.** See the following sample definitions:

```
DEFINE PROCESS(MANTIS) REPLACE APPLICID('/mantis/mqs/trigger.sh') +
DESCR('MANTIS TEST TRIGGER HANDLER') +
USERDATA('MSG FROM MANTIS MASTER:MQ_TRIGGER')

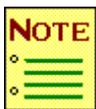
DEFINE QLOCAL('TRIGGER.QUEUE') REPLACE TRIGGER PROCESS(MANTIS)+
TRIGTYPE(EVERY) INITQ(INITIATION.QUEUE) +
DESCR('MANTIS APPLICATION TRIGGER QUEUE')

DEFINE QLOCAL(INITIATION.QUEUE) REPLACE
LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE)
```

## For OS/390

**Procedure.** Perform the following:

1. Define a local queue called TRIGGER.QUEUE. Model this queue on SYSTEM.DEFAULT.LOCAL.QUEUE. You can define this queue in one of two ways:
  - ◆ **Using the MQ ISPF Panels.** This is the preferred method.
  - ◆ **Using CSQUTIL.** You can only use this method for defining a local queue if security settings permit you to access the MVS MQSeries command program. If you select this method, modify the sample [UNIX MQSC definitions](#), listed in the previous section, to conform to the settings in step 2, below.
2. In the TRIGGER.QUEUE local queue, use the following settings:
  - ◆ Permit shared access = Y
  - ◆ Default share option = S
  - ◆ Put enabled =Y
  - ◆ Get enabled =Y
  - ◆ Trigger Type = EVERY
  - ◆ Trigger Set =YES
  - ◆ Process Name = MANTIS.TEST.MMQT
  - ◆ Initiation Queue = INITIATION.QUEUE
3. Use CSQUTIL or the MQ ISPF Panels to define an initiation queue called "INITIATION.QUEUE". You can model it after SYSTEM.DEFAULT.LOCAL.QUEUE.
4. Define a PROCESS with the following settings:
  - ◆ ProcessName = MANTIS.TEST.MMQT



---

You can set the ProcessName to any text you want. However, the ProcessName setting must match the "Process Name" for TRIGGER.QUEUE

---

- ◆ Application ID = MMQT
- ◆ UserData = "MSG FROM MANTIS MASTER:MQ\_TRIGGER"

**Additional step required if you will be using MQSeries sample programs.** In addition to the preceding MQSeries definitions for OS/390, you must use the following CICS definitions in order to use any of the MQSeries example programs:

```
MQ Interface program:
NAME=CSOXWMS,LANGUAGE=ASSEMBLER,RELOAD=NO,RESIDENT=NO

MANTIS front-end program for triggering:
NAME=CSOXTRIG,LANGUAGE=ASSEMBLER,RELOAD=NO,RESIDENT=NO
MANTIS transaction to be started by xxxx (cics trigger program name here):  NAME=MTRG
(suggested), PROGRAM=CSOXTRIG,TWASIZE=3240
```

## 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 “[MQSeries/MANTIS diagnostic considerations](#)” on page 95.
- ◆ **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:

1. Check the Interface name for returned values.

For detailed descriptions of these return values, see [“Errors”](#) on page 31.

2. 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 [“Fundamental usage”](#) on page 25.

3. Review the REASON field in the Interface layout.

The REASON field contains a number, returned by MQSeries, that further describes the problem. For more information on this number:

- ◆ 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 Guide*, SC33-1673.

For more information about the REASON field in the Interface views, see “[Fundamental usage](#)” on page 25.

4. 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 the following section, “[Dumping MQSeries Interface views](#)” on page 97.

## Dumping MQSeries Interface views

### Introduction to the “DUMP” value

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.

**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  CALL MQCONNECT
280  STOP
290  END

```

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

1. Initializes the MQCONNECT Interface by performing the following:
  - A. Specifying "INITCONN" in the FUNCTION field.
  - B. Properly setting the remaining connection fields.
2. Resets the FUNCTION field to "CONNECT".
3. Calls MQCONNECT to connect to the SYSTEM.DEFAULT.LOCAL.QUEUE.
4. Once control is returned, checks the Interface CALL for errors by comparing the Interface name to "".
5. If an error occurs, sets the FUNCTION field "DUMP".
6. Calls MQCONNECT again to dump the Internet layout.

## System-specific dump file descriptions

Consider the dump file description that is relevant to your system:

- ◆ **UNIX dump file.** The output file containing the dump output is called `mantis.dmp` (you cannot change this name). `mantis.dmp` may or may not already exist:
  - If `mantis.dmp` already exists—MANTIS appends the dump to the end of it. The user must have write privileges to the file.
  - If `mantis.dmp` does not already exist—MANTIS creates it in the current directory. The user must have write privileges to the current directory.
- ◆ **OS/390 CICS dump file.** The MQSeries Interface opens a Transient Data Queue called `mmqd` (you cannot change this name). You can associate `mmqd` with any file name, but it must have 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

## **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.

### **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

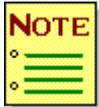
## General UNIX and OS/390 considerations

### 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/libmq5. This enables MANTIS to load the MQSeries internal Interface.



For information on setting MANTIS\_SHRLIB, refer to *MANTIS Administration OpenVMS/UNIX*, P39-1320.

- ◆ **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.

## OS/390

Consider the following for MQSeries installation under OS/390:

- ◆ **Enabling the MQSeries feature.** In OS/390, 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.
- ◆ **“C” runtime.** Cincom ships the two “C” runtime libraries with MANTIS. Perform the following:
  1. Concatenate the two “C” runtime libraries in your CICS start JCL, under the DFHRPL DD statement:

```
// DD DISP=SHR,DSN=SYS3.SAS700C.CICSLIB 'C' Runtime
// DD DISP=SHR,DSN=SYS3.SAS700C.CICSLoad 'C' Runtime
```

2. 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
```

MANTIS now dynamically loads the MQSeries entry points that are called from CSOXWMQS, enabling you to install new MQSeries releases without having to upgrade MANTIS.



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 OS/390 Concepts and Planning Guide, GC34-5650*
- ◆ *MQSeries for OS/390 System Setup Guide, SC34-5651*
- ◆ *MQSeries for OS/390 System Administration Guide, SC34-5652*

---

# 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

## OS/390

Consider the following for the OS/390 implementation of the MQCONNECT Interface:

- ◆ **Connections under CICS.** Under CICS, only one Queue Manager may be connected to at one time. However, multiple objects controlled by that queue manager may be connected to concurrently.
- ◆ **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.

---

# MQDISCONNECT

## UNIX

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

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

## OS/390

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

- ◆ It has no restrictions.
- ◆ It conforms to the routine requirements for:
  - 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.

### OS/390

Consider the following for the OS/390 implementation of the MQEXIT Interface:

**MQMD fields not used by the MQGET Interface.** Consider that for OS/390, several MQMD (Message Descriptor) fields do not exist. Although these fields appear in the MQGET Interface layout for OS/390, the MQSeries Interface does not use them. These fields are:

- ◆ MsgSeqNumber
- ◆ MsgFlags
- ◆ OriginalLength
- ◆ GroupId

---

# 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.

## OS/390

Consider the following for the OS/390 implementation of the MQPUT Interface:

- ◆ **MQMD fields not used by the MQSeries Interface.** For OS/390, four MQMD (Message Descriptor) fields do not exist. Although these four fields appear in the MQPUT Interface layout for OS/390, the MQSeries Interface does not use them. These fields are:
  - MsgSeqNumber
  - MsgFlags
  - OriginalLength
  - GroupId
- ◆ **MQPMO fields not used by the MQSeries Interface.** For OS/390, two MQPMO (Put Message Options) fields do not exist. Although these two fields appear in the MQPUT Interface layout for OS/390, they are not used by the MQSeries Interface:
  - RecsPresent
  - PutMsgRecFields

## **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.

### **OS/390**

Consider the following for the OS/390 implementation of the MQROLLBACK Interface:

- ◆ MQSeries MQI mqback is not supported in OS/390 CICS. Although using this Interface does not generate an error, the Interface does nothing.
- ◆ To rollback a logical unit of work containing MQSeries updates, use MANTIS RESET. For more information on MANTIS RESET, refer to *MANTIS Language, OS/390, VSE/ESA*, P39-5002.
- ◆ 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.

### OS/390

Consider the following for the OS/390 implementation of the MQCOMMIT Interface:

- ◆ The MQSeries mqcmit routine is not supported in OS/390 CICS. Although using the Interface does not generate an error, the Interface does nothing.
- ◆ To commit a logical unit of work containing MQSeries updates, use MANTIS COMMIT. For more information on MANTIS COMMIT, refer to *MANTIS Language, OS/390, VSE/ESA*, P39-5002.
- ◆ If the MANTIS application is running in Pseudo-Conversational Mode, a CONVERSE, by default, performs the following:
  1. Ends the MANTIS transaction.
  2. 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.

### OS/390

Consider the following for the OS/390 implementation of the MQCOMMIT Interface:

- ◆ The MQSeries mqbegin routine is not supported in OS/390 CICS. Although using the Interface does not generate an error, the Interface does nothing.
- ◆ Since, under CICS, all updateable MQSeries transactions are logged by default, there is no requirement for an MQBEGIN.
- ◆ For more information on COMMITting transactions and units of work, refer to *MQSERIES Application Programming Guide*, SC33-0807.

## MQEXIT

### UNIX

Consider the following for the UNIX 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.

### OS/390

Consider the following for the OS/390 implementation of the MQEXIT Interface:

The MQEXIT Interface is not supported in OS/390 CICS. Although using this Interface does not generate an error, the Interface does nothing. Maintain all open handles in the user program and disconnect them manually.

## **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 OS/390.

### **OS/390**

Consider the following for the OS/390 implementation of the MQTM Interface:

- ◆ For triggering, the MQTM record is passed to OS/390 CICS MANTIS. The MQTM Interface layout contains all MQTMC2 fields. The only field in the Interface layout that is not usable by OS/390 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 an OS/390-only field. It serves as a storage address of the MQTM record that CSOXTRIG passes to MANTIS. For more information, see [“MQSeries/MANTIS triggering”](#) on page 75.